UNIVERSITÀ DI PISA
DOTTORATO DI RICERCA IN INGEGNERIA DELL'INFORMAZIONE

POWER-EFFICIENT RESOURCE ALLOCATION IN CLOUD DATA CENTERS

DOCTORAL THESIS

Author
Giuseppe Portaluri

Tutor (s)
Prof. Stefano Giordano

Reviewer (s)
Prof. Franco Davoli
Prof. Nelson L. S. da Fonseca

The Coordinator of the PhD Program
Prof. Marco Luise

Pisa, May 2018

XXX

Abstract

Cloud Data Centers (DCs) provide computational resources and services for a huge number of customers. A great variety of applications, such as online storage, scientific computing, and web services, requires a set of resources located in DCs, which should be available when needed in a flexible and dynamic way. Therefore, Cloud providers need to manage DC resources carefully in order to satisfy as many service requests as possible to increase their own revenues. Indeed, the major challenge in DCs deployment consists in facing the rapidly increasing demand of Cloud services, which asks for a flexible and dynamic design of the Cloud. Another important aspect that should be considered is the interaction between the Cloud and the 5G network. The 5G strict requirements can be matched only integrating the Cloud functionality with the next-generation mobile networks applying the Mobile Edge Computing (MEG) and Mobile Edge Cloud (MEC) paradigms. This thesis offers a detailed overview on power-efficient techniques for DC management, and it mainly provides several contributions: first, it tackles the resource allocation problem introducing novel power-aware placement strategies for tasks and Virtual Machines (VMs) to be placed within the DC; next, it considers the DC networking part showing several load balancing and resilience mechanisms for Software Defined DCs; then, it outlines a topology-aware scheduler for distributed applications using the Message Passing Interface (MPI) library, that is used to avoid network congestion and minimize the overall communication length among all the processes. Finally, the thesis focuses on the Internet of Things (IoT) scenario, where devices produce data that can be consumed by softwares (i.e., tasks or VMs) typically running in the DC. More in detail, this thesis deals with the communication among IoT devices using the Time Synchronous Channel Hopping (TSCH), and it presents a centralized scheduler able to meet both real-time constraints and communication deadlines. The results show that the developed techniques improve physical resource utilization in DCs reducing the power costs and increasing the return on investments (RoI): resource allocation is performed using multi-objective techniques which optimize both power consumption and DC performance by computing the allocation pattern of thousands of VMs and tasks in less than ten seconds. Load balancing and resilience techniques provide robustness and high resource availability for Software Defined DCs while the MPI scheduler performs the communication between processes avoiding network congestion and exploiting the communication parallelism. Conclusively, it will be shown that in the IoT scenario results are promising: the implemented scheduler significantly reduces the number of violated deadlines with respect to other classical schedulers.

List of publications

International Journals

1. Portaluri, G., Adami, D., Gabbrielli, A., Giordano, S., and Pagano, M. (2017, July). Power Consumption-Aware Virtual Machine Placement in Cloud Data Center . *IEEE Transactions on Green Communications and Networking* . IEEE.

International Conferences/Workshops with Peer Review

1. Ojo, M., Giordano, S., Portaluri, G., and Adami, D. (2017, December). Throughput Maximization Scheduling Algorithm in TSCH Networks with Deadline Constraints. *Globecom Workshops (GC Wkshps), 2017 IEEE*. IEEE.

2. Portaluri, G., Adami, D., Giordano, S., and Pagano, M (2017, November). A Novel Allocation Strategy for Virtual Machines in Software Defined Data Center. *NFV-SDN Workshops, 2017 IEEE*. IEEE.

3. Pinna, A. S., Portaluri, G., and Giordano, S. (2017, July). Shooter localization in wireless acoustic sensor networks. *Computers and Communications (ISCC), 2017 IEEE Symposium on* (pp. 473-476). IEEE.

4. Ojo, M., Giordano, S., Portaluri, G., Adami, D., and Pagano, M. (2017, May). An energy efficient centralized scheduling scheme in TSCH networks. *Communications Workshops (ICC Workshops), 2017 IEEE International Conference on* (pp. 570-575). IEEE.

5. Portaluri, G., Adami, D., Gabbrielli, A., Giordano, S., and Pagano, M. (2016, December). Power Consumption-Aware Virtual Machine Allocation in Cloud Data Center. *Globecom Workshops (GC Wkshps), 2016 IEEE* (pp. 1-6). IEEE.

6. Adami, D., Giordano, S., Pagano, M., and Portaluri, G. (2016, October). A novel SDN controller for traffic recovery and load balancing in data centers. *Computer Aided Modelling and Design of Communication Links and Networks (CAMAD), 2016 IEEE 21st International Workshop on* (pp. 77-82). IEEE.

7. Portaluri, G., and Giordano, S. (2016, October). Multi Objective Virtual Machine Allocation in Cloud Data Centers. *Cloud Networking (Cloudnet), 2016 5th IEEE International Conference on* (pp. 107-112). IEEE.

8. Adami, D., Gabbrielli, A., Giordano, S., Pagano, M., and Portaluri, G. (2015, December). A fuzzy logic approach for resources allocation in cloud data center. *Globecom Workshops (GC Wkshps) 2015* (pp. 1-6). IEEE.

9. Portaluri, G., and Giordano, S. (2015, October). Power efficient resource allocation in cloud computing data centers using multi-objective genetic algorithms and simulated annealing. *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on* (pp. 319-321). IEEE.

10. Portaluri, G., Giordano, S., Kliazovich, D., and Dorronsoro, B. (2014, October). A power efficient genetic algorithm for resource allocation in cloud computing data centers. *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on* (pp. 58-63). IEEE.

List of Abbreviations

A-ITRA Analytic ITRA. 41, 42, 46, 51

API Application Program Interfaces. 2, 5, 12

ASN Absolute Slot Number. 81

BF Best Fit. 68, 91, 92

BW Bandwidth. 27

DC Data Center. 1–3, 5, 12–17, 19–21, 27, 30, 56, 62–64, 66, 91, 92

DVFS Dynamic Voltage and Frequency Scaling. 14, 16

EWF Enhanced Worst Fit. 92

F-ITRA Fuzzy ITRA. 42

FC Fixed Cost. 67, 68

GA Genetic Algorithm. X, 6–8, 14, 80, 84, 86–90, 92

HPC High Performance Computing. 12

IoT Internet of Things. 2, 3, 5, 79, 92

IPS Instructions per second. 19, 21

IT Information Technology. 1, 2, 13, 39, 41

ITRA IT Resource Allocator. 39

M2M Machine-to-machine. 2

MAC Medium Access Control. 2, 81

MEC Mobile edge computing. 2

MILP Mixed Integer Linear Programming. 12, 14, 29

MOGA Multi-objective Genetic Algorithm. 4, 6, 15, 21, 24, 30, 31, 91

MPI Message Passing Interface. 4, 12, 62, 73, 74, 92

NIC Network Interface Controller. 62

NSGA-II Non-dominated Sorting Genetic Algorithm II. 9, 20, 21, 24, 30, 31, 86

NSH Network Statistics Handler. 64, 67

PF Pareto Front. 9, 11, 12, 20, 24

QoE Quality of Experience. 1, 92

QoS Quality of Service. 1, 14, 62, 79, 92

RA Reference Area. 74–76

RoI Return on Investment. 5, 91

RP Recovery Path. 65, 68

RPF Reference Pareto Front. 9, 34

SA Simulated Annealing. 4, 10–12, 15, 21, 24, 91

SDN Software Defined Networking. 2, 4, 5, 62–65

SLA Service Level Agreement. 14

SPF Shortest Path First. 67

ToR Top of the rack. 15, 17, 18

TSCH Time Slotted Channel Hopping. X, 79–83, 87, 89, 92

TSMP Time Synchronized Mesh Protocol. 79

VM Virtual Machine. 2, 3, 5, 12–15, 27, 29–31, 34, 39, 66, 67, 91, 92

VNF Virtual Network Function. 5

WF Worst Fit. 68, 91, 92

WP Working Path. 65, 68

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Cloud data centers and 5G

In the last few decades, Data Centers (DCs) have rapidly evolved not only in terms of hardware resources and services, but also from the architectural point of view. Indeed, many and various services are made available to users, from online storage to a variety of applications. All these services require resources that are located "somewhere" in the Cloud. Users do not care where they are, they only care them to be available when needed matching the desired quality and security levels. To satisfy the rapidly growing requirements of the users, a new powerful Cloud Computing architecture has emerged, where users may ask and receive information and services dynamically through the network, from an abstract resources layer. With the demands of Cloud services continuously increasing, adopting a flexible and dynamic design of the Cloud, able to meet the complex requests of the modern era of global information, has become a major challenge for DCs designers.

On the one hand, the types of services offered by Cloud providers are evolving; on the other, the internet architecture is significantly changing as well. The 5th generation of mobile networks (5G) is currently one of the main subjects under investigation in the Information Technology (IT) field. The 5G standard outlines precise and strict requirements the network has to match in terms of Quality of Service (QoS) and Quality of Experience (QoE): the official ITU draft [8] defines the minimum requirements for three usage scenarios for mobile, low-latency, and machine-type communications. For instance, the minimum peak data rate in a single mobile station will be 20 Gbps in downlink and 10 Gbps in uplink. As the data rate increases with respect to the 4G networks, the user-plane latency has to be significantly reduced to 4 ms for the mobile broadband scenario and to 1 ms for the ultra-reliable and low-latency communications. Since this very low end-to-end latency requirement cannot be achieved using traditional

networks, it will also be necessary to provide Cloud capabilities to the access network shifting the contents to be accessed towards the edges. Mobile edge computing (MEC) enables this feature integrating DC functionalities in close proximity to the end users [81].

One of the most relevant issues in Cloud and MEC concerns the internal organization of the IT resources (e.g. computing and network devices) within the DC since the under-utilization of the physical resources inside the DC may affect the performance, lead to customer dissatisfaction, and increase costs due to the energy and power wastage. Indeed, the whole set of devices in a DC requires a relevant amount of energy to work properly, and the optimization of energy and power consumption can reduce operational costs increasing DC provider revenues.

The Cloud DC resources are managed using an orchestrator which leverages on the Software Defined Networking (SDN) paradigm. SDN is an enabling architecture that decouples the control plane from the data plane in network devices. Switches forward packets according to a set of rules installed by the centralized controller, and they can be programmed using specific Application Program Interfaces (API). When an SDN switch processes packets of a network flow, it tries to match one of the rules already present in the forwarding table. If no rule matches, the switch sends the packet to the centralized controller; then, the controller analyzes the packet and installs new rules on the proper switches. The SDN controller enables the implementation of high-level applications able to monitor and manage physical resources through the APIs it exposes, and all the strategies discussed in this thesis are designed to be a software module of the SDN controller.

The 5G standard defines another important use case: the Machine-to-machine (M2M) communications which are complementary to MEC. An example of M2M is the communication between devices in the Internet of Things (IoT) scenario in which those devices produce data that are consumed by applications typically executed in the DC. The interaction between IoT devices is currently under investigation aiming at the optimization of the energy consumed by each device during communications using energy-efficient Medium Access Controls (MACs) and schedulers.

1.2 Thesis motivation structure

Most of the related works in the literature either neglect power consumption or do not allocate resources jointly (i.e., all together) focusing only on single resource or allocating them one at a time. This thesis introduce novel allocation strategies which take into account both the power consumed by computing and network devices, and the performance of the physical infrastructure. Furthermore, this thesis considers internal communications within the DC (i.e., the so called East-West communications) evaluating the network flow allocations and scheduling in terms of reliability and performance. In the last part, it introduces an energy-efficient communication scheduler in centralized IoT that is a complementary field in the MEC scenario. Briefly, its main contributions concern the design of:

- algorithms and multi-objective strategies for power-efficient and task and Virtual Machine (VM) allocation in DC;

- load balancing and network recovery strategies for network flows in Software Defined DCs;
- a congestion-aware flow scheduler for distributed applications;
- centralized scheduling strategies for IoT devices.

The rest of the thesis is structured as follows:

- Chapter 2 introduces the basic concepts related to multi-objective optimization and heuristics, and the tools that have been adopted.
- Chapter 3 deals with DC describing the multi-objective power-efficient and performance-aware allocation strategies for tasks and VMs;
- Chapter 4 outlines load balancing techniques and resilience mechanisms for Software Defined DC, and it presents a communication scheduler for distributed applications ;
- Chapter 5 introduces our centralized scheduler for real-time communications among IoT devices;
- Chapter 6 concludes the thesis.

CHAPTER 2

Background

This chapter introduces some important concepts that will be used throughout the thesis. First, we present the SDN paradigm that allows a flexible network management of both switches and IoT devices. Next, we describe two multi-objective optimization techniques: Multi-objective Genetic Algorithm (MOGA) that is effective in terms of execution time and quality of the solutions retrieved, and a very efficient local search technique called Simulated Annealing (SA). Then, we introduce Message Passing Interface (MPI) and collective communications for distributed applications. Finally, we describe the software tools used to perform our experiments.

2.1 Software Defined Networking

SDN [61] is an emerging, dynamic, and flexible network architecture which decouples the forwarding plane from the centralized control plane. One of the SDN advantages relies on the possibility to maintain the network-wide view updated improving the network management and configuration. The intelligence present in SDN is localized in software-based controllers while switches can be programmed through open interfaces. The SDN reference model consists of three layers: *Infrastructure layer*, *Control layer*, and *Application layer*.

Switching devices at the data plane are included in the infrastructure layer and have two main functions: the collection of network information (that is stored in local memory and flushed to the centralized controller), and the packet forwarding according to the received rules. As described above, switches are completely depleted from their "intelligence" since they are only able to apply received rules. Into the data plane, an SDN switch first identifies the forwarding rule that matches with the packet; then, it forwards the packet to the next hop according to IP or MAC address (as in legacy routers),

as well as to TCP/UDP connections, VLAN tags, and other parameters.

The control layer is a bridging layer between infrastructure and application layers, and it defines two interfaces: the one interacting with the infrastructure layer is called *south-bound interface*, and it specifies interfaces to access functions provided by switching devices possibly including network status report and packet forwarding rules import. The interface between the control layer and application layer is called *north-bound interface*; it provides service access points in various forms, for example API. Since multiple controllers could coexist for a large administrative network domain, it's possible to have another kind of communication at this layer, defining an *east-west communication*. SDN controller implements two main functions: *network controlling*, including policies imposed by the application layer and packet forwarding rules for the infrastructure layer and *network monitoring*, in the format of local and global network status.

Last layer contains SDN applications designed to fulfill user requirements that are able to access and control switching devices performing management tasks such as dynamic access control, seamless mobility and migration, server load balancing, network virtualization. Moreover, SDN applications can manipulate the underlying physical networks using the high level language provided by the control layer. Among all the possible applications, we are interested on the optimization of computing and network resources within the DC.

2.1.1 SDN and orchestration

DC providers aim to maximize the Return on Investment (RoI) which requires an efficient and adaptive utilization of the available resources [27]. Two of the possible strategies for RoI maximization involve under-utilization avoidance of the computing and networking resources, and operational costs minimization. When the number of satisfied end users grows, DC providers experience an increment of their revenues, so they aim to satisfy as many customers as possible improving the management of all available physical resources. On the other hand, optimizing the power and energy consumption choosing the proper resource usage may reduce costs significantly. In this scenario, the SDN paradigm plays a fundamental role in DC orchestration because:

- the controller has the capability to maintain a global, centralized and consistent view of the network focusing specially on the automated VMs and Virtual Network Functions (VNFs) placement [24]; and

- SDN exposes northbound interfaces that can be exploited for the interaction with applications implemented on top of the controller [53] as already reported above.

We discuss our proposals designed to improve the allocation of tasks and VMs in the DC, to better manage internal DC network traffic, and to increase the effectiveness of communications in IoT devices in the next chapters.

2.2 Multi-objective optimization

2.2.1 Multi-objective Genetic Algorithms

The allocation problem is combinatorial and non-convex, and it is a variant of the multi-objective bin packing problem which is NP-Hard. In this section, we introduce a heuristic for the optimization of multi-objective, non-linear, and constrained problems called GA or MOGA. MOGAs [37] are iterative stochastic optimization methods based on the principles of natural selection and evolution. In GAs, candidate solutions (called *individuals* or *phenotypes*) evolves toward better solutions applying the genetic operators. Each solution has a set of properties (called *chromosomes* or *genotype*) which can be recombined, randomly mutated, and altered to form a new set of solutions named *generation*. A flowchart representing the behavior of GA and MOGA is represented in Figure 2.1. During each iteration, the set of all the candidate solutions (called *population*)

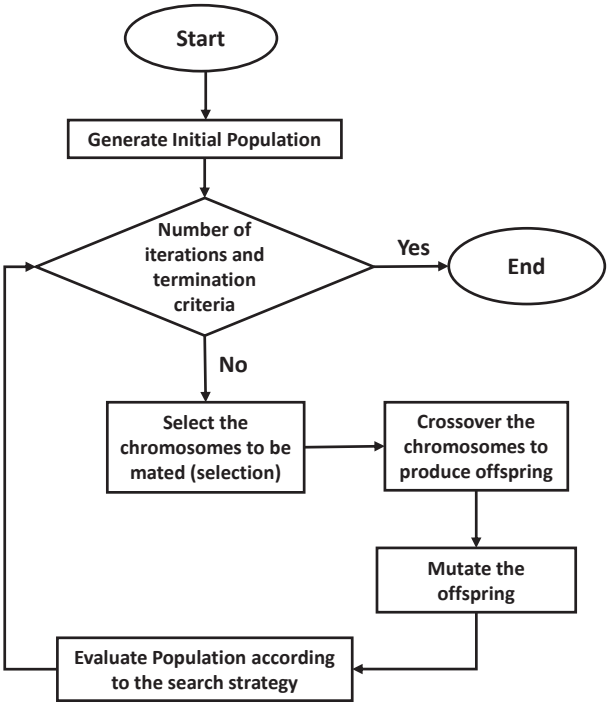

Figure 2.1: *Flow Chart for our proposed GA-based algorithm.*

is evaluated considering the fitness of every individuals applying problem-dependent *fitness functions* usually related to the objective functions. The fittest individuals are selected and survive for consequent iterations. GAs start generating an initial random population; then, during the execution of the algorithm, the population evolves across the iterations towards a better one. Each chromosome represents a solution to the problem at hand, and it can be encoded as a string of symbols, a binary bit string, a vector of numbers, or a combination of the previous structures. Each chromosome comprises a number of individual structures called *genes* describing part of the solution. The *selection operator* chooses some of the elements of the current population to produce new elements. New chromosomes are formed either by merging two chromosomes from

current generation using a *crossover operator* or by modifying a chromosome using a *mutation operator*. The set of generated chromosomes is called *offspring*. A new population is formed using a proper *search strategy* which choose the best chromosomes and rejects the others keeping the population size constant. Fitter chromosomes have higher probabilities of being selected with respect to the others. After several iterations, the algorithms converge to the best chromosome which hopefully represents the optimum or a sub-optimal solution of the problem.

Genetic Operators

Crossover The crossover operator represents one of the most important GA operators. It works on two different chromosomes at a time generating an offspring combining both chromosomes' features. As example, the crossover operator could take two chromosomes both encoded with a binary string, "cut" them in the same position (i.e., cut the related encoded binary string), and take the first part from one of the two parents and the last from the other one. According to this procedure, the generated chromosome includes parts of both parents. Several parameters influence the behaviour of all genetic operators. For instance, the *crossover probability* is the probability to apply the crossover operator to a couple of chromosomes. Denoting $pSize$ the population size and P_c the crossover probability, the expected size of the offspring is $P_c * pSize$. Other parameters depend on the implementation of the adopted crossover operator. Examples of crossover implementation are:

- *one-cut point crossover* (Figure 2.2): two chromosomes choose the same random-cut point dividing them in two parts (left and right) and generate the offspring taking the left segment from one parent and the right segment from the other one and/or vice versa.

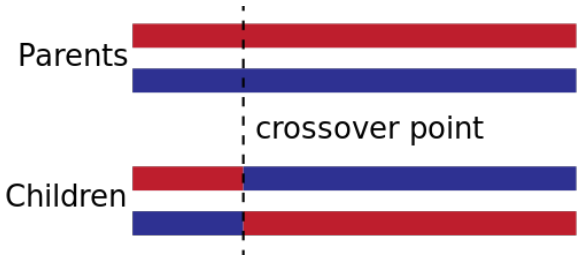

Figure 2.2: *One-cut point crossover.*

- *two-cut points crossover*(Figure 2.3) and *multi-cut points crossover* which follow the same approach from before, applying two or more cut points instead of one;

- *uniform crossover* (Figure 2.4): a fixed mixing ratio between parents is decided and the offspring is generated mixing the two parents according to that ratio in a probabilistic way; uniform crossover enables the parent chromosomes to contribute at gene level rather than at segment level.

All the crossover procedures described above never break a single gene since, every time a cut is executed, it always falls between two consecutive genes.

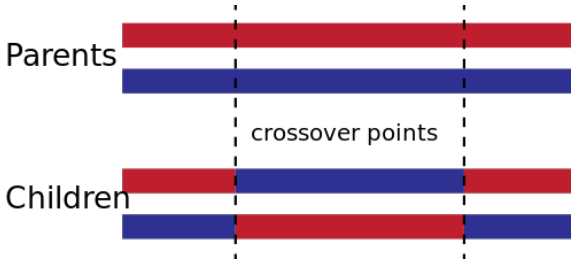

Figure 2.3: *Two-cut point crossover.*

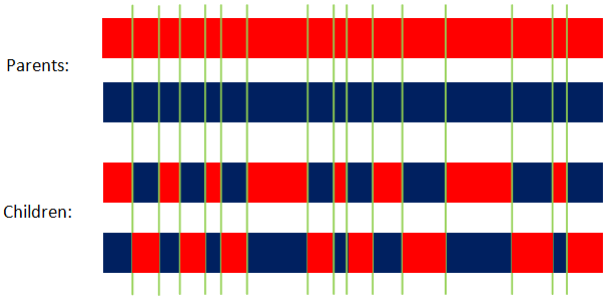

Figure 2.4: *Uniform crossover with mixing ratio 0.5.*

Mutation The mutation operator is a unary operator that produces random changes in various chromosomes accomplishing two tasks: the provision of genes missing at the beginning, and the replacement of lost genes from the population. This operator is strongly influenced by the mutation probability (P_M) which represents the probability of introducing new genes into the current population. Mutation operator with higher values of P_M produces an offspring that may differ significantly from the original parents and GA looses the possibility to "learn" from the previous history; on the other hand, some important genes could never get in the solutions for lower values of P_M. The *random mutation* changes a certain number of genes according to the value of P_M, and it is one of the simplest and widely adopted implementations of the mutation operator..

Selection The selection operator selects a subset of the chromosomes in the population that will be recombined or altered using the crossover and mutation operators, and it drives the algorithm to a certain region of the solution space. Typically, it sorts all the solutions according to their fitness value and it selects randomly all the solutions that fit better than a parametrized selection threshold. If the threshold is high (i.e., only few elements are selected), GA will focus on narrower areas of the solution space. On the other hand, lower threshold values allow the heuristic to explore wider areas of the solution space. A possible trade-off strategy is to implement a selection operator with a variable threshold that should increase during the iterations, so that the heuristic explores a wider solution region at the beginning focusing on the best local solutions only at the end.

One of the most important selection operator is called *Tournament selection*. It runs a "tournament" between a certain number of chromosomes randomly chosen from the population and selects the winner according to the fitness values; selection pressure

can be easily adjusted by changing the tournament size indeed if the tournament size is larger, weak individuals have a smaller chance to be selected. One of the most adopted implementation of the selection operator is called *Binary Tournament* having a tournament size equal to two.

Pareto Front

Multi-objective solutions are not comparable to each other because they are not commensurable. For this reason, we introduce the concept of Pareto Front (PF). Considering two solutions $\mathbf{x}$ and $\mathbf{y}$, and a set of objectives that should be minimized $f_1, f_2, ..., f_m$, $\mathbf{x}$ *dominates* $\mathbf{y}$ if exists a value k such that:

$$\begin{cases} f_k(\mathbf{x}) < f_k(\mathbf{y}); \\ f_l(\mathbf{x}) \leq f_l(\mathbf{y}) \;\; 1 \leq l \leq m \wedge l \neq k. \end{cases} \tag{2.1}$$

If (2.1) does not hold, $\mathbf{y}$ is said to be *non-dominated* by $\mathbf{x}$.The set of all feasible non-dominated solutions is called PF, and the main goal of a multi-objective optimizer is to find the best set of *Pareto-optimal solutions* optimizing the current problem.

Adopting stochastic solvers, we also refer to the definition of Reference Pareto Front (RPF) that is a PF obtained considering all solutions retrieved across different runs for the same problem. The RPF is essential for the evaluation of the computed solution quality that is quantified using three indicators PF that also assess the performance of the algorithms: the *epsilon* indicator [84] provides the factor by which an approximation set is worse than the RPF; the *spread* metric [18] quantifies how widely the observed Pareto solution set spreads over the objective space when the design objective functions are considered altogether; and the *hypervolume* computed as the volume in the objective space covered by solutions within the PF (higher values of the hypervolume metric represent a better approximation).

NSGA-II

Non-dominated Sorting Genetic Algorithm II (NSGA-II) [28] is a widely adopted search strategy for multi-objective solvers. It has a complexity of $O(FS^2)$ denoting with F the number of objectives and with S the population size. This heuristic sorts the population of solutions into different non-domination levels with a procedure called *ranking*: if a solution p dominates a solution q, then p belongs to a higher level than q. This procedure is repeated for every solution creating different groups or *non-domination levels* (solutions from the same group are non-dominating themselves); an integer value called *rank* is assigned to each non-domination level (1 is the best level, 2 is the second best level, and so on). When applying selection and sorting, NSGA-II is able to deal with constraints and unfeasible solutions. The alternatives in comparing two solutions are:

- both solutions are feasible and the one with the best fitness is chosen;
- only one solution is feasible and that one is chosen;
- both solutions are unfeasible and the one with the smallest overall constraint violation is chosen.

Using multi-objective optimization, it is necessary to introduce the definition of *constraint domination*: a solution i is said to constrained-dominate a solution j, if any of the following conditions is true:

1. solution i is feasible and solution j is not;
2. solutions i and j are both unfeasible, but solution i has a smaller overall constraint violation;
3. solutions i and j are feasible and solution i dominates solution j.

All feasible solutions are ranked according to their non-domination level which is based on the objective function values. Among two unfeasible solutions, the one with the smaller constraint violation has a better rank. When the ranking procedure is carried out, a number of chromosomes equal to the population size is taken from the best ranked solutions. If adding a rank of non-dominated solutions implies exceeding the population size, only a subset of solutions from this rank are chosen. These solutions are chosen according to the crowding distance method. The crowding distance value of a solution provides an estimate of the density of solutions surrounding it. Crowding distance of point i is an estimate of the size of the largest cuboid enclosing i without including any other point (Fig. 2.5). Boundary solutions which have the lowest and the

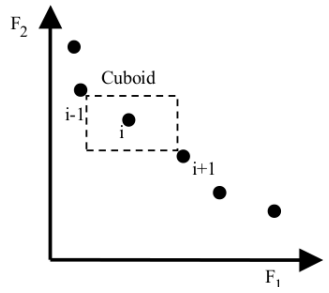

Figure 2.5: *Crowding distance for i-th solution in a two objectives algorithm.*

highest objective function values are given an infinite crowding distance. Solution A is better ranked than solution B if and only if:

$$non - dominated\, level(A) < non - dominated\, level(B). \tag{2.2}$$

or

$$\begin{cases} non - dominated\, level(A) = non - dominated\, level(B); \\ crowding\, distance(A) > crowding\, distance(B). \end{cases} \tag{2.3}$$

Once the new population is built, it undergoes the application of the selection and genetic operators to generate a number of new solutions, from which the new population will be built for the next iteration.

2.2.2 Simulated Annealing

SA is a fast and robust technique used in single and multi-objective optimization problems [69], and it is inspired by the metal cooling down mechanism. It performs a local search from an initial point in the solution space, and it moves towards a random solution present in its *neighbourhood*. Both definitions of neighbourhood and neighbour

solutions are problem-dependent. When SA explores a neighbourhood starting from an initial solution, if the chosen neighbour is non-dominated, SA moves in it, since it is preferred with respect to the solutions previously retrieved. Otherwise, the neighbour solution is either rejected or accepted with a lower probability. It is worth noting that a solution can be accepted even though it does not belong to the PF only to avoid local minimum traps. The probability of acceptance of sub-optimal solution depends on a parameter named *temperature* which is high at the beginning of the computation and monotonically decreases reaching lower values at the end, and it is computed in (2.4) being Δs the difference between the neighbour solution objective function and the current solution objective function, and T the temperature.

$$P_a = exp\left(-\frac{\Delta s}{T}\right). \tag{2.4}$$

SA avoids local traps specially at the beginning when temperature is higher but it "cools down" at the end reducing the probability to move in a less convenient state.

We adapted SA to solve multi-objective problems implementing an external archive of the best non-dominated solutions computed during the iterations. In literature, there are several mechanisms that could be used to determine the acceptance probability for multi-objective SA as described in [69]. In our problem, we compute the multi-objective acceptance probability in (2.5) denoting with d the 2-norm distance between current and neighbour solution objectives.

$$P_a = exp\left(-\frac{d}{T}\right). \tag{2.5}$$

Another important aspect for both single and multi-objective SA is the cooling scheduling procedure, i.e, how the temperature decreases among all the iterations. Some possible cooling procedures decrease linearly the temperature at each iteration while others halve or reduce geometrically the temperature value at every step [69]; in order to achieve better performances during the search strategy a fine tuning of each parameter is needed.

Last considerations concern problem constraints. SA itself does not provide any specifications about constrained problems. We adopted the following strategy for managing constraint violations: we start from the current solution and search for a feasible neighbour. If the chosen neighbour is not feasible, we look for a feasible “neighbour of the neighbour”, and we recursively repeat this procedure terminating it if any of the two following conditions is verified:

- a solution not violating any constraint is found;
- the number of neighbours explored is equal to a specific threshold.

When the second case occurs, a new solution is randomly generated, and it is compared with the solution found during the neighbourhood traversal phase which minimizes the sum of all violated constraints. Finally, the best among the two is chosen (i.e., which minimizes the sum of violated constraints) and the local search continues from that solution.

2.3 MPI

Nowadays, cloud DC infrastructure provides support for High Performance Computing (HPC) applications running on single or multiple servers. Multiple processes of the same distributed application need to adopt a communication pattern to exchange functional data; one of the most important communication patterns for parallel and distributed applications is defined by the MPI. MPI defines a set of APIs functions to allow high-performance parallel programs to exchange data between processes and complete tasks in parallel [23]. An application written using the MPI APIs is portable among different platforms.

MPI defines a set of routines that support point-to-point communications between pairs of processes for parallel and distributed applications. MPI defines also collective communications that may involve multiple senders and/or multiple receivers that are translated in a set of unicast communications. Usually, collective communications do not exploit the underneath physical topology since MPI by default does not perform a topology-discovery during the initialization phase. In the following chapters, we present a topology-aware scheduling of MPI collective communications for distributed applications.

2.4 Software tools

In this section we briefly introduce all the software tools that we used to perform our experiments.

jMetal jMetal is a Java-based framework mainly focused on development, experimentation and study of metaheuristics for solving multi-objective optimization problems [32]. This framework includes a number of classic and modern state-of-the-art optimizers and genetic heuristic implementation, and it computes automatically all the PF quality indicators.

CPLEX ILOG OPL-CPLEX Development System is a rapid development system for optimization models adopting a high-level interface to embed models for solving Mixed Integer Linear Programming (MILP) problems [44].

SA framework We use a self-programmed solver for multi-objective SA. It is flexible and allows the customization of the cooling procedure, constraint management strategy, and objective number.

DC simulator We adopt a customized simulator to perform the dynamic VM allocation. The simulator keeps track of the whole DC topology and resource utilization placing the VM according to an allocation strategy that can be easily programmed.

CHAPTER 3

Resource Allocation in Data Center

One of the main challenges in cloud computer industry is DC energy efficiency and scalability. Total power consumption due to DC, in a country such as United States, doubled in 6 years (from 2000 to 2006) reaching nearly 61 billion kW h or 1.5% of total US electricity consumption [34]. For every watt delivered [19] only 30% is consumed by the IT equipment, 33% are spent in chillers, 18% in uninterruptible power suppliers, 9% for computer room air conditioning, 5% in power distribution units, 3% in humidifiers, 1% for lightning and 1% in transformers. Even though the fraction of power consumed by IT devices is limited to one third of the total amount, the power optimization of IT devices may significantly increase DC provider revenue.

This chapter deals with power-efficient placement of software containers (i.e., tasks or VMs) onto the available physical resources. First, we consider separately two placement strategies: the *static* allocation strategy which optimizes the placement of a group of software containers, and the *dynamic* allocation strategy that places a single task or VM as soon as an external client demands for its allocation. The two strategies are better characterized in the next sections. We furthermore distinguish between task and VM allocation as follows:

- static task allocation is presented in Section 3.2;
- static VM placement is discussed in Section 3.3;
- dynamic VM allocation is analyzed in Section 3.4;

3.1 Related Works

A variety of published works focuses on the problem of resource allocation in Cloud DC. This section classifies the literature in three main parts: works that do not consider

power consumption, power-aware static allocators, and power-aware dynamic allocators.

3.1.1 Power-unaware allocators

Authors in [78] perform the initial VM placement exploiting two different approaches: GA and fuzzy logic minimizing the resource wastage without meeting any network requirement and having a high completion time. In [30], authors present an optimization technique designed for VM allocation minimizing the costs and guaranteeing some performance levels. The allocator presented in [39] combines both GAs and MILP to optimize the allocation of VMs while it meets four different QoS requirements. In [59] authors allocatesVM minimizing the maximum execution time, using three different models showing results only for a limited number of virtual machines and physical nodes. In [15] authors propose a resource allocator for distributed clouds that is based on five approximation algorithms able to reduce inter-cloud and inter-rack network traffic, but they do not consider any computational requirement for VMs only constraining the number of VMs that could be allocated on each server. Authors apply in [72] the Hungarian method to optimize the resource penalty during each VM allocation.

3.1.2 Power-aware static allocators

In [75], authors developed a particular kind of Service Level Agreement (SLA) called Green SLA. Green SLA uses best effort scheduling, which minimizes task execution time and energy-performance trade off. This approach implements a number of advanced power management strategies such as Dynamic Voltage and Frequency Scaling (DVFS) and supports parallel execution. Authors in [16] implement a power-aware VM allocator for groups of VMs using a modified version of the Multiple Knapsack problem. Another allocation method based on Fuzzy Logic is presented in [78] that performs the VM placement combining GA and Fuzzy Logic, but it has a high completion time. VMPlanner [34] is a VM and traffic flows allocator able to reduce DC power cost by putting in sleep mode network elements. VMPlanner solves the VMs allocation problem with three different algorithms which use approximation and are not scalable due to only a limited number of switches and racks that could be considered. In [78], the authors propose a system which maps VMs to physical resources using genetic algorithm improved with fuzzy multi-objective optimization. This approach tries to reduce the amount of power consumed by the servers, optimize CPU and memory utilization, and minimize peak temperatures inside the facility. In [74], authors introduce a power-aware VM allocation procedure for DCs, but they neglect both the path allocation and the power consumption of the network. Another approach, presented in [58], is able to allocate tasks with the objectives of minimizing the longest task completion time while optimizing the energy efficiency at the same time in heterogeneous Grid systems composed of multi-core processors. However, that work does not take into account the network requirements. Authors in [36] present two exact algorithms for energy efficient VM scheduling. In [17], authors suggest a heuristic optimization algorithm based on particle swarm, but they suppose that the number of VM requests is much lower than the number of physical servers, and they disregard the power consumption of network devices. In [36], two exact algorithms for energy-efficient scheduling of VMs are

compared, but all VMs have the same requirements, and the network resources are not taken into account.

3.1.3 Power-aware dynamic allocators

The approach, presented in [55], manages dynamic resource re-allocation in DCs using a multi-agent version of the fuzzy controller. Only CPU and RAM are taken into account, while disk and network requirements are neglected. Authors present a dynamic allocator in [77] that neglects the power consumption due to the networking devices. The allocator presented in [79] dynamically allocates VMs putting in sleep mode underutilized servers. In [20] the authors propose another energy efficient dynamic allocator for VMs. It is implemented using a modified version of the Best Fit Decreasing algorithm to allocate VMs using their utilization factors. This approach allows to dynamically reallocate VMs and supports also heterogeneous hardware. Finally, in [42], authors describe a dynamic VM allocator addressing the problem of energy-efficient task allocation in the system in the presence of a time-varying grid energy price and the unpredictability and time variation of provisioned power by renewable energy sources, but they neglect the path allocation phase.

3.2 Static Task Allocation

In this section, we present our static allocation strategy for tasks. We optimize task placement using an allocator based on MOGA; then, we furthermore compare the quality of the solutions computed using our allocation strategy with another one based on a multi-objective implementation of SA.

3.2.1 System Model

Large scale DCs are usually structured in a three-tier fat-tree architecture [21] (see Fig. 3.1). This topology is composed by three different layers: *access*, *aggregation*, and *core*. The access layer provides the connections of servers, which are arranged into racks, to a single Top of the rack (ToR) switch. The aggregation layer provides redundant connection of ToR switches; each ToR switch is connected to a pair of more powerful aggregation switches. A set of ToR switches and servers connected to the same pair of aggregation switches is named *pod*. At a higher layer, core switches guarantee connection among different pods and towards the external gateway. In our simulation environment, a single rack contains of up to 24 servers and a pod includes up to 8 racks. The link interconnecting server and ToR switch has capacity of 1 Gbps while ToR and aggregation switches are connected by a pair of links having capacity of 10 Gbps each.

The proposed algorithm allocates in the DC, which is empty at the beginning, a set of independent *tasks* which are characterized by:

- a number of instructions to be executed;
- a constant amount of bandwidth required to perform the execution.

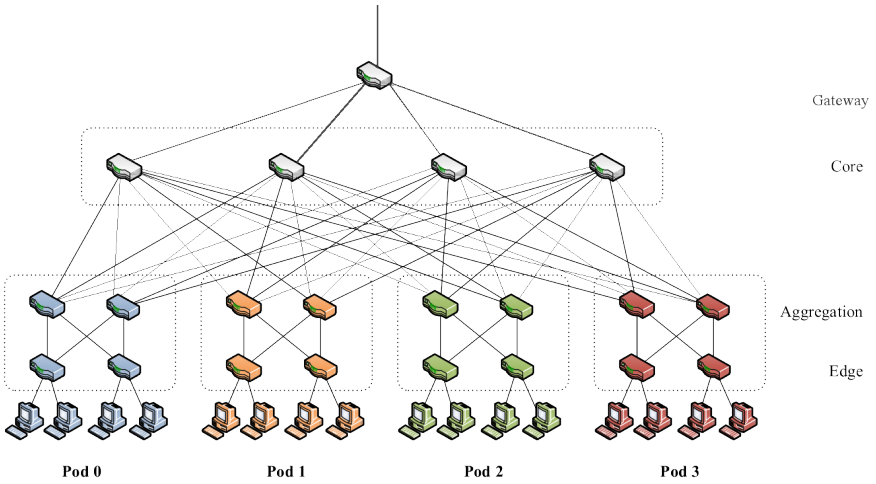

Figure 3.1: *Fat-tree topology in data centers.*

Table 3.1: *Symbols related to tasks and servers.*

Symbol	Description
N	task number
M	servers
$\#ins_i$	the total number of instruction of task i-th;
B_i	the i-th task bandwidth requirement
IPS	instruction per second executed by CPUs
n_j	tasks allocated on j-th server
T_c	the maximum completion time between all the tasks
I_j	instructions to be executed on j-th server
T_j	the j-th server completion time

As all tasks are independent, they do not require to communicate during their execution and the bandwidth requirement is only related to communication between computing servers and DC gateway.

Computing servers are modeled with single-core processors offering a fixed computational power expressed in instructions per second. The tasks allocated for execution on the same server will share server's processing power equally. The servers left idle can be put into a sleep mode minimizing their power consumption. Two power saving strategies are typically implemented on hardware: DVFS and dynamic power shutdown. DVFS reduces power consumption by lowering down operating frequency and/or voltage; on the other hand, dynamic power shutdown saves power during idle times by powering down as much as possible all the sub-components. We suppose the aforementioned approaches applied in both computing hardware and network switches, as a result we model the power consumption of servers and switches as described in (3.6) and (3.8) respectively. As first approximation, we consider servers to be either idle or fully utilized, and devices left idle are turned off to save power. We denote all the symbols adopted in this section for tasks and servers, power consumption, and network topology in Table 3.1, Table 3.2 and Table 3.3 respectively.

In our optimization problem, we encode solutions as an integer vector of size equal to the number of tasks. The value assigned to every position of the chromosome rep-

Table 3.2: *Symbols related to network topology.*

Symbol	Description
S	switches
R	racks
C_i	the throughput of the i-th link
C_{MAX}	the sum of maximum capacity of all uplinks
V_{Lj}	the bandwidth utilization on j-th server link
V_{Tk}	the bandwidth utilization on k-th ToR switch
$\#link_k$	the number of uplinks in the k-th switch

Table 3.3: *Power symbols.*

Parameter	Value
P_{SRP}	peak power consumption for servers
P_{SWP}	peak power consumption for switches
P_{SWI}	idle power consumption for switches
P_{SRj}	the power consumption of the j-th server
P_{SR}	the total server power consumption
P_{SWk}	the power consumption of the k-th switch
P_{SW}	the total switch power consumption
P	the global power consumption

resents a server where the corresponding task is allocated to (this value ranges from 1 to the number of servers). As introduced in Section 2.2.1, the allocation problem is a NP-Hard, non-convex, and multi-objective minimization problem; more specifically, the two objectives to be minimized are:

1. Maximum total completion time of all tasks (*makespan*);
2. Power consumption of servers and network switches.

The first objective is directly related to the performance of the whole DC, while the second objective to the device power consumption. We associate the fitness functions in (3.1) with the makespan and power consumption respectively, and we compute the completion time of a single server considering the sum of all instructions to be executed on the same server and dividing it for IPS as described in (3.2), (3.3) and (3.4). Then, the total completion time is the maximum completion time among all the server as in (3.5).

$$\begin{cases} f_1 = T_c; \\ f_2 = P. \end{cases} \tag{3.1}$$

$$x_{ij} = \begin{cases} 1, & \text{if the i-th task is allocated on the j-th server;} \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

$$I_j = \sum_{i=1}^{N} x_{ij} \, \#ins_i. \tag{3.3}$$

$$T_j = \frac{I_j}{IPS}. \tag{3.4}$$

$$T_c = max\ _{1 \leq j \leq M}\ T_j. \tag{3.5}$$

The power consumption of computing servers is modeled using binary law, assuming that each processor is either executing tasks at the full speed or stays idle as reported in (3.6). Network device power model follows a linear trend as shown in (3.8): consumption is proportional to the traffic load (that is computed in (3.7)), and it ranges between P_{SRI} (when the switch is idle) and P_{SWP} (when all the links are fully utilized). As written above, idle devices are turned off to save more power.

$$P_{SRj} = \begin{cases} P_{SRP}, & \text{if } n_j \geq 1; \\ 0 & \text{if } n_j = 0. \end{cases} \tag{3.6}$$

$$l_k = \sum_{i=1}^{\#link_k} \frac{C_i}{C_{MAX}}. \tag{3.7}$$

$$P_{SWk}(l_k) = \begin{cases} P_{SWI} + (P_{SWP} - P_{SWI})l_k & \text{if } 0 < l_k \leq 1; \\ 0 & \text{if } l_k = 0. \end{cases} \tag{3.8}$$

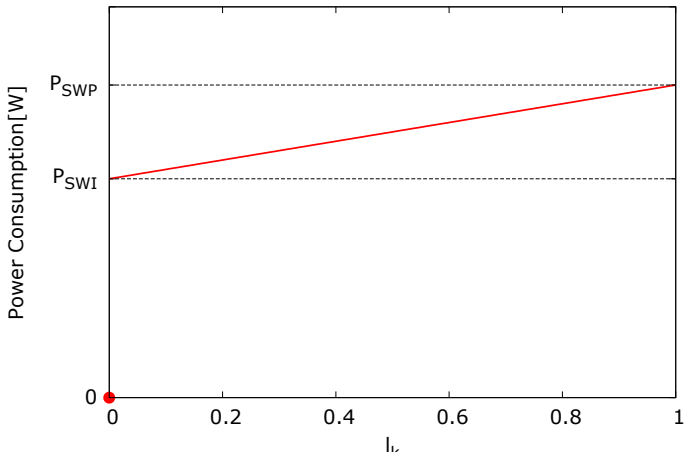

Figure 3.2: *Network switch power consumption.*

Our problem is subject to two different constraints: it is not possible to allocate on a single server tasks demanding higher bandwidth than the available link capacity (1 Gbps), and the 20 Gbps uplink capacity of ToR switches cannot be exceeded by the sum of the traffic; the two constraints are formalized in (3.14).

$$\begin{cases} V_{Lj} \leq 1Gb/s & 1 \leq j \leq M; \\ V_{Tk} \leq 20Gb/s & 1 \leq k \leq R. \end{cases} \tag{3.9}$$

$$x_{ijk} = \begin{cases} 1, & \text{if the i-th task is allocated on the j-th server of the k-th rack;} \\ 0, & \text{otherwise.} \end{cases} \tag{3.10}$$

$$sgn(x) = \begin{cases} 0\ \forall x \leq 0; \\ 1\ \forall x > 0. \end{cases} \tag{3.11}$$

Table 3.4: *Topology, server and task related parameters.*

Parameter	Value
Servers per rack	24
Racks per pod	8
IPS	$1,2 * 10^9 [instr/s]$
Task instruction distribution	Uniform in the range $[5; 10] * 10^9$ $[instr]$
Task bandwidth distribution	Uniform in the range $[1; 1000] * 10^8$ $[bps]$

Table 3.5: *Power consumptions.*

Parameter	Value [W]
Server P_{SRP}	300
ToR P_{SWP}	200
ToR P_{SWI}	160
Aggregation P_{SWP}	2500
Aggregation P_{SWI}	2000

$$c_{j1} = \left(\sum_{i=1}^{N} x_{ij} \; B_i - 1Gbps \right) sgn \left(\sum_{i=1}^{N} x_{ij} \; B_i - 1Gbps \right). \tag{3.12}$$

$$c_{k2} = \left(\sum_{j=1}^{M} \sum_{i=1}^{N} x_{ijk} \; B_i - 20 \; Gbps \right) sgn \left(\sum_{j=1}^{M} \sum_{i=1}^{N} x_{ijk} \; B_i - 20 \; Gbps \right). \tag{3.13}$$

$$\begin{cases} c_1 = \sum_{j=1}^{M} c_{j1}; \\ c_2 = \sum_{k=1}^{R} c_{k2}. \end{cases} \tag{3.14}$$

The aforementioned definition of two constraints does not assign any penalty if network links are not congested. Whenever tasks allocated to the same server (or to the same rack) require more than available bandwidth, the exceeding amount of bandwidth is added to the constraint violations. It is important to recall that with this heuristic, solutions that are violating constraints are still considered during the search phase, but using the proper search heuristic they tend to disappear in the last generations.

3.2.2 MOGA Experimental setup

In this section we describe the executed tests with the developed algorithm and the obtained results.

Scenario

DC topology, tasks, servers and values for hardware power consumption are described in Tables 3.4 and 3.5, while the algorithm configuration setup is presented in Table 3.6. The number of servers per rack and rack per pod are typical values for a DC [20], and the number of Instructions per second (IPS) is frequent in single core processors such as Intel Pentium at 1 GHz. Task instructions number and bandwidth requirement are generated randomly using two different uniform distributions. In Table 3.5, power consumption values are reported, and we consider the switch idle power consumption to be the 80% of the peak consumption [20].

Table 3.6: *Genetic algorithm parameters.*

Parameter	Value
Population size	100
Iterations	25000
Crossover operator	One point
Crossover probability	0.9
Mutation operator	Random mutation
Mutation probability	$\frac{1}{\text{task number}}$
Selection operator	Binary tournament
Search heuristic	NSGA-II

Results

Three different experiments were performed. In the first experiment, we analyse the best solutions taking into account the two objectives separately in order to explore the two edges of the obtained PF. The second experiment shows an example of PF obtained and a superposition of different PFs obtained through different experiments using the number of servers as a parameter of interest. Finally, the third experiment shows the experimental time complexity as a function of the number of tasks to allocate. In our experiments, we suppose the two objectives to be equivalent: power consumption and completion time could be considered equally relevant, so the vector with minimal module can be considered as the best solution. On the contrary, if either makespan or power consumption would be more relevant with respect to the other objective, the two metrics could be weighted to find the preferred solution. The platform used to execute these experiments is an Intel I7 3630QM 2.4 GHz equipped with 8 GB RAM with OS Ubuntu 11.04 and the jMetal framework.

We execute the first experiment considering as input values the ones presented in Table 3.7. We show the solutions computed by the algorithm which minimize makespan and power consumption found respectively in Figures 3.3 and 3.4. As expected, the maximum completion time is linear with respect to the number of tasks; on the other hand, the trend is different for power consumption since keeping constant the number of physical devices while increasing the number of tasks, the final effect is to reach the limit of the sum of the peak power consumption for every devices into the DC. We show in Figure 3.5 the PF retrieved in a single run for the allocation of 3000 tasks in a 1536 servers DC: in this case, makespan ranges between 39 seconds and 51 seconds and the power consumption between 425 KW and 443 KW. Figure 3.6 displays the superposition of PFs computed varying the number of servers and allocating the same tasks; each execution is repeated forty times and we plot all solution retrieved. Increasing the number of servers, leads to a deprecation of the results obtained in terms of power consumption and completion time as expected. The last experiment measures the framework performance to compute the solution with a fixed number of servers (i.e., 1536) only varying the task number which is the only variable input parameter. Figure 3.7 displays the results with a trend line: average execution times are close to the quadratic trend function that fits well the measured values, so the framework has a

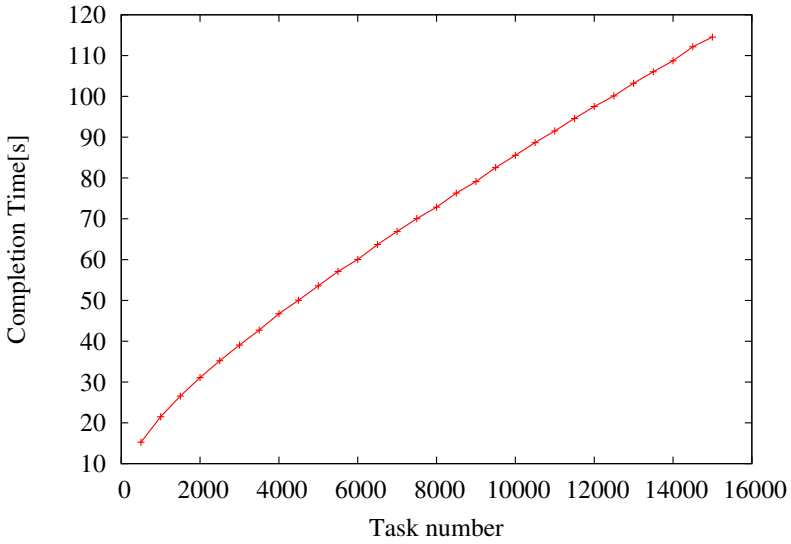

Figure 3.3: *Best task completion times.*

quadratic dependence on the number of input tasks.

Table 3.7: *Experiment 1: Input values.*

Parameter	Value [W]
Server number	1536
Task number	[500:15000] with steps of 500
Runs	40

3.2.3 SA and MOGA Experimental setup

We evaluate the effectiveness of MOGA using SA as term for comparisons. Considering a single allocation vector (i.e., a solution for our problem), we define its neighbourhood as the set of solutions which allocate all tasks on the same servers except for a single one i.e., the two vectors differ exactly for a single value. As described in Section 2.2.2, our strategy to avoid constraints violation traverses the neighbourhood of unfeasible solutions until either a feasible solution is found or a maximum number of solutions are evaluated. For our problem, we set the number of maximum evaluations equal to the number of tasks.

Scenario

DC topology, tasks, servers and hardware power consumption are described in Tables 3.8 and 3.9, setup values for MOGA and SA are presented in Tables 3.10 and 3.11. The number of IPS is frequent in processors such as Intel Core i7 875K at 3GHz; and the task instruction number and the bandwidth requirement for each task are generated randomly using two different uniform distributions. In Table 3.5, power consumption values are reported. P_{SWI} is considered to be the 80% of P_{SWP} [21]. In Table 3.6 we show the original parameters proposed for NSGA-II and two classical Crossover and

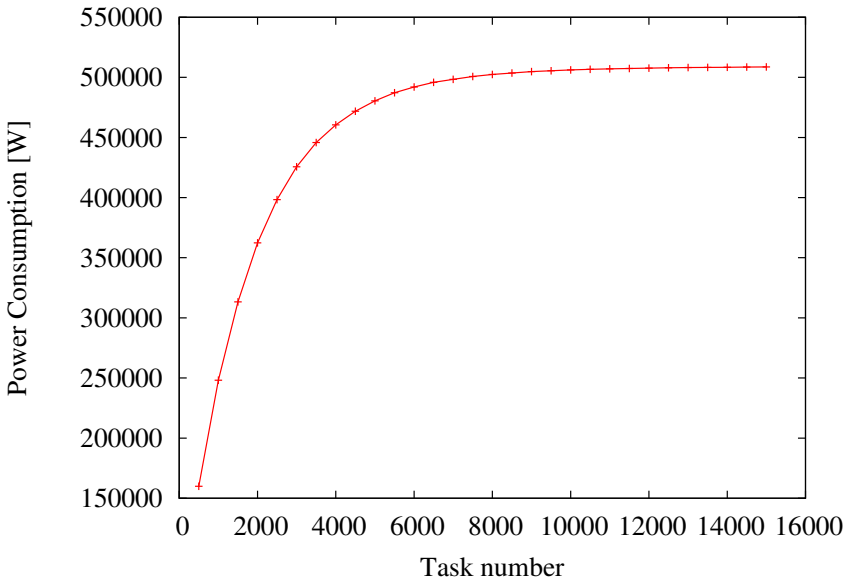

Figure 3.4: *Best power consumption values.*

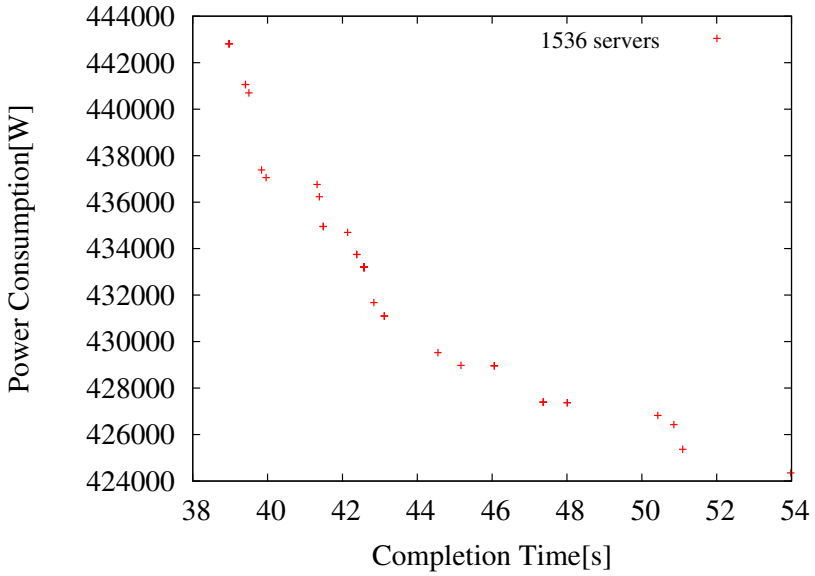

Figure 3.5: *Pareto front obtained by a single experiment.*

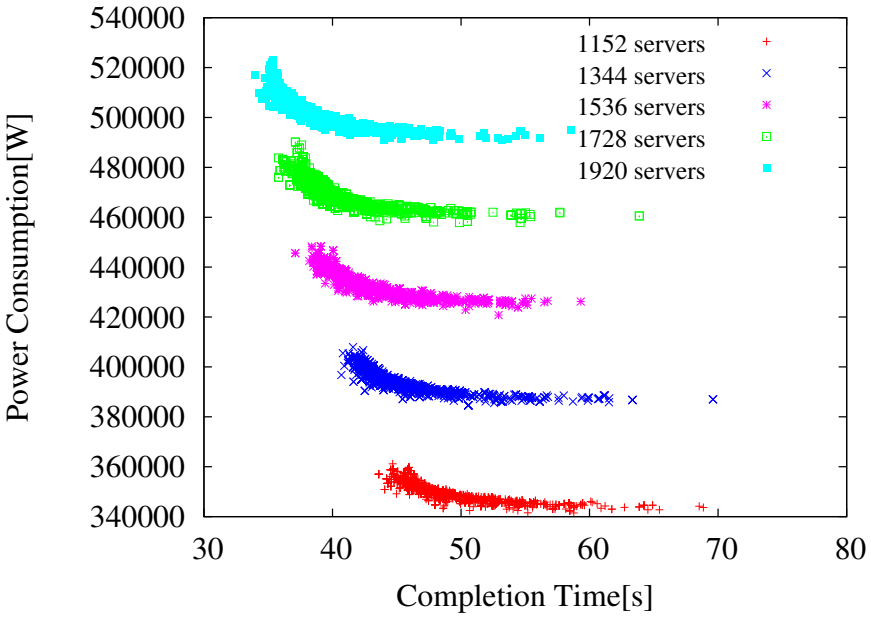

Figure 3.6: *Solution with different number of servers.*

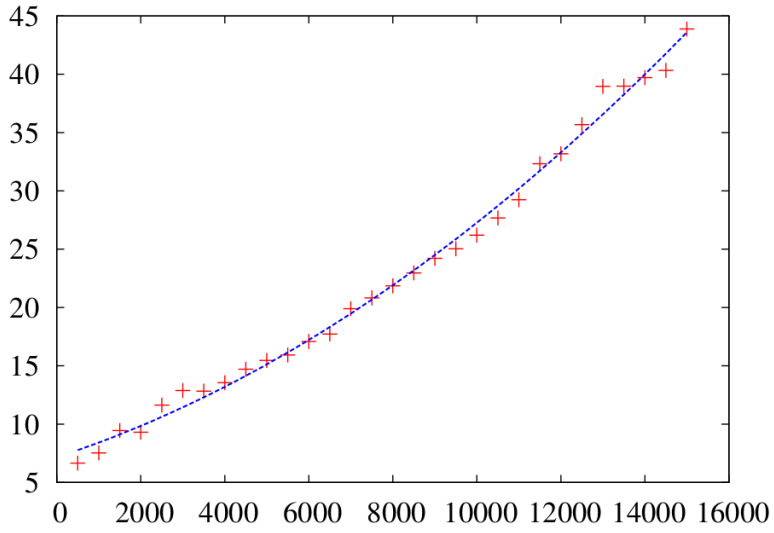

Figure 3.7: *Execution time of the algorithm.*

Table 3.8: *Topology, server and task related parameters.*

Parameter	Value
Server per rack	24
Rack per pod	8
IPS	$90 * 10^9 [instr/sec]$
Task instruction distribution	Uniform in the range $[18; 72] * 10^{10}$ $[instr]$
Bandwidth distribution	Uniform in the range $[1; 20] * 10^6$ $[bps]$

Table 3.9: *Power consumptions.*

Parameter	Value [W]
Server P_{SRP}	300
ToR P_{SWP}	200
ToR P_{SWI}	160
Aggregation P_{SWP}	2500
Aggregation P_{SWI}	2000

Table 3.10: *MOGA parameters.*

Parameter	Value
Population size	100
Iterations	25000
Crossover operator	One point
Crossover probability	0.9
Mutation operator	Random mutation
Mutation probability	$\frac{1}{\text{task number}}$
Selection operator	Binary tournament
Search heuristic	NSGA-II

Table 3.11: *SA and input parameters.*

Parameter	Value
Initial temperature	100;500; 1000;1500
Cooling scheduling	Decremental
Server number	1536
Task number	[500:20000] with steps of 500
Runs	40

Mutation operators. Table 3.11 describes the parameters for SA. As cooling scheduling, at each iteration the temperature value is decreased by one unit.

Results

The platform used to execute these experiments is an Intel I7 3630QM 2.4 GHz equipped with 8 GB RAM with OS Ubuntu 12.10, the jMetal framework for MOGA computation and a self-programmed framework for SA implementation. All experiments are performed using input values presented in Table 3.11. Figure 3.8 shows the best solutions for T_c. MOGA performs better on average with respect to SA. More in detail, MOGA finds values lower on average from 13 to 19% rather than SA1500. Figure 3.9 shows the best values for P, the difference between MOGA and SA is smaller than 6% on average. Figure 3.10 shows the PF computed on 8000 tasks including all the best solutions retrieved at each run. In it, the best values are achieved by MOGA that performs slightly better than SA1500.

Last experiment measures the time performance of the two frameworks. In Figure 3.11, MOGA execution time grows almost quadratically with the number of tasks while SA grows much slower respect to MOGA following a linear trend and bounding below 3 seconds the average execution for 20000 tasks.

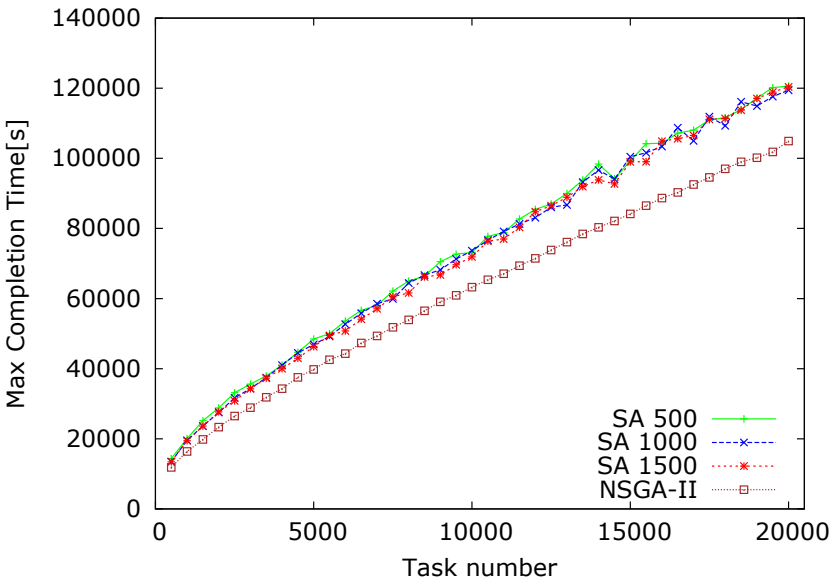

Figure 3.8: *Best T_c values.*

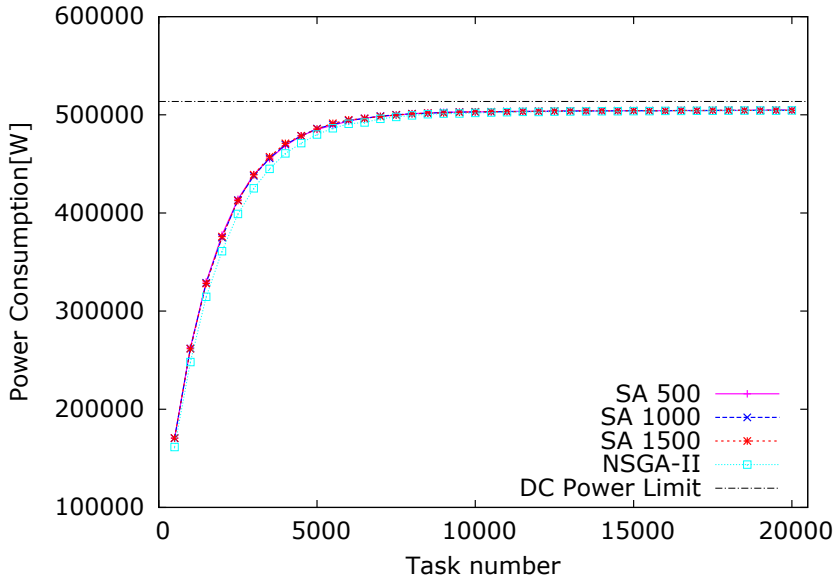

Figure 3.9: *Best P values.*

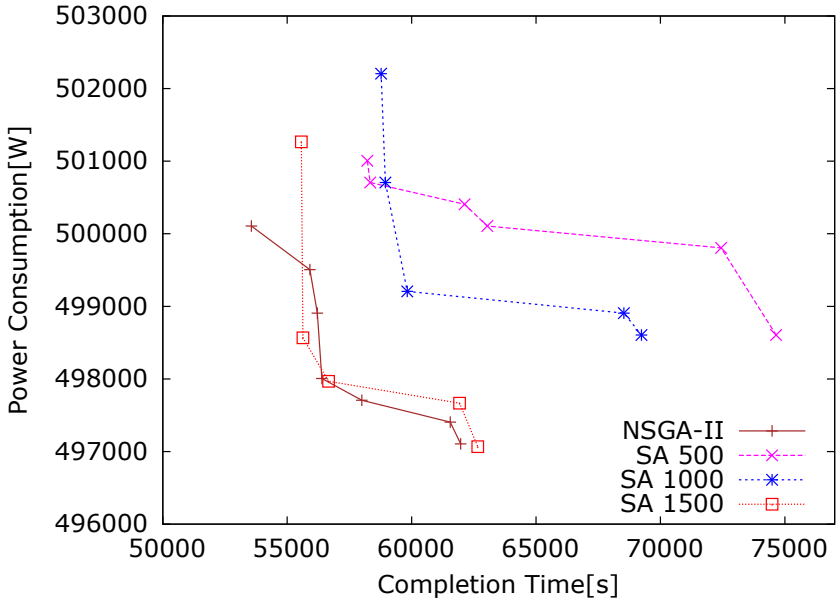

Figure 3.10: *Pareto front.*

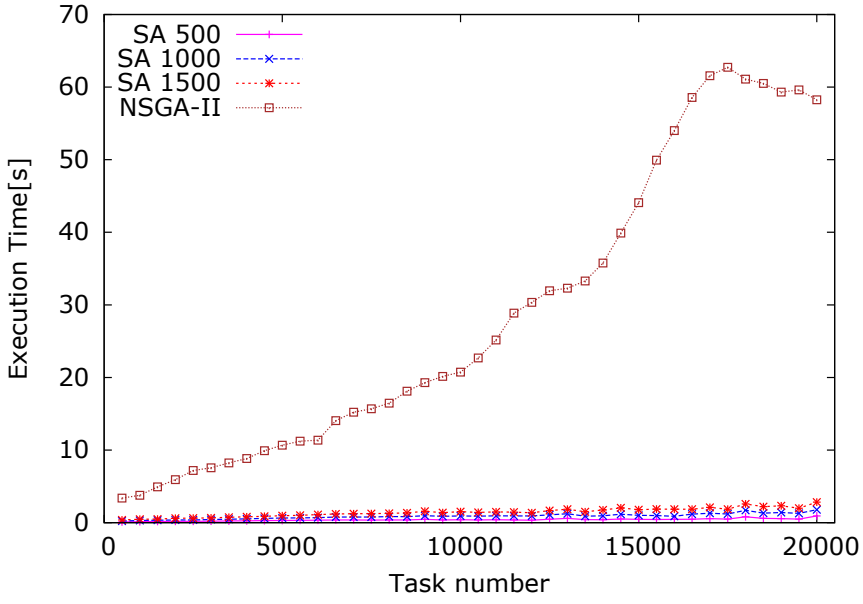

Figure 3.11: *MOGA and SA execution time.*

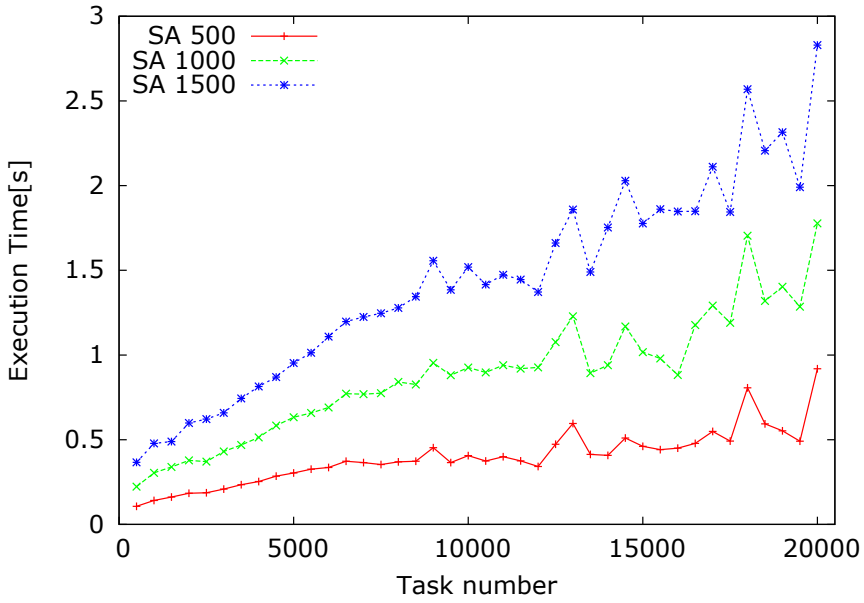

Figure 3.12: *SA execution time.*

3.3 Static VM Allocation

3.3.1 System Model

Another important aspect of resource management in DC is related to VMs allocation. VMs have different requirements with respect to tasks, for this reason we designed a new placement strategy. More specifically, each VM has four requirements: CPU, RAM, disk and North-South Bandwidth (BW) (i.e., traffic exchanged between VMs is not considered) which is typical in many VM applications such as Virtual Desktop Infrastructure. Lastly, VMs are supposed to be in a running state for a very long time and never be deallocated.

Power Consumption

The switch consumption model is the one already presented in (3.8) for task allocation. Conversely, we adapted the server power model to the new requirements: in this scenario, CPU load depends on the VMs that are allocated on the same server. Server power model follows a linear trend that is proportional to the CPU load. As for switch, server consumption ranges between two idle and peak values as described in (3.15). Server load is set equal to 100% if the sum of the CPU requirements of all VMs allocated on the same server is greater than the server availability. We define all symbols that we use in this section in Table 3.13.

$$P_{SR_i}(c(i)) = \begin{cases} P_{SRI} + (P_{SRP} - P_{SRI})\dfrac{c(i)}{100} & \text{if } 0 < c(i) \leq 100; \\ 0 & \text{if } c(i) = 0. \end{cases} \tag{3.15}$$

Table 3.12: *Symbols related to power consumption.*

Symbol	Description
P_{SRj}	the power consumption of the j-th server
$P_{SR} = \sum_j P_{SRj}$	the sum of power consumptions of all servers
P_{SRI}	the server idle power consumption
P_{SRP}	the server peak power consumption
P_{SWk}	the power consumption of the k-th switch
$P_{SW} = \sum_k P_{SWk}$	the sum of power consumption of all switches
P_{SWI}	the switch idle power consumption
P_{SWP}	the switch peak power consumption
$P = P_{SR} + P_{SW}$	the power consumption of the whole DC
Γ_i	the throughput of the i-th link
C_{MAX}	the sum of capacities of all uplinks
$\#link_k$	the number of uplinks in the k-th switch
$l_k = \sum_{i=1}^{\#link_k} \frac{\Gamma_i}{C_{MAX}}$	the load factor of the k-th switch
$c(i)$	the cpu percentage allocated on the i-th server

Table 3.13: *Symbols related to VMs and network topology.*

Symbol	Description
N	the number of VMs to be allocated
M	the total number of servers
$A \in \mathbb{R}^{N*M}$	the binary allocation matrix
e	the total over-provisioning factor or excess
$c(i)$	the CPU allocation percentage of the i-th server
$m(i)$	the RAM allocation percentage of the i-th server
$d(i)$	the disk allocation percentage of the i-th server
s	the number of servers exceeding at least one resource
P	the total power consumption of servers and switches
C_{max}	the maximum CPU allocated quota among all servers
M_{max}	the maximum RAM allocated quota among all servers
D_{max}	the maximum disk allocated quota among all servers
C_{tot}	the total CPU excess among all servers
M_{tot}	the total RAM excess among all servers
D_{tot}	the total disk excess among all servers

3.3.2 Mathematical model

Instead of considering all the resource separately, we aggregate them in a single parameter named *over-provisioning* factor in order to improve the efficiency of the algorithm. The over-provisioning factor is defined as the minimum additional quota for a specific resource (i.e. CPU, RAM, Disk) needed to satisfy all the VM requests allocated on the same server. We shortly name *excess* the maximum over-provisioning factor among all the resources and all servers. If the available resources are sufficient to fit all the VM requests, an allocation is optimal if and only if the computed excess is zero (i.e., no resource needs to be over-provisioned).

We formalize the allocation problem as a MILP problem which tries to minimize at the same time the excess and the total amount of power consumption. The problem to be solved is defined in (3.16).

$$
\begin{aligned}
&\text{minimize} && e \text{ and } P && \\
&\text{subject to:} && e \geq 0 && \\
& && P \geq 0 && \\
& && A(i,j) \in \{0,1\} && \\
& && \sum_{j=1}^{M} A(i,j) = 1 && \forall 1 \leq i \leq N \\
& && \sum_{i=1}^{N} c(i)A(i,j) \leq 100 + e; && \forall 1 \leq j \leq M \\
& && \sum_{i=1}^{N} m(i)A(i,j) \leq 100 + e; && \forall 1 \leq j \leq M \\
& && \sum_{i=1}^{N} d(i)A(i,j) \leq 100 + e; && \forall 1 \leq j \leq M
\end{aligned}
\tag{3.16}
$$

We recall the definition of the binary variable x_{ijk} previously defined in (3.10):

$$
x_{ijk} = \begin{cases} 1 & \text{if the i-th task is allocated} \\ & \text{on the j-th server of the k-th rack;} \\ 0 & \text{otherwise.} \end{cases}
$$

Our problem constrains racks and pods bandwidth allocation as follows:

$$
\begin{aligned}
&\text{constraint}_1 : \sum_{i=1}^{N} B_i x_{ijk} \leq 1\ Gbps && 1 \leq j \leq M \\
& && 1 \leq k \leq R; \\
&\text{constraint}_2 : \sum_{i=1}^{N} \sum_{j=1}^{M} B_i x_{ijk} \leq 20\ Gbps && 1 \leq k \leq R;
\end{aligned}
\tag{3.17}
$$

Since it corresponds to an initial allocation problem, for the sake of simplicity and without losing generality, we consider an empty DC (i.e., no VMs are initially already

Table 3.14: *Data Center parameters.*

Parameter	Value
Server per rack	24
Rack per pod	8
Total number of servers	768
Total number of racks	32
Total number of pods	4

Table 3.15: *NSGA-II parameter settings.*

Name	Number of iterations	Crossover probability	Population Size
NSGA100	25000	0.9	100
NSGA1000	25000	0.9	1000
NSGA095	25000	0.95	100
NSGA05	25000	0.5	100
NSGAVIT	10000*N	0.9	100

allocated); furthermore, each server has the same characteristics in terms of CPU power, RAM and Disk size. However, we modified the framework in order to read an initial state of the server occupancy and use servers with differentiated hardware. In this problem, our goal is to perform resource allocation lowering the value of e and P.

We use CPLEX as a comparison for our multi-objective solver. Since CPLEX solves only single objective optimization problems, we choose as objective to be optimized e neglecting the total power consumption P. In other words, CPLEX emulates a traditional power-unaware allocator. Another issue of CPLEX is scalability because it finds the optimal solution only for a limited input size (i.e., up to 100 VMs). For what concerns the proposed MOGA implementations, is it possible to improve the quality of the solutions retrieved for one of the two objectives adding a set of auxiliary objectives that are strongly correlated with it; we describe the sets of auxiliary objectives in Section 3.3.3.

3.3.3 Evaluation

DC topology is modeled as a three-tier fat tree and the adopted parameters are summarized in Table 3.14. We considered five different variants of NSGA-II (described in Table 3.15), tuning three parameters which mainly characterize the search strategy:

- number of iterations;
- crossover probability;
- population size.

As reported in Section 3.3.2 we designed three different problem settings having a different number of auxiliary objectives. In table 3.16 those problem settings are described. It is worth noting that NSGA-II performance is not degraded adding highly correlated or dependent objectives [46]. CPU, RAM, Disk and Bandwidth requirements are generated uniformly at random as described in Table 3.17. Switch and server power consumption values are described in Table 3.18. Top of the rack and aggregation

Table 3.16: *Search heuristic settings.*

Name	Number of Objectives	Objectives
VMProblem1 (VMP1)	3	e,s,P
VMProblem2 (VMP2)	6	e,s,P C_{max}, R_{max}, D_{max}
VMProblem3 (VMP3)	9	e,s,P C_{max}, R_{max}, D_{max} C_{tot}, M_{tot}, D_{tot}

Table 3.17: *VM generation parameters.*

Requirement	Value
N	50, 75 and [100;2000] with step size 100
CPU	
RAM	[10;40]%
Disk	
Bandwidth	[1;20] Mbps

switches have different values of power consumption. Typical value of idle power consumption for switches is about 80% of the peak consumption [21]. Parameters which characterize genetic operators are listed in Table 3.19. We performed 40 independent repetitions of the same experiment varying 22 times the input size (i.e. number of VMs to be allocated) and comparing the average execution times and the best solutions found. Due to its limited scalability, CPLEX is able to perform its execution only for 50, 75 and 100 VMs: for larger problems the number of variables and constraints becomes very high and the amount of memory needed to run the optimization engine grows over 5 GB, and the process is automatically terminated by the operating system. As it is shown in figure 3.17 and better detailed in figure 3.18, the MOGA approach solves the allocation problem two orders of magnitude faster compared with CPLEX, reducing execution time from hundreds of seconds to seconds; execution time increases linearly with the number of VMs. The exact solver finds solutions only for a very limited amount ofVMs compared to MOGA and is not useful in the current scenario that requires fast and scalable response. In Figures 3.13 to 3.15, we compare the three problem settings considering only the first objective e. As it is shown, CPLEX and MOGA find the same optimal solutions; among the different problem settings, VMP2

Table 3.18: *Power consumption values.*

Parameter	Value [W]
P_{SRI}	100
P_{SRP}	300
ToR P_{SWI}	160
ToR P_{SWP}	200
Aggregation P_{SWI}	2000
Aggregation P_{SWP}	2500

Table 3.19: *MOGA parameters.*

Parameter	Value
Crossover operator	Two points
Mutation operator	Random mutation
Mutation probability	$\frac{1}{\text{task number}}$
Selection operator	Binary tournament
Search heuristic	NSGA-II

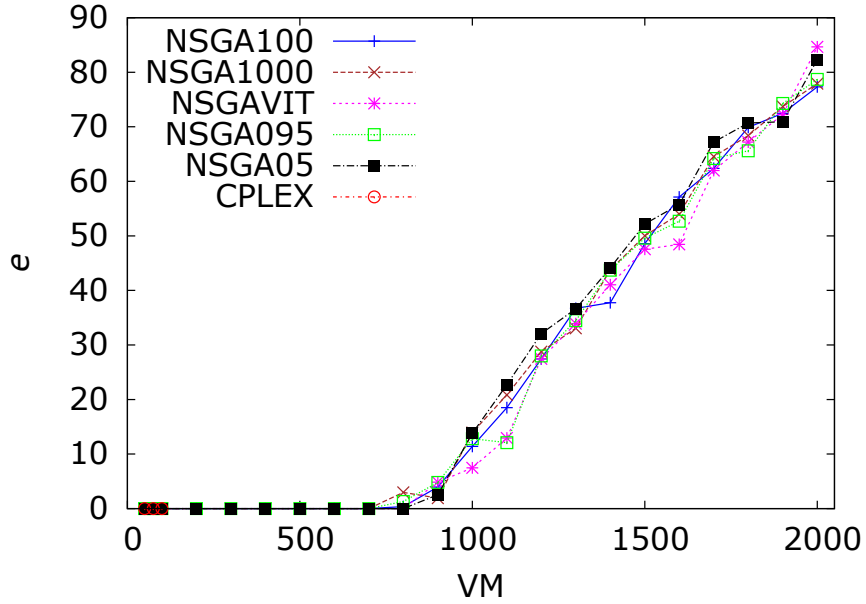

Figure 3.13: *Best solution for VMP1.*

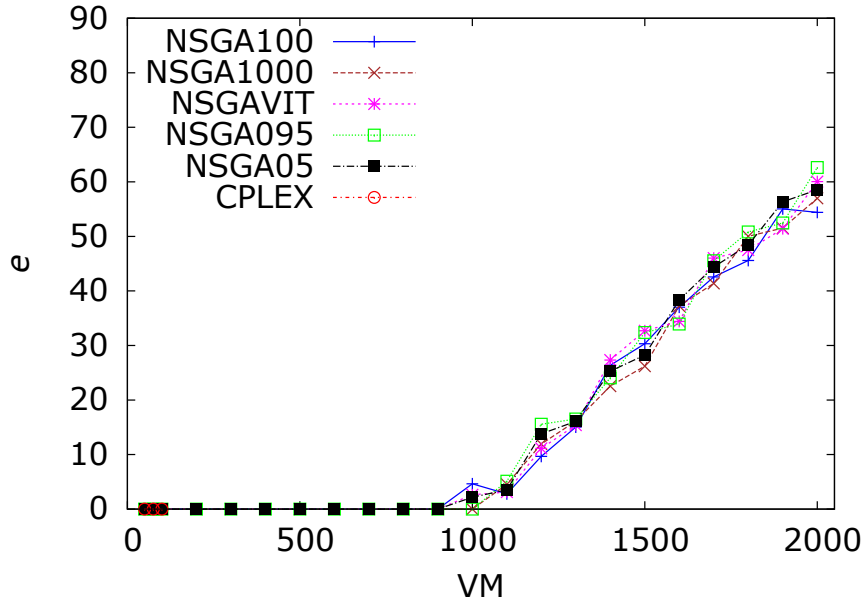

Figure 3.14: *Best solution for VMP2.*

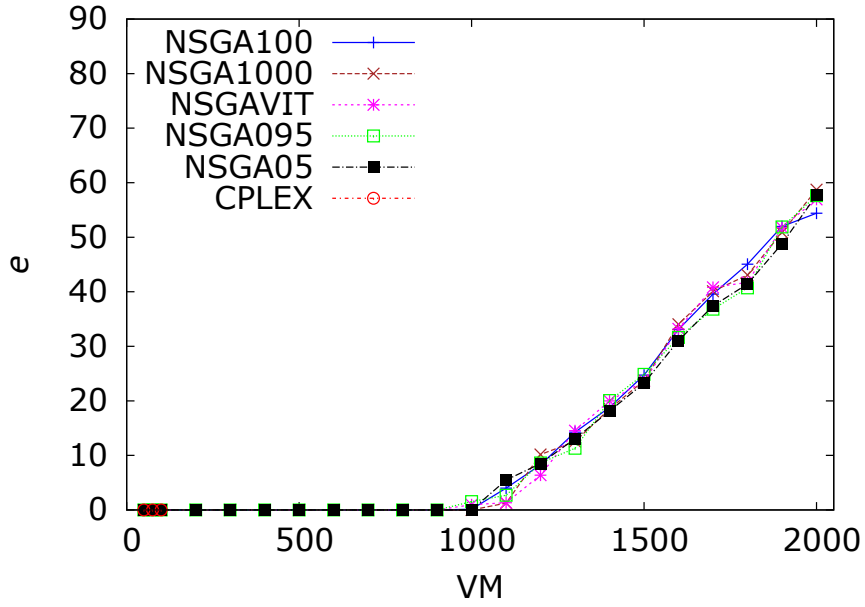

Figure 3.15: *Best solution for VMP3.*

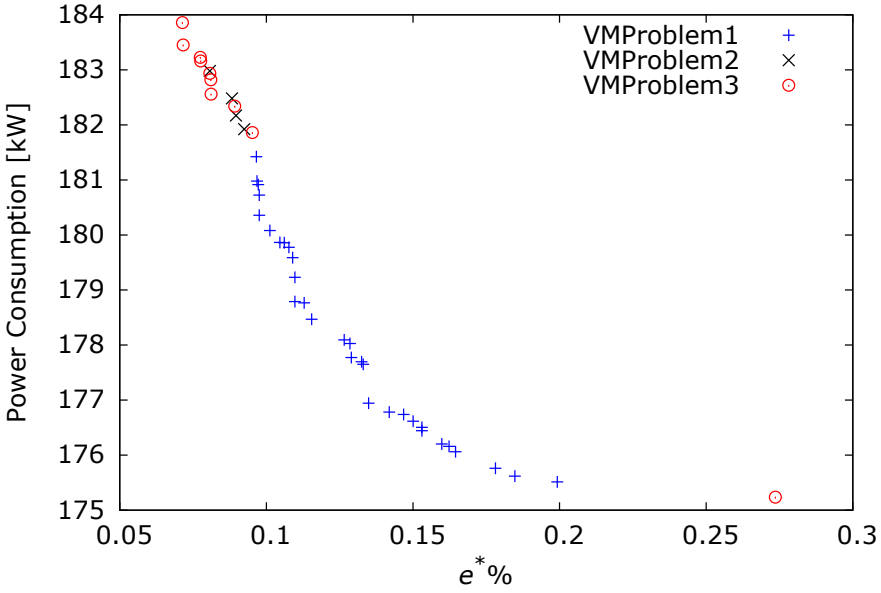

Figure 3.16: *Reference Pareto front.*

and VMP3 find a better minimum exploiting the set of correlated auxiliary objectives, reducing e from 90 (computed using VMP1) to 60 during the allocation of 2000 VMs. Of course, when we consider the set of auxiliary objective, we improve e trading off P as depicted in the RPF (Figure 3.16) which is computed out of the 40 independent runs for each VMProblem using only NSGA100. RPF it is mainly divided in three areas: the first area includes all the best solutions in terms of e that are computed by VMP3; the second one groupes all trade-off solutions between the two objectives retrieved by VMP1 and VMP2; and the third one collects all solutions optimizing P mainly computed by VMP1.

VMP1, VMP2 and VMP3 are able to explore the whole solution space but can narrow some parts of it. These settings could be used according to DC provider policy (i.e. if one of the two objectives has higher priority or when trade-off solutions are considered acceptable).

We use boxplots to evaluate the three quality metrics introduced above. Boxplots depict the main statistical information of a group of numerical data, and they indicate the maximum and the minimum values, and the quartiles. In the current representation, the lowest and the highest horizontal dashes represent the minimum and the maximum values respectively, the bottom and the top edges of the box display in order the first and the third quartiles, the thicker dash inside the box shows the second quartile, and the dots depictmembership functions the outlier elements.

We show the boxplots in Figures 3.19 to 3.21 for each indicator considering the allocation of 1000 VMs as a case study. Analysing separately all problem instances, in VMP1 all the heuristic performs similarly having the same confidence interval. In VMP2, NSGAVIT (that runs more iterations than the others) and NSGA100 reach better confidence intervals for epsilon. Further increasing the number of objectives, all settings experience some degradation of the epsilon value, but in this case NSGA100 and NSGAVIT perform better. The spread indicator is evaluated in Figure 3.21. VMP1 and VMP3 find solution on a narrowed part of the solution space while VMP2 search for solutions in a wider area. The trend of the hypervolume indicator is shown in Figure 3.20. NSGA1000 solutions have the lower values of hypervolume compared to the other approaches while NSGA05 reaches higher values for HV. NSGA05 having a lower crossover probability, recombines and replaces a less number of chromosomes during each iteration; for this reason, less fit solutions survive among the iterations and could be altered through the mutation to explore different parts of the solution space. Spread boxplots in Figure 3.21 show that increasing the number of auxiliary objectives the main objectives are less spread among each others: VMP2 and VMP3 weight more e rather than P reducing its value and finding solutions that better optimizes the first main objective.

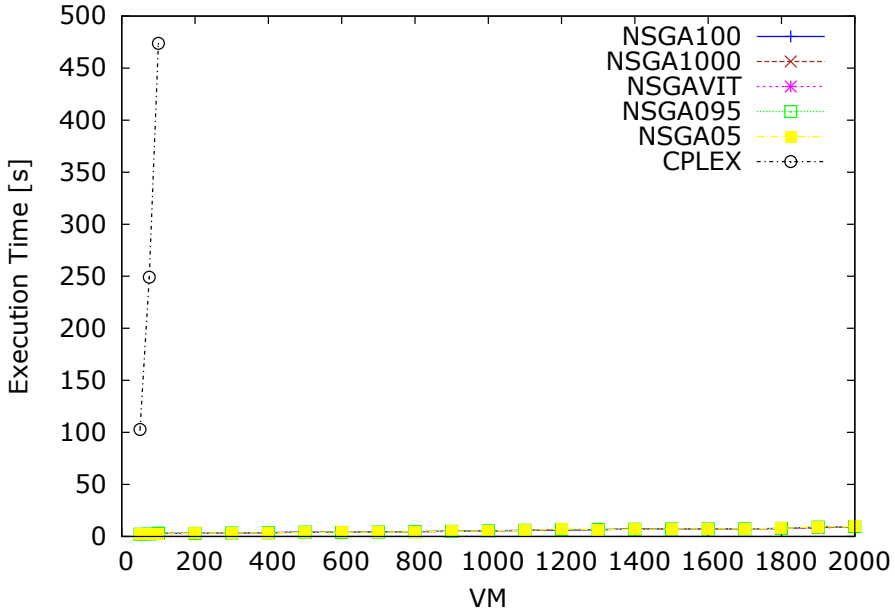

Figure 3.17: *Execution times.*

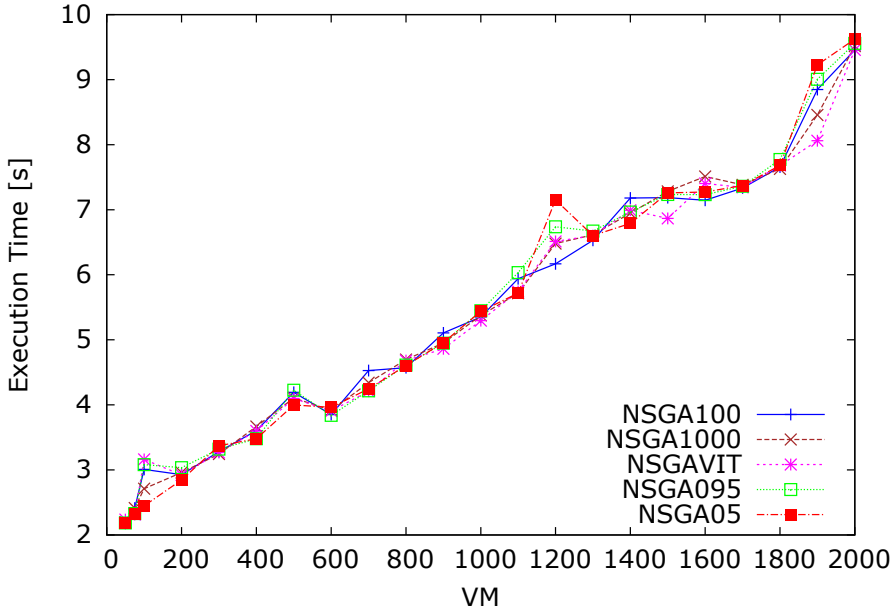

Figure 3.18: *MOGA execution times.*

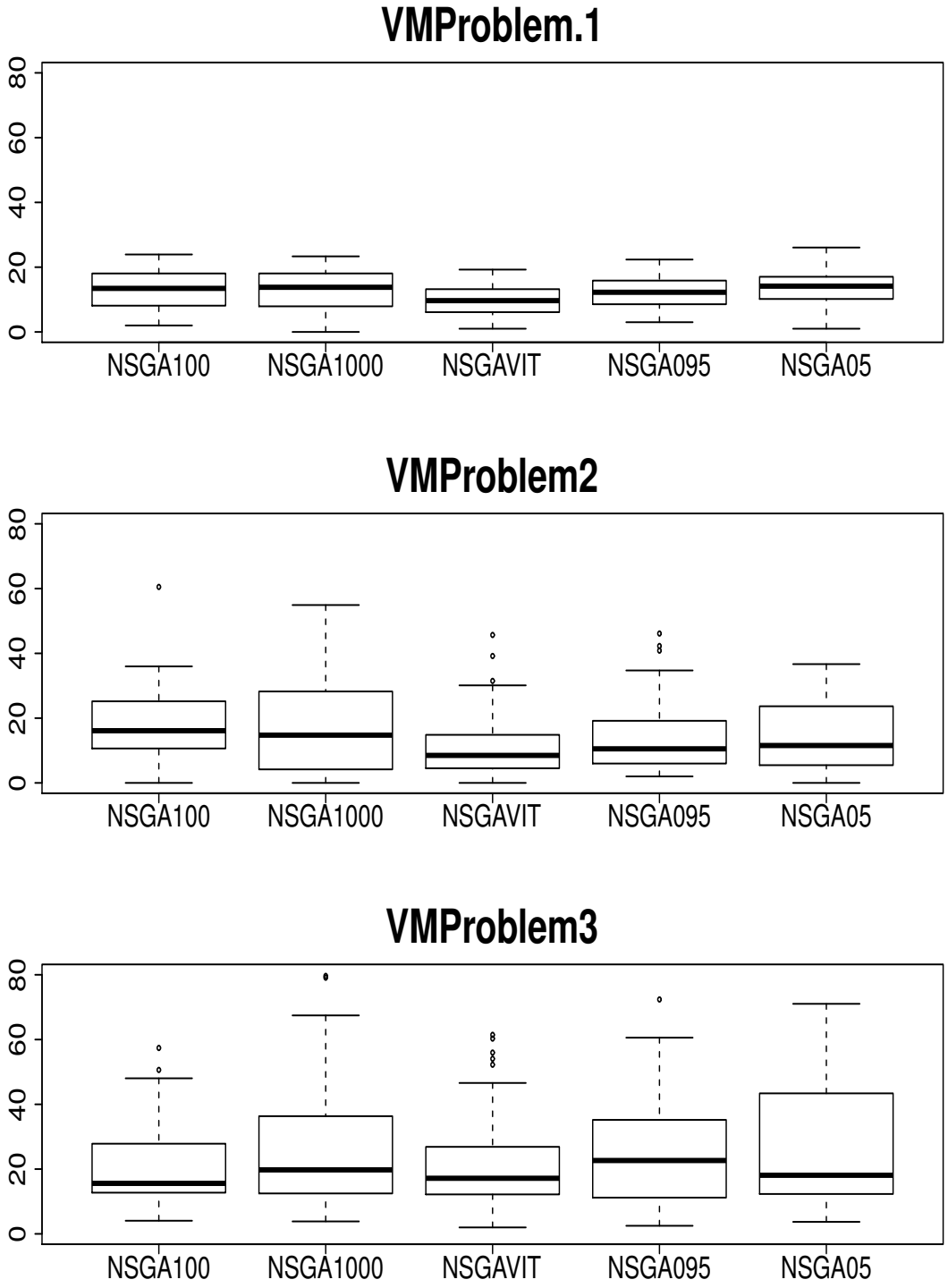

Figure 3.19: *Epsilon Boxplots.*

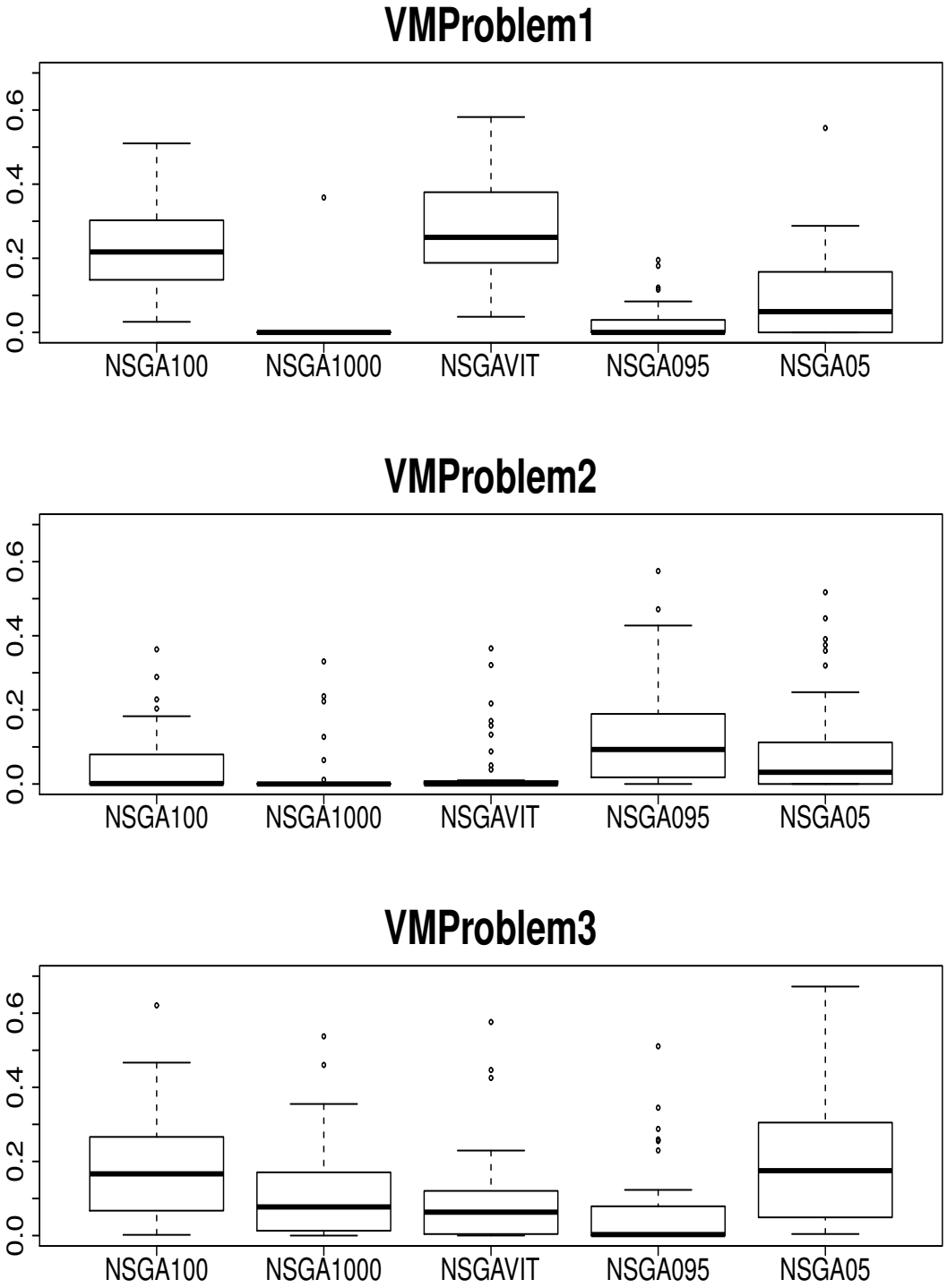

Figure 3.20: *Hypervolume Boxplots.*

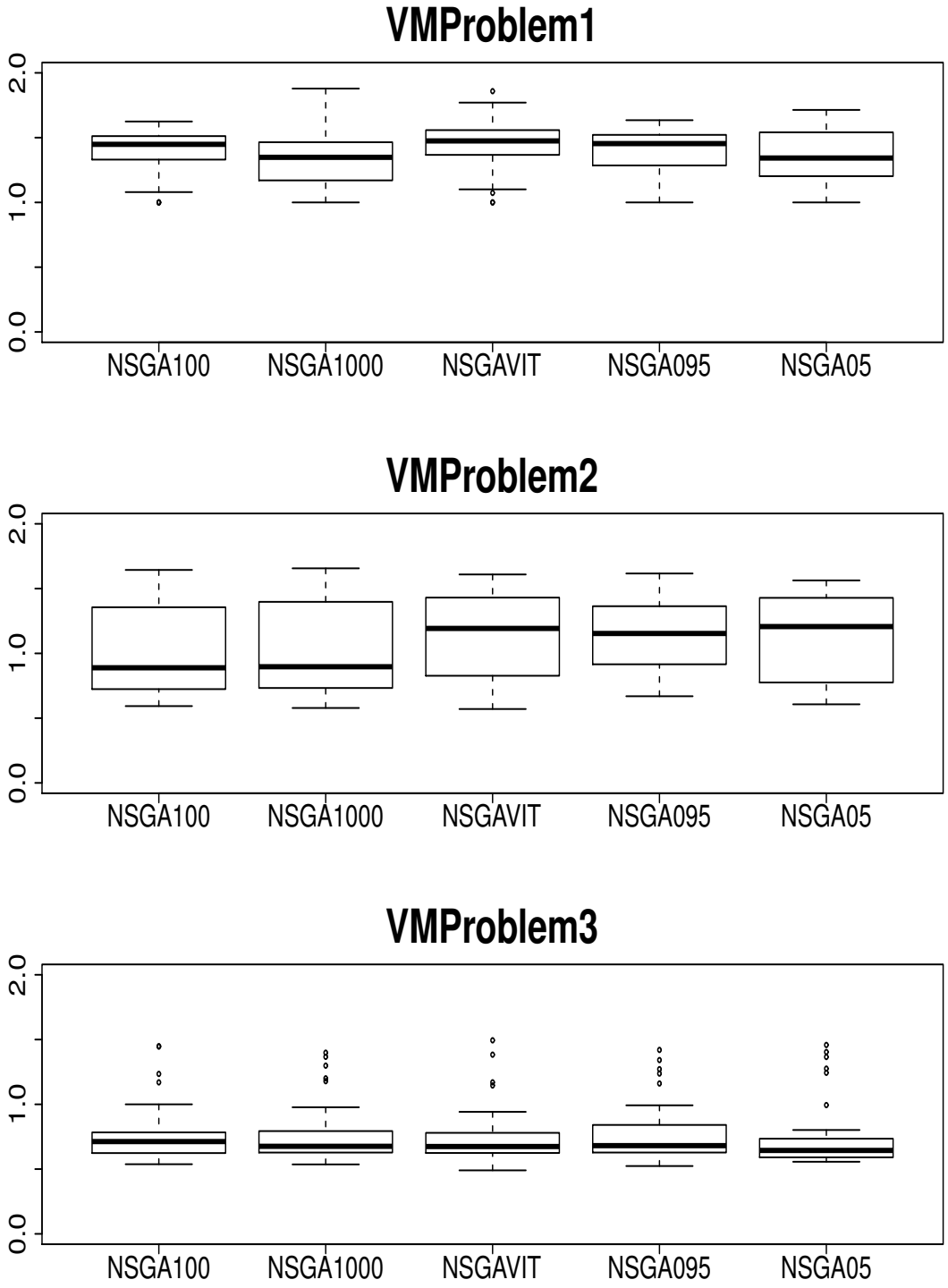

Figure 3.21: *Spread Boxplots.*

3.4 Dynamic VM Allocation

The last part of this chapter deals with dynamic VM allocation. In this section, we describe our IT Resource Allocator (ITRA) that is used to place dynamically VMs on servers and reserve network paths using a power-efficient allocation scheme.

3.4.1 Virtual Machine Allocation

The main goal of the ITRA is to accept as many VM requests as possible, reducing at the same time the network power consumption. Each VM request is characterized by four parameters representing the peak utilizations of CPU, RAM, disk and bandwidth. The server selection consists of the following steps:

1. Compute the candidate server list, i.e., the set of servers with enough IT resources to satisfy the request:
   (a) if the list is empty, the request is rejected;
   (b) otherwise, go to the next step.
2. Select the policy between:
   (a) *Multi Resource Best Fit* (BF) that strongly consolidates the system resource utilization choosing the server that has the least resources availability;
   (b) *Multi Resource Worst Fit* (WF) that selects the server having the highest resources availability, so as to balance the load among all the available servers.
3. Select the *best* server according to one of the possible strategies:
   (a) disjoint or joint Analytic ITRA (described in Section 3.4.1);
   (b) disjoint or joint Fuzzy ITRA (described in Section 3.4.1);
   (c) disjoint or joint Multi-Objective Dynamic Allocator (MODA) (described in Section 3.4.1).

The joint allocation strategies consider at the same time both computational and network requirements to perform the allocation of VMs. Instead, disjoint strategies split the allocation procedure in two different steps:

1. choose the server where to allocate the VM evaluating only the computational requirements rejecting the request if no server is available;
2. taking into account the bandwidth requirement, find the minimum-cost path connecting the chosen server to the gateway rejecting the request if no path is available.

ITRA associates the minimum-cost network path with each available server when a new request comes, and it discards servers that do not have enough resource nor at least an available path. The cost of the path is computed as the amount of power that will be consumed by the new network flow. More clearly, ITRA allocates the network path minimizing the increment of power consumption of the network devices. We provide a more detailed view about the path allocation procedure in Section 3.4.2.

For the sake of clarity, we summarize the symbols used in Table 3.20.

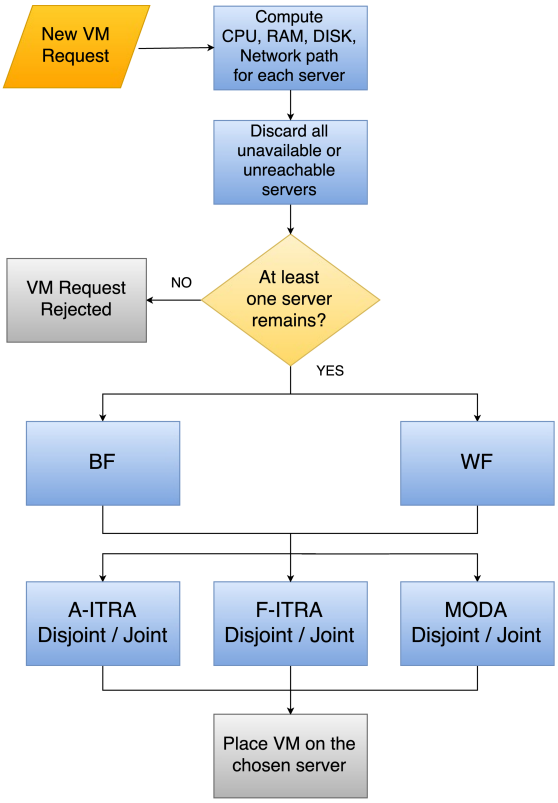

Figure 3.22: *IT Resource Allocator algorithm.*

Table 3.20: *Model Parameters.*

Parameter	Description
CPU_s	% of the s-th server free CPU after the placement of the VM.
RAM_s	% of the s-th server free space in RAM after the placement of the VM.
DISK_s	% of the s-th server free space in the storage after the placement of the VM.
PC_s	Cost value of the minimum-cost path from server s to the external gateway.
PCM	Sum of the costs of all the links in the network.
I_A^s	Availability index for the s-th server made by A-ITRA.
I_F^s	Availability index for the s-th server made by F-ITRA.
$f(\boldsymbol{x})$	Defuzzification method applied to the input vector $\boldsymbol{x}$.
N	Number of servers.

Analytic ITRA

Analytic ITRA (A-ITRA) computes for each candidate server the *A-ITRA Availability Index* (I_A) that takes into account the availability of IT resources. The joint version of the A-ITRA Availability Index is computed as follows:

$$I_A^s = \frac{1}{300}\left[\text{CPU}_s + \text{RAM}_s + \text{DISK}_s\right] + \alpha\frac{\text{PC}_s}{\text{PCM}}. \tag{3.18}$$

where

$$\alpha = \begin{cases} -1 & \text{if WF is adopted;} \\ 1 & \text{if BF is adopted.} \end{cases} \tag{3.19}$$

The first component of I_A represents the normalized availability of the system resources (by definition the three values range from 0 to 100) and the second one is the normalized path cost (i.e., $\frac{\text{PC}_s}{\text{PCM}}$). A-ITRA chooses the server minimizing I_A in case of BF, or the server maximizing I_A in case of WF. Note that different values of α are used since in both cases the path cost must be minimized.

When we adopt the disjoint A-ITRA, we calculate the Availability Index taking into account only the computational resources:

$$I_A^s = \frac{1}{300}\left[\text{CPU}_s + \text{RAM}_s + \text{DISK}_s\right]. \tag{3.20}$$

As for the joint version, disjoint A-ITRA first chooses the server minimizing or maximizing I_A^s when we respectively adopt BF or WF. Next, it computes the minimum-cost path towards the gateway, and associates it with the selected server.

Fuzzy IT Resource Allocator

Fuzzy Logic [35, 82] is a technique that deals with uncertain, imprecise, or qualitative information, as well as with precise information in systems which cannot be described by a formal and analytically tractable mathematical model. In Boolean logic, an element x can or cannot belong to a set $\mathcal{A}$ with a membership degree respectively equal to 1 or 0. Instead, in Fuzzy Logic, the membership degree of x to a fuzzy set $\mathcal{F}$ has a value in a continuous interval between 0 and 1. Fuzzy set theory allows an element to have partial membership degree in one or more fuzzy sets. This membership degree is obtained through membership functions that map elements into the interval [0, 1].

Fuzzy Logic can be used for the design of control systems, called Fuzzy Logic Controller (FLC). The core of the FLC [50, 51] is the inference engine, whose role is to apply the inference rules (IF-THEN rules) contained in the rule base. IF-THEN rules are made by premises and conclusions, and embody the system control strategies. Since fuzzy rules use fuzzy sets and their associated membership functions to describe system variables, two operations are necessary for translations between conventional and fuzzy values: fuzzification and defuzzification. The former maps input values into one or more fuzzy sets, the latter produces a single conventional value that best represents the inferred fuzzy values.

More specifically, our FLC includes the following modules (see [10] for more details):

- *Inputs fuzzification*: we defined four input fuzzy sets ("Not available", "Small", "Medium", and "Large") and their membership functions. Figure 3.23 shows an example: while RAM_s and DISK_s are fuzzified in a single fuzzy set (i.e., "Medium" for RAM and "Large" for DISK), CPU belongs to two different sets with a certain probability (0.7 in "Small" and 0.3 in "Medium");
- *Inference engine*: we considered two different inference processes:
  1. Mandami (Fuzzy ITRA (F-ITRA) M), that uses the min operator;
  2. Sugeno [70] (F-ITRA S), that applies both linear or constant membership functions;

  and six output sets, "Empty", "Almost empty", "Medium", "Almost full", "Full" and "Not available", whose corresponding membership functions are depicted in Figure 3.24;
- *Defuzzification method*, denoted with f, that computes the center of gravity as defuzzification algorithm for the aggregated fuzzy subset.

Similarly to A-ITRA, we use the FLC to compute the *F-ITRA Availability Index (I_F)* for each server. We define the joint and disjoint F-ITRA Availability Index in (3.21) and (3.22) respectively assuming as values for α the ones reported in (3.19). Analogously to A-ITRA, both disjoint and joint F-ITRA choose the server maximizing or minimizing I_F^s each in order when WF or BF is adopted. Moreover, disjoint F-ITRA computes the minimum-cost path only when the server is selected.

$$I_F^s = f(\text{CPU}_s, \text{RAM}_s, \text{DISK}_s) + \alpha \frac{\text{PC}_s}{\text{PCM}}. \tag{3.21}$$

$$I_F^s = f(\text{CPU}_s, \text{RAM}_s, \text{DISK}_s). \tag{3.22}$$

Multi-Objective Dynamic Allocator (MODA)

MODA allocates VMs using a technique based on the multi-objective optimization: available resources are the objectives that should be optimized all together. Solutions are not comparable among each others in a multi-objective computation, so we use the concept of Pareto front [65] to deal with the incommensurability of vector solutions.

If we consider two solutions s and t for a minimization problem, and we associate a set of m objectives denoted with f_1, f_2, ... , f_m, we say that s dominates t if an integer value d exists such that:

$$\begin{cases} f_d(s) < f_d(t) & 1 \leq d \leq m; \\ f_l(s) \leq f_l(t) & 1 \leq l \leq m \wedge l \neq d. \end{cases} \tag{3.23}$$

Solutions s and t are *non-dominated* if both s does not dominate t and viceversa, and the set including all the non-dominated solutions is called *Pareto front*. Within a Pareto front, the solution that optimizes all the objectives at the same time is called *ideal vector* [25].

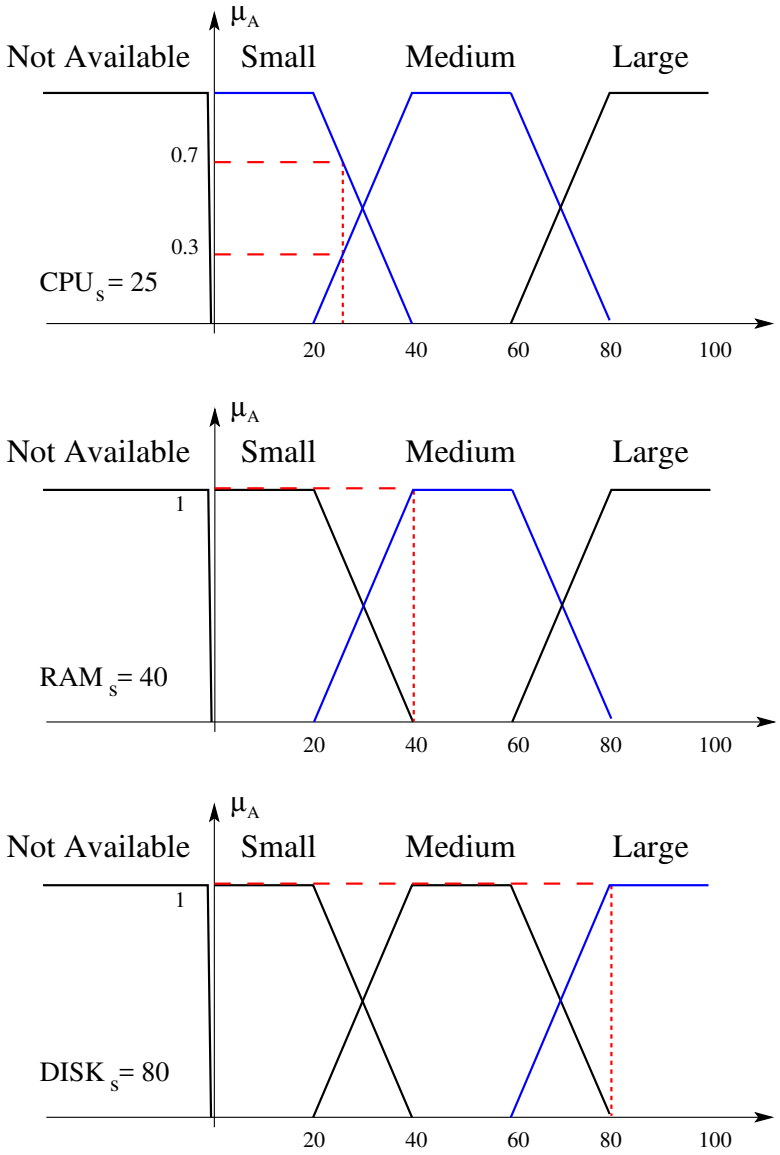

Figure 3.23: *Inputs membership functions.*

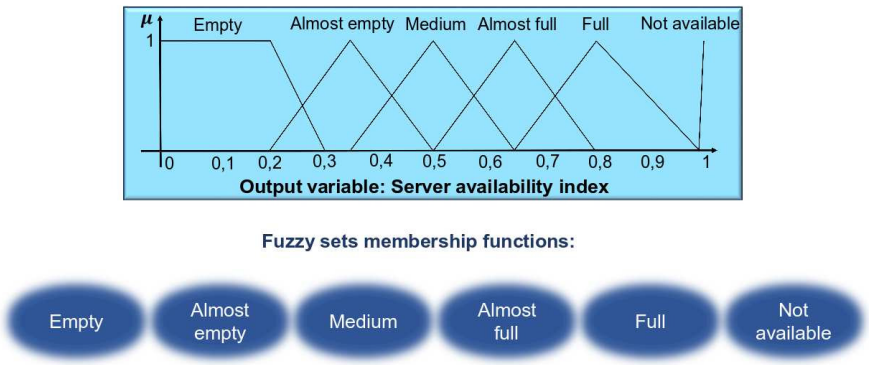

Figure 3.24: *Inference process outputs fuzzy sets and their membership functions.*

MODA computes in multiple steps the allocation procedure; when a new VM should be allocated, MODA creates a list of servers able to fit the request. The difference between joint and disjoint MODA concerns only the number of objectives that should be optimized:

- disjoint MODA considers only CPU_s, RAM_s and DISK_s, and it computes the minimum-cost path after the phase of server selection;
- joint MODA considers CPU_s, RAM_s, DISK_s and PC_s, and it associates the minimum-cost path with each server during the server selection phase.

Then, MODA computes the Pareto front for the current allocation, and finally it chooses one of the possible allocations according to the applied policy (i.e., BF or WF) and one of the two possible strategies:

1. Choosing randomly one solution within the Pareto front (MODA-R);
2. Normalizing all the objectives and taking the solution that has the minimum distance from the ideal vector (MODA-D).

The joint MODA solves the minimization problem described in formula (3.24) when it adopts the BF policy:

$$\begin{array}{lll} \text{minimize} & \min\limits_{1\leq s\leq N};\{\text{CPU}_s,\ \text{RAM}_s,\ \text{DISK}_s,\ \text{PC}_s\}\ ; & \\ \text{subject to:} & 0 \leq \text{CPU}_s \leq 100 & 1 \leq s \leq N; \\ & 0 \leq \text{RAM}_s \leq 100 & 1 \leq s \leq N; \\ & 0 \leq \text{Disk}_s \leq 100 & 1 \leq s \leq N; \\ & 0 \leq \text{PC}_s \leq \text{PCM} & 1 \leq s \leq N. \end{array} \tag{3.24}$$

The allocator maximizes instead the computing resources in case we set the WF policy, but it still minimizes the path cost:

$$\begin{array}{lll} \text{minimize} & \min\limits_{1\leq s\leq N};\{\text{-CPU}_s,\ \text{-RAM}_s,\ \text{-DISK}_s,\ \text{PC}_s\}\ ; & \\ \text{subject to:} & 0 \leq \text{CPU}_s \leq 100 & 1 \leq s \leq N; \\ & 0 \leq \text{RAM}_s \leq 100 & 1 \leq s \leq N; \\ & 0 \leq \text{Disk}_s \leq 100 & 1 \leq s \leq N; \\ & 0 \leq \text{PC}_s \leq \text{PCM} & 1 \leq s \leq N; \end{array} \tag{3.25}$$

As described above, disjoint MODA solves the same optimization problems without considering PC and associating the minimum-cost path only with the selected server.

3.4.2 Power Consumption and Network Path Computation

We adopt the power consumption model for switches presented in [48], whose parameters are summarized in Table 3.21.

Table 3.21: *Notation of the parameters related to power consumption.*

Parameter	Description
P_{switch}	the average power consumption of a single switch.
$P_{chassis}$	the constant power consumed by chassis.
n	the number of line cards in the switch.
$P_{linecard}$	the fixed amount of power consumed by line cards.
$P_{load}(i,r)$	the power consumed by the i-th port while transmitting at bit-rate r.
$P_{max}(i)$	$P_{load}(i,r)$ while transmitting at the maximum bit-rate.
P_{fix}	$P_{chassis} + n * P_{linecard}$.
$B(i)$	the bandwidth transmitted by the i-th line card.
$B_{max}(i)$	the maximum capacity of the i-th line card.
R	the bandwidth request of a new (i.e., not allocated) VM.
w	the link weight.

The power model for switches is:

$$P_{switch} = P_{chassis} + n * P_{linecard} + \sum_{i=1}^{n} P_{load}(i,r). \tag{3.26}$$

We express an equivalent (and simplified) formulation of (3.26) by grouping all the fixed terms and taking into account that the load-dependent power component. The power consumption is linearly proportional to the transmission rate, as the authors showed in [67], and we assumed that the line cards of a link are active even when no packet is transmitted:

$$P_{switch} = P_{fix} + \sum_{i=1}^{n} \frac{B(i)}{B_{max}(i)} * P_{max}(i). \tag{3.27}$$

We might estimate the utilization values retrieving periodically the number of bytes transmitted by each link from the SDN controller; then, we could be able to compute the

average bandwidth during the considered period [11]. After pruning the links without enough available bandwidth, i.e.,

$$B(i) + R \leq B_{max}(i); \tag{3.28}$$

we used the Dijkstra's algorithm for minimizing power consumption; for each line card, we set as weight the increment of the power cost due to the transmission of the new flow.

$$w_i = \frac{R}{B_{max}(i)} * P_{max}(i). \tag{3.29}$$

In this analysis we considered only the power consumption of network devices and we neglected the server consumption because we supposed all the servers to be equal, and we applied them the same consumption profile.

3.4.3 Experiments and Results

We describe in this section two sets of experiments. In Section 3.4.3, we focus on power consumption and path allocation algorithm comparing our power-aware path allocation with the classical equal cost routing. Then, we evaluate the power consumption of the networking devices. In the second experiment, we analyze the performance of our allocators in terms of accepted VMs; we describe this experiment in Section 3.4.3. At the end, we show the difference between the joint and disjoint A-ITRAs.

In our experiments we use as baseline the simplest possible allocation strategy named *First Fit* (FF) that allocates VMs one by one on the first available server, and we execute 30 independent runs for each experiment.

Simulation scenario

The simulation scenario consists of 16 servers interconnected through a three-tier fat tree network topology [13], which is one of the most widely adopted topologies within DCs. Three-tier fat tree is structured in three different layers: *access*, *aggregation*, and *core*. Access switch provides connection to servers, and it is connected to a pair of aggregation switches. Access switches provide connection to servers and are connected to a pair of aggregation switches. Servers and access switches connected to the same couple of aggregation switches belong to the same *pod*. Core switches guarantee connectivity between all the pods and the external gateway. All the servers had the same hardware and they were empty at the beginning.

In our experiments, we supposed that VMs remained active until the end of the simulation. As already mentioned, VMs are characterized by CPU, RAM, disk and bandwidth. The values of such parameters are generated as described in Table 3.22, while the parameters related to power consumption and network equipment are reported in Table 3.23.

Power-aware network path allocation

In this experiment, we evaluate the power efficiency of our path allocation strategy. The only differences with respect to the parameters reported in Table 3.23 are the two power consumption profiles for aggregation and core switches. Within the same pod, the two aggregation switches have different power profiles and both of them are connected

Table 3.22: *VM request parameters.*

Parameter	Value
CPU percentage RAM percentage Disk percentage	Uniformly distributed in $[10; 30]$
Bandwidth	Uniformly distributed in $[100; 300]$Mbps

Table 3.23: *Power consumption and network parameters.*

Parameter	Value
Access switch P_{fix}	160W
Access switch P_{max}	200W
Aggregation switch P_{fix} Core switch P_{fix}	2000W
Aggregation switch P_{max} Core switch P_{max}	2500W
Server-Access link bw	1Gbps
Access-Aggregation link bw Aggregation-Core link bw	10Gbps

to the same number of less consuming and more consuming core switches. The two profiles for aggregation and core switches are shown in Table 3.24. We generate 60 requests with the parameters described in Table 3.30, and we allocate them using the FF policy. More specifically, we adopted FF since we were interested only in evaluating the behavior of the network path allocator regardless the server placement strategy.

We use violin plots [43] to show the main statistical information for our allocators. Violin plots represent the first and the third quartiles as the extreme points of the bold line and the median using a white dot, and they trace the probability density function symmetrically at both sides of the plot. We compare our power-aware network allocation strategy (PA-Dijkstra) with the classic power-unaware equal cost path (PU-Dijkstra) in Figure 3.25. The PA-Dijkstra saves on average around 1 kW with respect to the PU-Dijkstra. The amount of power saved corresponds to the 3% of the total power consumption due to all the switches.

Performance evaluation of the allocation strategies

We carried out two different experiments evaluating:

- the "steady state" of the system after 80 VMs, when on average all the allocators saturate the available resources, and new VM requests may not be allocated;
- the "transient state" of the system after 20, 40, and 60 VM requests.

Table 3.24: *Aggregation and core switch power profiles.*

Profile	P_{fix} [W]	P_{max} [W]
First profile	2000	2500
Second profile	2500	5000

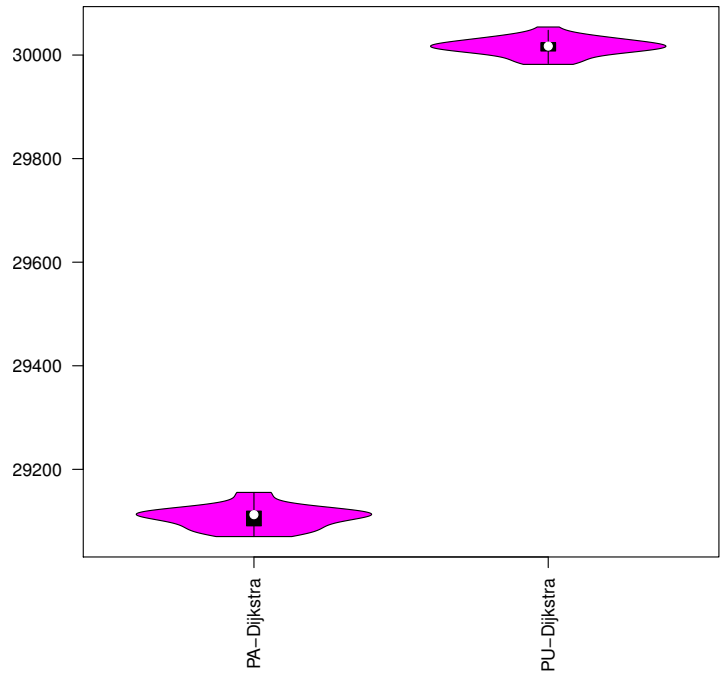

Figure 3.25: *Power consumption after 60VM requests.*

In both experiments we compared our joint with disjoint allocators evaluating them in terms of accepted/rejected requests. We used Matlab Simulink to perform our simulations, doing 30 independent runs for each experiment.

As described above, we generated 80 VM requests in the first experiment. Our purpose is to evaluate the steady state of the system reducing the availability of many resources as possible. In Figures 3.26 and 3.27 we show the results respectively for joint and disjoint allocators using as metric the number of rejected requests and the FF policy as comparison term. Furthermore, we show in Table 3.25 all the numerical values related to each policy and allocation strategy. Joint allocators reach on average best results compared to the disjoint approaches, and they also guarantee a lower standard deviation. We underlined the two best possible mean and standard variation values among all possible combinations. All joint allocators are able to satisfy on average 66 requests regardless the adopted policy or strategy. On the contrary, disjoint allocators decrease their effectiveness when BF policy is adopted because the consolidation procedure quickly saturates the bandwidth availability producing congestion at the higher layers of the topology. Moreover, disjoint BF allocators have a very high standard deviation because sometimes the network path computation (performed only when the server is already chosen) may fail, and the request may be blocked even if there is enough available computational resources. Lastly, all the BF allocators have a higher standard variation compared to the corresponding WF ones, so WF allocators are usually more stable. In the second experiment, we focused on the transient phase inspecting the evolution of the system after steps of 20 allocation requests. We show the number of rejection after 20 and 40 VM requests respectively in Figures 3.28 and 3.29 for the disjoint allocators. It is worth noting that joint allocators do not reject any VM until the 40th request. After the 60th request, BF joint allocators are still able to allo-

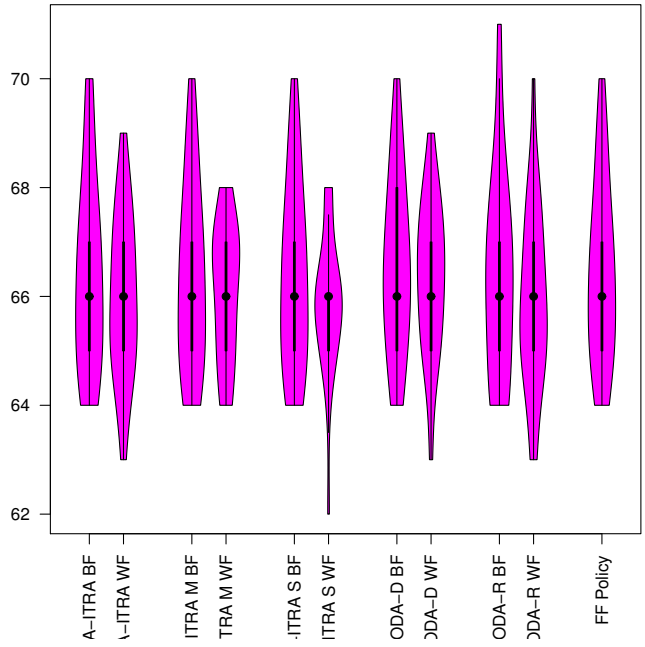

Figure 3.26: *VMs allocated by joint allocators.*

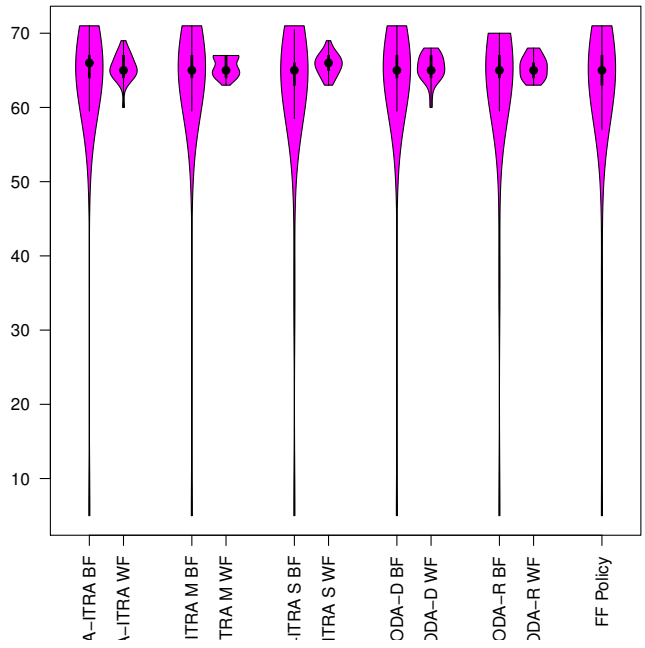

Figure 3.27: *VMs allocated by disjoint allocators.*

Table 3.25: *VMs allocated by disjoint allocators after 80 requests.*

	Joint		Disjoint	
	Mean	Std Dev	Mean	Std Dev
A-ITRA BF	66.38	1.76	63.10	11.56
A-ITRA WF	66.00	1.46	65.38	1.78
F-ITRA M BF	66.34	1.70	63.03	11.61
F-ITRA M WF	66.07	1.25	65.48	1.30
F-ITRA S BF	66.34	1.70	61.66	12.58
F-ITRA S WF	65.90	1.26	65.76	1.48
MODA-D BF	66.55	1.62	63.10	11.56
MODA-D WF	66.34	1.37	65.38	1.82
MODA-R BF	66.34	1.86	62.28	12.46
MODA-R WF	65.93	1.60	65.38	1.52
FF	66.41	1.64	61.72	12.70

cate all the requests unlike disjoint and WF joint allocators. We show the violin plots in Figures 3.30 and 3.31 and the numerical value in Table 3.26.

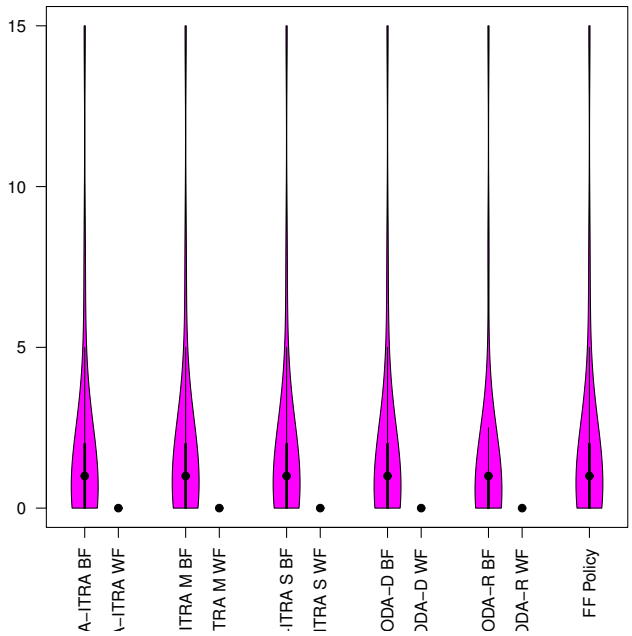

Figure 3.28: *VMs rejected by the network after the 20th request.*

Summarizing all the results, joint allocators perform always equally to or better than the corresponding disjoint approach. In the low load phase (i.e., when the DC is not overloaded), BF joint allocators satisfy all the requests, while WF ones perform better during the high load phase. For what concerns only disjoint allocators, WF policies reach results similar to the joint ones, so we suggest to adopt WF policy in this case choosing the allocation strategy according to the parameter that preferably should be optimized (i.e., mean or standard deviation of the allocated requests). In our scenario,

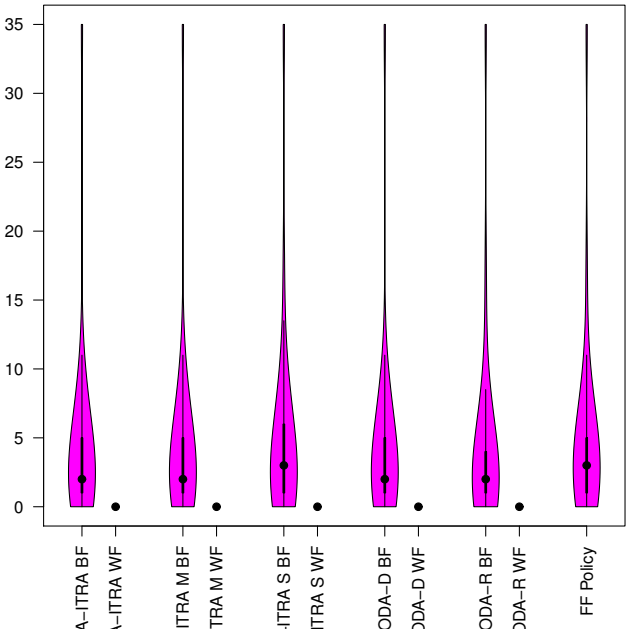

Figure 3.29: *VMs rejected by the network after the 40th request.*

all the joint allocation strategies using the same policy (i.e., BF or WF) obtained similar results with respect to the mean and the standard deviation. The A-ITRA approach is the least computational expensive, while the others increase their complexity obtaining little improvements. However, all the joint allocation strategies are valid alternatives to be adopted in a real system. For what concerns the two policies, BF groups all the VMs in the least number of servers, so it may reduce the number of active computing devices; on the other hand, BF quickly saturates the access network links. The WF policy uses all the servers in the transient phase since it is a load balancing policy and it less congests the network. We suggest to adopt BF policy with a high oversubscription rate between the network layers or when servers could be put in sleep mode for power saving reason. On the contrary, we strongly suggest to use the WF policy if the DC provider prefers to leave all computing resources active and running, or the oversubscription rate is low.

3.4.4 Enhanced Worst Fit

As we discussed above, allocators based on WF allocate on average more VMs with respect to the ones implementing the BF policy. As side effect, worst fit fragments the available resources that can not be used to allocate other VMs. In order to mitigate the resource fragmentation, we present a new policy that combines both the behaviours of BF and WF.

The main difference with respect to the classic approaches, is that E-WF allocates VMs taking into account the history of the previous requests. We point out the E-WF code implementation in Algorithm 1. At the very beginning, E-WF works exactly as

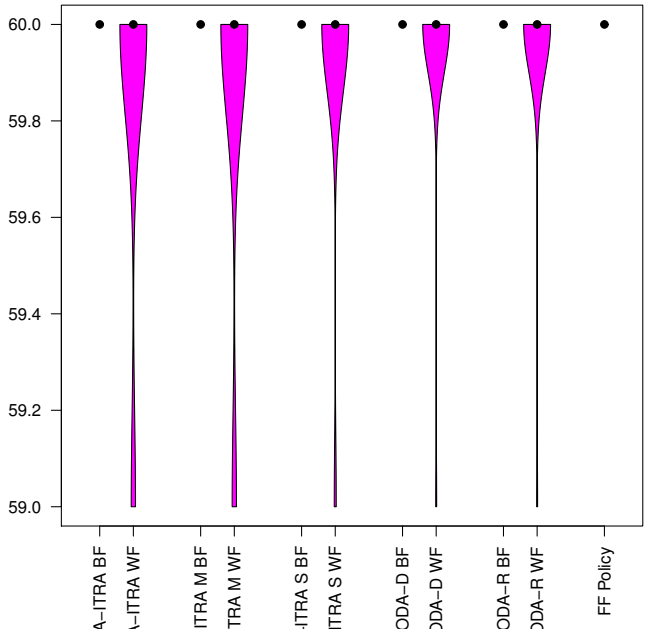

Figure 3.30: *Snapshot of the joint allocators after 60 requests.*

WF, but it memorizes the history of all the previous requests. In the database of the previous requests, each resource is stored separately. More clearly, E-WF separates the requirements of each VM request in four lists, namely CPU, RAM, disk and bandwidth as shown in Algorithm 1. Also in this case, we may include more or different resources without loss of generality. When the E-WF allocator receives a number of VM requests greater than a certain threshold, its behaviour changes and each new VM allocation is evaluated according to the procedure that is described in Algorithm 1. We set the threshold equal to the number of servers, so the allocator places exactly one VM per server before changing its behaviour if no requests are rejected. E-WF computes and associates with each server a third objective for each new request. We call this new objective *penalty*. E-WF uses the penalty value to forecast (and avoid) the future resource fragmentation.

We assign a penalty for each resource that is less available than the corresponding first quartile in the history of all the previous requests, and the amount of the penalty is equal to the difference between the computed quartile and the available resource. We use the first quartiles as thresholds: if the availability of some resources is less than the quartile, there is a low probability that new requests may saturate the resource avoiding fragmentation and we prefer other servers having a lower penalty. Considering the symbols denoted in Table 3.27, we compute the penalty value for the i-th resource of server s as (3.30) and code the penalty computation in Algorithm 2.

$$p_s^{(i)} = \begin{cases} q_s^{(i)} - r_s^{(i)} & \text{if } q_s^{(i)} > r_s^{(i)}; \\ 0 & \text{otherwise.} \end{cases} \tag{3.30}$$

Algorithm 1 Multi-objective Enhanced Worst Fit Allocator.

Require: VM allocation request req, Server list
Ensure: VM allocation server and network $path$, or rejection
1: $pf \leftarrow$ Pareto front of the current allocation
2: $g \leftarrow$ gateway vertex
3: $q \leftarrow$ vector of quartiles of each resource
4: **for each** server s **do**
5: **if** all the resources are enough to satisfy the request and at least one path has enough bandwidth to connect s to g **then**
6: $rfit \leftarrow ComputeResourceFitting(req, s)$
7: $penalty \leftarrow ComputePenalty(req, s, q)$
8: $< path, consumption > \leftarrow ModifiedDijkstra(s, g)$
9: associate $path$ with s
10: append $< penalty, rfit, consumption >$ to pf
11: **else**
12: try another server
13: **end if**
14: **end for**
15: $reqnum \leftarrow reqnum + 1$
16: **if** pf contains at least one feasible solution **then**
17: eliminate all non-dominated solutions
18: choose the solution which minimizes the objectives according to their priority.
19: choose the best server and allocate the VM
20: **else**
21: reject the request
22: **end if**
23: add the request to the history
24: **for** $i := 1$ to res **do**
25: update the vector q
26: **end for**

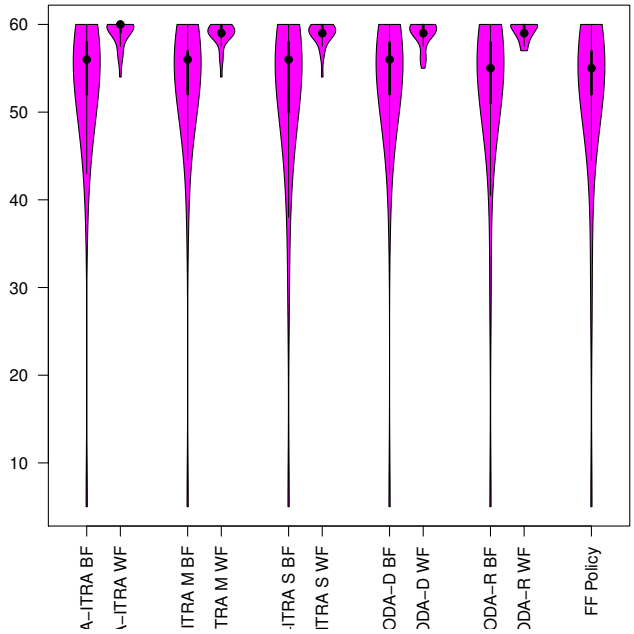

Figure 3.31: *Snapshot of the disjoint allocators after 60 requests.*

The total penalty for a specific allocation on server s is defined as:

$$p_s = \sum_i p_s^{(i)}. \tag{3.31}$$

The penalty value is one of the three objectives of our optimization problem that should be minimized. Since we added another objective, we define the new priority values associated with each objective in Table 3.28. We remark that higher priority values correspond to lower objective priorities and vice versa. When we adopt E-WF, we aim to minimize the penalty value rather, and we consider the other two objective when more solutions have the same penalty value using the same mechanism implemented for BF and WF.

Algorithm 2 ComputePenalty.

Input: request req, server s, quartile vector q.
Output: penalty value

```
1: reqnum ← the number of received allocation requests
2: penalty ← 0
3: if reqnum ≤ threshold then
4:     for i := 1 to res do
5:         if req[i] < q[i] then
6:             penalty ← penalty + (q[i] − req[i])
7:         end if
8:     end for
9: end ifreturn penalty
```

Table 3.26: *VMs allocated by disjoint allocators after 60 requests.*

	Joint		Disjoint	
	Mean	Std Dev	Mean	Std Dev
A-ITRA BF	60	0	52.62	10.38
A-ITRA WF	59.86	0.35	58.93	1.56
F-ITRA M BF	60	0	52.66	10.34
F-ITRA M WF	59.86	0.35	58.79	1.42
F-ITRA S BF	60	0	51.45	11.28
F-ITRA S WF	59.93	0.26	58.83	1.39
MODA-D BF	60	0	52.69	10.34
MODA-D WF	59.97	0.19	58.83	1.47
MODA-R BF	60	0	52.10	11.15
MODA-R WF	59.97	0.19	59.14	1.03
FF	60	0	51.62	11.23

Table 3.27: *E-WF Symbols.*

Symbol	Description
s	the current server
res	total number of resources
$i \in \{1, \ldots, res\}$	the resource index
$p_s^{(i)}$	the penalty value of the i-th resource of server s
p_s	the total penalty value associated with server s
$r_s^{(i)}$	the i-th resource availability index after the allocation of the current request on server s
$q_s^{(i)}$	the first quartile computed from the i-th resource requests distribution

Simulation Scenario

DC topology Our simulation scenario consists of two DCs organized according to the *three-tier fat tree* and the *spine-and-leaf* topologies that we describe in this section. The three-tier fat tree is one of the most widely adopted topology [65]. This topology is structured in three different layers: access, aggregation, and core.

Access layer provides the connectivity of servers, that are grouped into racks and connected to a single *Top of the rack* (ToR) switch.

Each ToR switch is linked to a pair of more powerful Aggregation switches that provide redundant connectivity to ToR switch. A set of ToR switches connected to the same pair of aggregation switches (including those aggregation switches) is named *pod*. Core switches at a higher layer guarantee connection among different pods and towards the external gateway.

The spine-and-leaf [1] is also a tree topology, but it is composed only of two layers: *leaf* and *spine*. The leaf layer consists of access switches that connect to devices such

Table 3.28: *Objective Priorities for EWF.*

Objective	Priority
penalty	1
resource fitting	2
path consumption	3

as servers, while the spine layer is the backbone of the network and is responsible for interconnecting all leaf switches. Every leaf switch connects to every spine switch in the fabric. The path is randomly chosen so that the traffic load is evenly distributed among the top tier switches. If one of the top tier switches were to fail, it would only slightly degrade performance throughout the DC.

Evaluation metrics In our experiments, we evaluate the effectiveness of our allocation strategies in terms of VM requests allocated/rejected, and power consumed. We use two metrics to evaluate the performance of our allocators: *number of VM request rejected* that we aim to minimize, and *relative ranking* (or shortly *ranking*) that is used to establish which strategy allocates more VMs with respect to the others. We denote as *best strategy* the strategy allocating more VMs, and as *worst strategy* the one allocating less VMs during a single run. The ranking procedure sorts all the strategies according to the number of rejected VM requests, and assigns a ranking value equal to:

$$r_i = 4 - \#bs_i; \tag{3.32}$$

where r_i is the ranking value of the i-th strategy, and $\#bs_i$ is the number of strategies that allocate more VMs than the i-th one. More clearly, the ranking procedure assigns a ranking value of four to the best strategy down to one to the worst strategy except when two or more strategies tie. We provide an example of ranking values for tying strategies in Table 3.29.

Our second aim is to reduce the overall power consumption of all the network elements, namely server network interface cards (NICs) and switches. For the evaluaton of the power consumption, we generate a lower number of requests such that all the allocation strategies do not reject any VM requests in order to avoid unfair comparisons since the allocation of a different number of VMs alters the amount of power consumed. Then, we compare our PA network allocation strategy with a classic shortest path first (SPF) routing algorithm evaluating the amount of power consumed by both approaches.

VM generation and runs We randomly generate different types of VM requests characterized by one of the three profiles that we detail in Table 3.22:

- *balanced*;
- *cpu-intensive*;
- *bandwidth-intensive*.

At the beginning, the VM generator produces a balanced number of VM requests for each profile, and it shuffle all the requests in a random order. All the allocation strategies receive the same VM requests at the same order in a single run. We execute 30

Table 3.29: *Example of ranking assigment.*

Strategy	VM Rejected	Ranking value
E-WF	1	4
WF	1	4
BF	5	2
FF	10	1

independent repetitions of the same experiments (i.e., we generate 30 different sets of VM requests for each experiment), and we use violin plots [43] to show the most important statistical information of our results.

Table 3.30: *VM request parameters.*

Profile	Parameter	Value
Balanced	CPU percentage RAM percentage Disk percentage	Uniformly distributed in $[5; 15]$
	Bandwidth	Uniformly distributed in $[5; 10]$Mbps
CPU intensive	CPU percentage	Uniformly distributed in $[15; 20]$
	RAM percentage Disk percentage	Uniformly distributed in $[5; 15]$
	Bandwidth	Uniformly distributed in $[5; 10]$Mbps
Bandwidth intensive	CPU percentage RAM percentage Disk percentage	Uniformly distributed in $[5; 15]$
	Bandwidth	Uniformly distributed in $[20; 30]$Mbps

Experiments and Results

Performance evaluation Our simulator generates 100 VM requests, namely 33 balanced, 33 cpu-intensive and 34 bandwidth-intesive requests, that are allocated on two different DCs. We first simulate the fat tree DC with the topology parameters and the power profiles respectively in Tables 3.31 and 3.32. The results are shown in Figures 3.32 and 3.33: E-WF allocates more VMs with respect to the other approaches. More in detail, FF and BF behave similarly, WF improves the former two strategies, and E-WF reaches the best results allocating on average five VMs more than the other strategies. In E-WF, the probability density function is more dense around the value of 25, while it rapidly decreases increasing or decreasing the number of rejected VMS. Instead, the other strategies follow a different trend and their probability density function has two local maxima. Moreover, for E-WF, the probability to reject more than 30 VMs is very low differently from the other strategies. In Figure 3.33 we show the ranking values:

Table 3.31: *Fat tree topology parameters.*

Parameter	Value
Server number	16
Server bandwidth	100 Mbps
Access switch bandwidth	500 Mbps
Aggregation switch bandwidth	1000 Mbps
Core switch bandwidth	10000 Mbps

Table 3.32: *Fat tree power profile.*

Device	Idle power [W]	Peak power [W]
Server	100	150
Access	150	300
Aggregation	250	500
Core	500	1000
Gateway	700	1200

Table 3.33: *Spine-and-leaf topology parameters.*

Parameter	Value
Server number	16
Server bandwidth	100 Mbps
Leaf switch bandwidth	100 Mbps
Spine switch bandwidth	1000 Mbps

Table 3.34: *Spine-and-leaf power profile.*

Device	Idle power [W]	Peak power [W]
Server	100	150
Leaf	125	300
Spine	500	1000
Gateway	700	1200

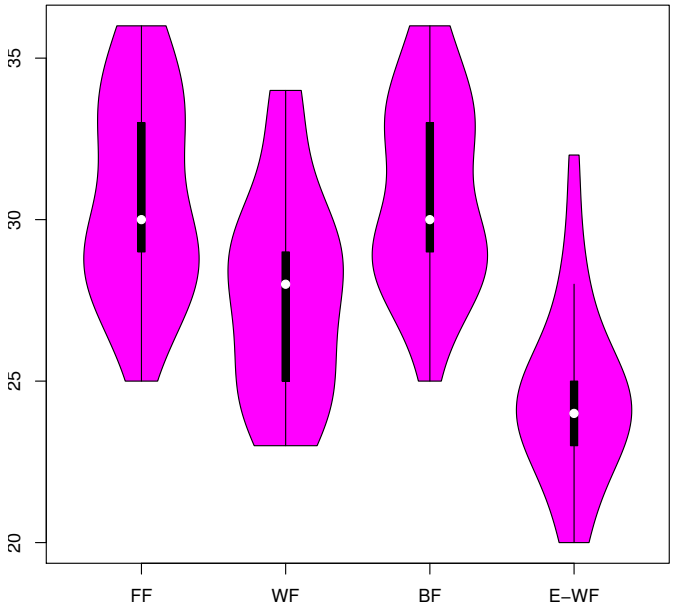

Figure 3.32: *VM requests rejected out of 100 demands simulating a three-tier fat tree topology.*

E-WF allocates almost every time the highest number of VMs and performs better than the other allocation strategies.

We consider now the spine-leaf topology with the parameters reported in Tables 3.33 and 3.34, and we show the results of our simulations in Figures 3.34 and 3.35: E-WF improves the other strategies allocating more VMs and reaching high ranking values, obtaining similar results as in the fat tree DC scenario.

Power consumption evaluation As mentioned in previous sections, we measure the power consumption when our strategies allocate the same 50 VM requests without any rejections. The 50 requests are divided in 16 balanced, 16 cpu-intensive, and 17 bandwidth-intensive. We characterize all the network devices with the power profiles described in Tables 3.35 and 3.36 for the two topologies; in both cases, we characterize some devices with two different consumption profiles. More specifically, when we adopt the fat tree topology, half of the access and aggregation switches in the same pod are characterized with a "more consuming" profile, and the same characterization is realized in the leaf-spine topology, in which half of the server NICs and leaf switches are characterized by two different power profiles.

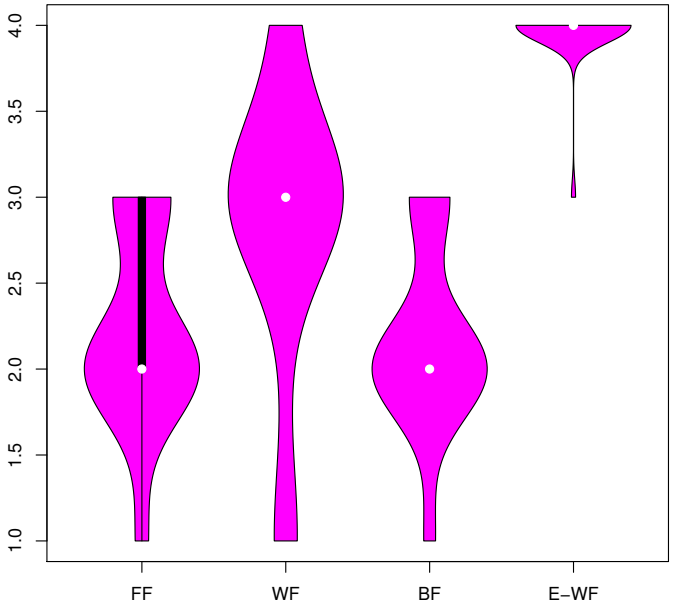

Figure 3.33: *Allocation strategy ranking after 100 VM requests simulating a three-tier fat tree topology.*

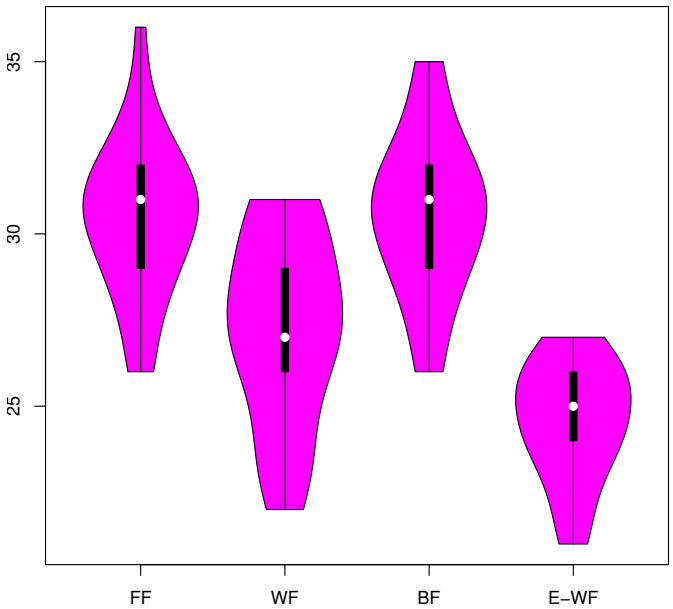

Figure 3.34: *VM requests rejected out of 100 demands simulating a spine-and-leaf topology.*

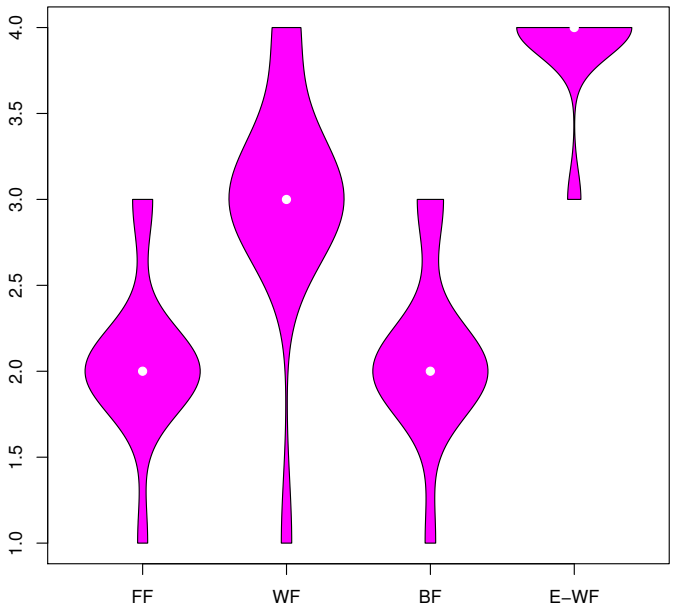

Figure 3.35: *Allocation strategy ranking after 100 VM requests simulating a spine-and-leaf topology.*

Table 3.35: *Fat tree power profile.*

Device	Idle power [W]	Peak power [W]
Server	100	150
Access	150	300
	250	500
Aggregation	250	500
	700	1200
Core	500	1000
Gateway	700	1200

Table 3.36: *Leaf-Spine power profile.*

Device	Idle power [W]	Peak power [W]
Server	100	150
	150	300
Leaf	125	300
	500	1000
Spine	500	1000

We compare the PA allocation strategies with the SPF ones, and we show the results for the two topologies in Figures 3.36 and 3.37. PA allocators save more power with respect to the SPF approaches. Among the PA allocation strategies, all of them have more or less the same trend except FF which always increases the power consumption. The PA network allocator significantly reduces the power consumed by the network devices in both the simulated environments.

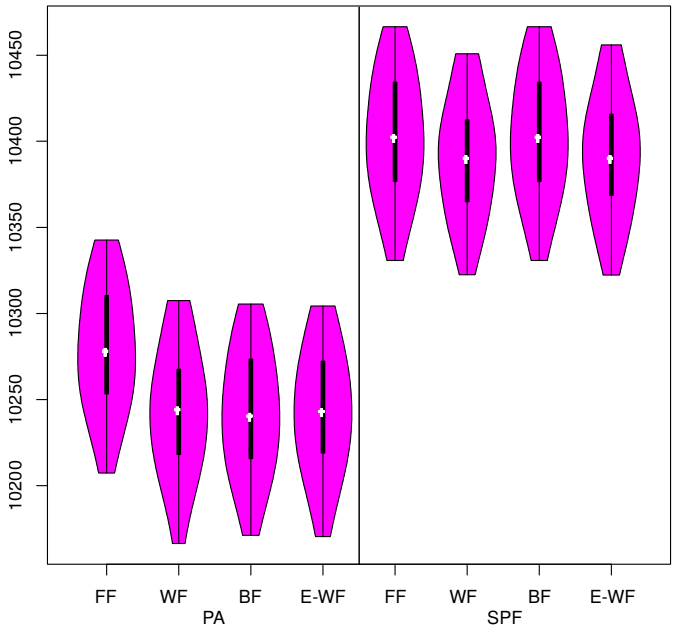

Figure 3.36: *Power consumption after 100 VM requests (Fat tree).*

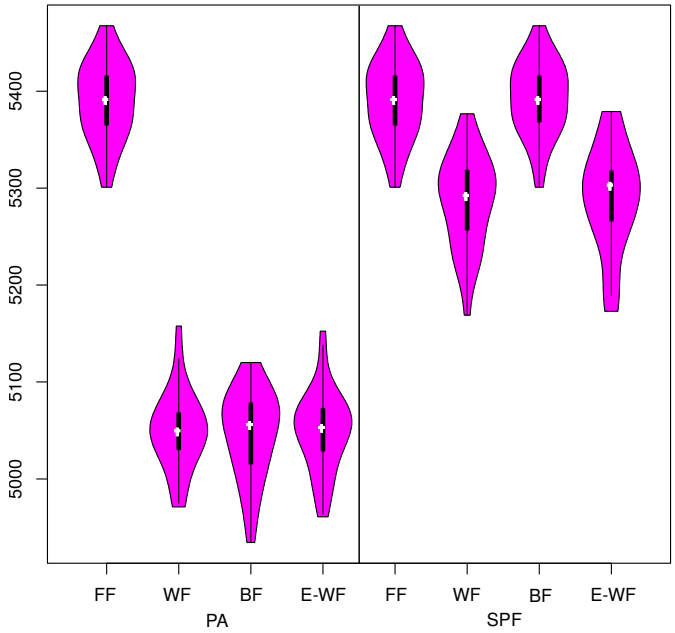

Figure 3.37: *Power consumption after 50 VM requests (Spine-and-leaf).*

CHAPTER 4

Data Center

DC network power consumption depends on switch idle and peak consumption, link load, and Network Interface Controllers (NICs) utilizations. In this scenario, retransmissions affect link utilizations and impact directly on power consumption. This chapter focuses on Software Defined DC network management analyzing two aspects separately: load balancing and resilience strategies for network flows; and scheduling of MPI collective communications in distributed applications. Both the aspects aim to improve the utilization of network resources and DC performance.

4.1 Load balancing and traffic recovery

In the last years, SDN has emerged as a novel paradigm for programmable networks by decoupling control and data planes [60,61]. Indeed, SDN provides an abstraction of the network infrastructure, over which it is relatively easy to build new functionalities and advanced services. The most deployed SDN protocol, OpenFlow [49, 54, 62], allows to set into OpenFlow-compliant switches forwarding rules established by a centralized intelligence, called controller. More in detail, network administrators can run on the controller management applications and resource optimization tools, implementing functions such as QoS, load balancing and class-based traffic recovery [12].

Traditional intra-domain routing protocols (e.g., RIP, OSPF, IS-IS) select a single path between any source-destination pair, thus leading to not uniform utilization of network resources and undifferentiated treatment of traffic flows with heterogeneous network requirements. Load balancing tries to distribute flows over the entire network, and we achieve it by defining variable costs of the links, depending on their current utilization estimated through OpenFlow statistics. In more detail, in our framework load balancing is carried out on a per-flow basis and routing changes are applied only to

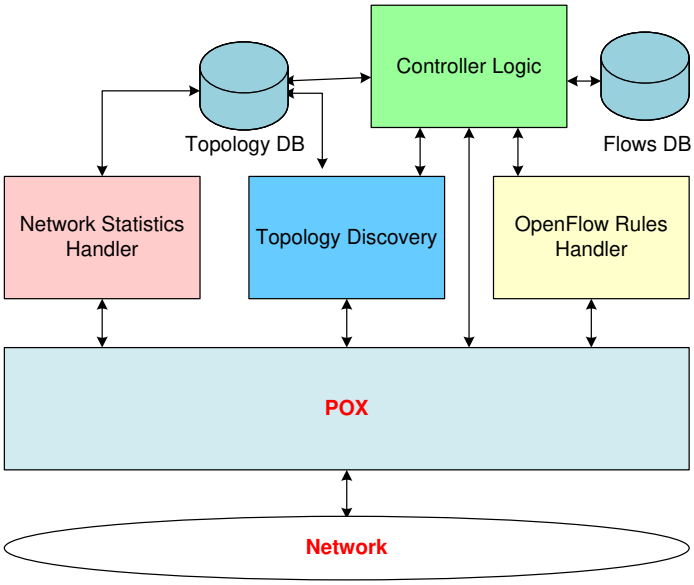

Figure 4.1: *Controller architecture.*

the new flows, leaving the *old rules* for the already active ones. The update of the links statistics is performed periodically according to a configurable time interval, which can be chosen as a trade-off between the typical time-scales of the traffic dynamics and the introduced overhead (both in terms of signaling traffic and computational processing time).

Resilience is an open issue for SDN networks since, after link or node failures, traffic paths are no more automatically reconfigured as occurs in IP/MPLS networks [71]. Indeed, Flow Table entries are now handled by the controller, but not directly updated in case of failure. Our SDN controller deals with this issue, implementing different recovery strategies (Protection and Restoration, which will be described in Section 4.1.1), based on class-of-service requirements.

Some previous works dealt with resilience in SDN, but they did not consider network traffic dynamics. Sharma et al. [68], introduced a fast restoration mechanism leveraging the configuration of static entries to forwarding table of OpenFlow switches. On the other hand, we dynamically assign variable cost to the links based on the current utilization. Routing paths are computed accordingly, leading to traffic distribution over the entire network. These topics have also been addressed by the authors in a recent previous work [12]. The main novel contribution of our framework is twofold. At first, we provide an enhancement of the controller architecture introducing new functionality. Then, experiments assess the effectiveness of our approach in a new case-study, based on fat-tree architecture. Such scenario is particularly relevant since the DC is one of the main application domains, in which OpenFlow drives innovation.

4.1.1 Controller Architecture

Our controller consists of four main modules (see Figure 4.1) that are described in the following.

Topology discovery module

The topology discovery service provides and updates information about the network infrastructure, i.e., hosts, switches and interconnection links. Such information is archived in the topology database. As the control application is built on top of a POX SDN controller [5], the topology discovery service is designed starting from two already existing software components: the *host.tracker* and the *openflow.discovery* [6].

It is worth highlighting that in case of DCs, the exact reconstruction of the hierarchical topology (typically 2 or 3 tiers)requires the identification of Edge, Aggregation and Core Switches. Since openflow.discovery does not provide this capability, the topology discovery service of our SDN controller is enhanced with an ad-hoc module exploiting ICMP packets sent by hosts to the DC gateway and vice versa and a-priori knowledge of servers and DC gateway address pool.

Network Statistics Handler

The Network Statistics Handler (NSH) retrieves port statistics from each switch S_i of the network and updates the topology database with the current value of link utilization. We estimate the utilization for each directed link, with bandwidth B connecting S_i to S_j as explained below. S_j keeps track of the number $N(t)$ of bytes received until time t. Using OpenFlow, NSH retrieves the value of $N(t)$ every T_s seconds. The quantity $[N(kT_s) - N((k-1)T_s)]$ represents the bytes received in the k-th time interval, and it is used to compute the link utilization $\hat{u}(kT_s)$. This value is used to update the cost of the corresponding link according to a suitable user-defined function (see Section 4.1.2 for a case study). In case of bidirectional links the cost is independently calculated for each direction.

Moreover, NSH is able to retrieve statistics at flow level and to keep track of mappings between flow-table entries and flows average rate that is estimated using the same approach applied at the port level. This information is contained in the so-called *flows database* and used for rerouting purpose as highlighted in the next subsection.

OpenFlow Rules Handler

This module translates the path information generated by the *Controller Logic* into OpenFlows rules and is responsible for adding/removing them in each switch.

Controller Logic

This module computes the flow paths taking into account load balancing issues, and it provides class-based traffic recovery when failures occur.

Starting from the information contained in the topology database, the Dijkstra's algorithm calculates a set of updated shortest path trees (see Figure 4.2), each rooted at a different switch of the network. This operation is triggered by any topology or cost change in the topology database and the Shortest Path Tree is used by the new flows. In more detail, the edge switch sends a PacketIn message to the OpenFlow controller when a new flow is generated. Based on the topology database mappings, this module associates to source and destination hosts the switches they are connected to, thus identifying the corresponding path.

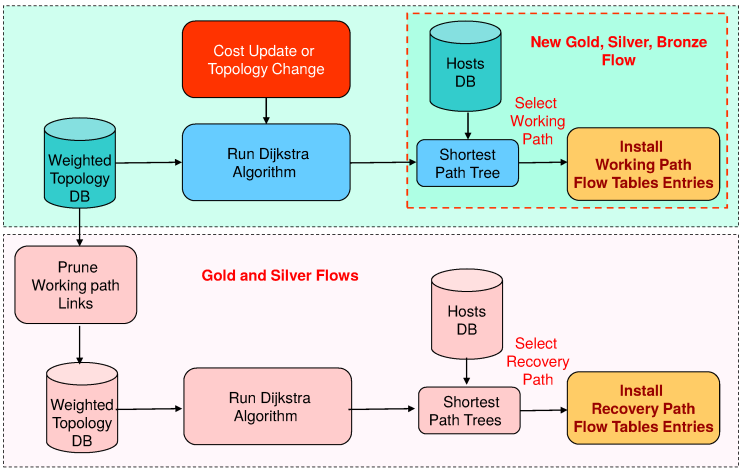

Figure 4.2: *Class-based path computation with load balancing.*

For traffic recovery purposes three classes, characterized by an increasing level of protection, are available: *Bronze*, *Silver* and *Gold*. In our architecture Bronze flows require a reactive traffic recovery strategy (*Restoration*), i.e., the Recovery Path (RP) is computed, and eventually activated, only after the Working Path (WP) failure. Silver flows benefit from a proactive technique (*Protection*): WP and RP are simultaneously computed, but OpenFlow rules for RP have a lower priority. Therefore, RP rules are activated only after removing WP rules. Gold flows also benefit from a proactive traffic recovery strategy, but WP and RP have the same priority level, so that packets are forwarded on two *link-disjoint* paths at the same time.

Failures are detected by means of OpenFlow PortStatus messages sent by switches to the controller; such messages are processed by the *Controller Logic* and the traffic recovery procedure (see Figure 4.3) is activated. From the flows database, the module retrieves all the flows forwarded on the broken link and the class membership information. Gold flows do not require any action as traffic is already replicated on the RP. The remaining flows are ordered based on class and bandwidth to minimize losses during the recovery phase. In case of Silver flows the *Controller Logic* instructs the *OpenFlow Rules handler* to remove the flow rules. Instead, Bronze flows require a longer recovery procedure. At first, the *Controller Logic* creates a new topology, pruning the broken link, and then runs the Dijkstra's algorithm, searching for the RP. If such a path is found, the *OpenFlow Rules Handler* installs the needed rules in the switches flow-table, otherwise, the flow is not recovered.

4.1.2 Experimental Results

Our controller has been developed within the POX [5] environment, a Python-based platform for the development and prototyping of SDN applications.

We present the results of the experimental tests carried out to evaluate the effectiveness of our control application, tested within the Mininet emulation environment [4].

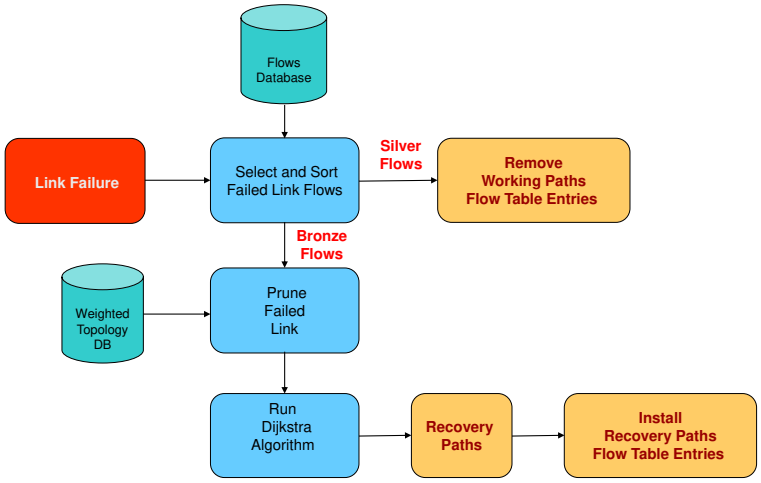

Figure 4.3: *Class-based traffic recovery.*

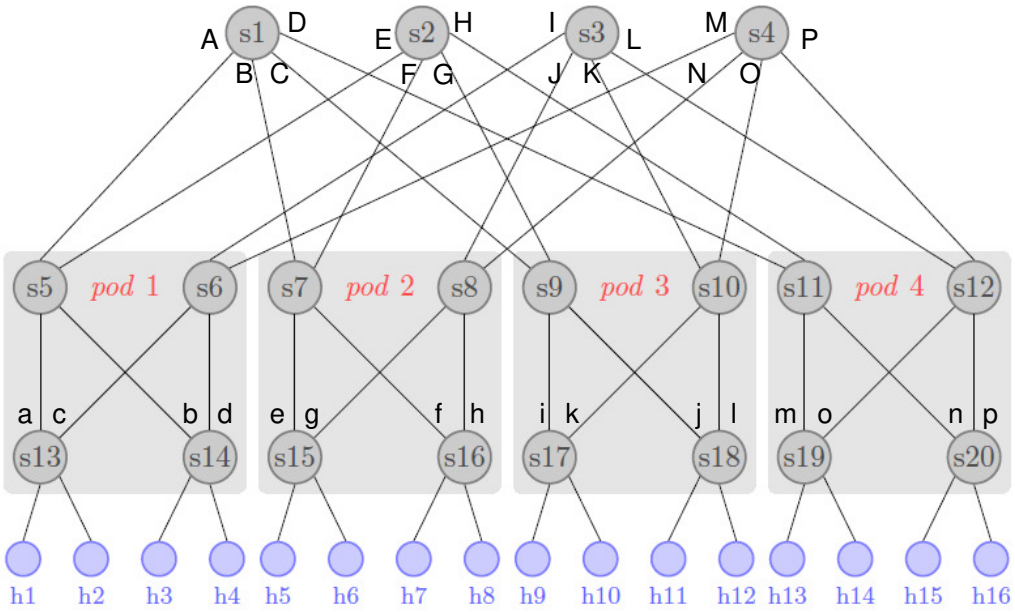

Figure 4.4: *Emulated topology.*

In more detail, we focus on networks with a fat-tree architecture, typical of DCs (see Figure 4.4), in which each VM can exchange data with all the other VMs. VM traffic is

generated using Iperf [2] and a polling time $T_s = 5$ seconds is assumed for the update of the traffic statistics by the NSH.

In the following, we suppose that a flow is uniquely identified by the 5-tuple

$$< src\_ip, dst\_ip, proto\_ip, src\_port, dst\_port > .$$

However, scalability can be achieved through aggregation, for instance considering all the traffic flows exchanged between each pair of VMs.

In order to assign a cost $c_{ij}[k]$ to the corresponding link (i, j), we consider three wide-sense increasing functions $f_i : [0, 1] \rightarrow [\alpha, \beta]$ (in the experiments we set $\alpha = 10$ and $\beta = 1000$):

$$f_1(u) = \alpha + (\beta - \alpha)\, u \tag{4.1}$$

$$f_2(u) = \alpha + \beta^u - \alpha^u \tag{4.2}$$

$$f_3(u) = \beta - \beta^{1-u} + \alpha^{1-u} \tag{4.3}$$

Moreover, to avoid route fluttering in case of small changes in the link utilization, we introduce a quantization step of 0.1 for the input values.

Note that the three functions weight in a different way the link utilization and the network administrator is free to choose the function which best suits her purposes. In more detail, after the quantization f_1 leads to a staircase function with equal height steps, while the height of the steps in f_2 and f_3 are progressively increasing/decreasing, respectively.

Load balancing

The following tests compare our load balancing strategies f_i ($i = 1, 2, 3$) based on dynamic cost assignment with a standard Shortest Path First (SPF) policy (i.e., Fixed Cost (FC) cost assignment). In our experiments each VM runs two different processes, namely Iperf client and server. Network traffic is generated according to the well-known heavy-tailed ON/OFF source model originally proposed in [76] to characterize the high variability and long memory features of actual traffic flows. In detail, ON period lengths follow a Pareto distribution with shape $\alpha = 1.5$ and location $k = 20$ s, corresponding to a mean activity period $T_{\text{ON}} = 60$ s. OFF intervals also follow a Pareto distribution, with $\alpha = 1.5$, $k = 10$ s and, thus, a mean value $T_{\text{OFF}} = 30$ s. In the following, we take separately into account UDP and TCP flows.

UDP flows In the emulation, the rate R of each ON/OFF source is uniformly distributed in the range $[r_{\text{inf}}, r_{\text{sup}}]$. To compare the performance of the cost assignment policies, we considered two different scenarios, low and high network utilization. Table 4.1 reports the average loss rates for the different policies. In case of low utilization the performance are quite similar with limited losses also in case of FC. On the contrary, for high utilization our variable cost strategies outperform FC, with slightly higher losses when f_3 function is considered.

TCP flows In these experiments the throughput depends only on the congestion status of the network as sources are supposed to be greedy.

Figures 4.5 to 4.8 show the average utilization for all the links of the emulated network, considering both directions separately, where the labels are denoted by the

Table 4.1: *Average Loss Percentages: UDP Flows.*

Transmission Rate (Mbps)	**Average Loss** (%)			
	f_1	f_2	f_3	**FC**
$R \in \mathcal{U}[0.064, 2]$	0.078	0.083	0.085	0.080
$R \in \mathcal{U}[0.8, 3.2]$	0.438	0.412	0.446	5.462

Table 4.2: *Average TCP Throughput.*

Average Throughput (Mbps)			
f_1	f_2	f_3	**FC**
5.11	5.12	4.81	3.09

letters introduced in Figure 4.4. Since FC always uses the shortest path between pairs of edge switches, some links are never used, while load balancing policies lead to the utilization of all links, with a distribution depending on the specific function f_i. This better use of network resources leads to a higher throughput as highlighted by Table Table 4.2. Indeed, in case of FC the presence of bottleneck links shared by several connections determines higher losses and throughput reduction.

Traffic Recovery tests

In this subsection we discuss some functional tests, aiming at validating the recovery procedure under two different approaches, known in the literature as Best Fit (BF) and Worst Fit (WF) [3]. The first strategy looks for the paths with the highest utilization, bringing to a consolidation policy, while the other one selects the least utilized paths, thus extending the load-balancing function to the recovery process.

As already stated in Section 4.1.1, protection and restoration strategies are used according to the traffic class. For sake of brevity, we only focus on the latter. In detail, let us consider Figure 4.4 and assume that two bronze flows are active, with the following features:

1. flow 1 (B=6 Mbps) from h_1 to h_5 with WP $WP_1 = s_{13} \rightarrow s_5 \rightarrow s_1 \rightarrow s_7 \rightarrow s_{15}$
2. flow 2 (B=2 Mbps) from h_2 to h_6 with WP $WP_2 = s_{13} \rightarrow s_6 \rightarrow s_3 \rightarrow s_8 \rightarrow s_{15}$

Then, suppose that link $s_3 - s_6$ fails. When this event is detected by the controller, a new path is computed for flow 2: as expected, in case of BF the new RP RP_{BF} is the same as WP_1, while WF chooses a completely disjoint path (i.e., whose links were not used by any traffic) $RP_{\text{WF}} = s_{13} \rightarrow s_6 \rightarrow s_4 \rightarrow s_8 \rightarrow s_{15}$. This behaviour is highlighted by the following figures, depicting the average rate as calculated by the *Network Statistics Handler* (with $T_s = 5$ s) for the involved links. In more detail, flow 2 stops utilizing link $s_6 \rightarrow s_3$ (see Figure 4.9) after the failure and starts using link $s_4 \rightarrow s_8$ in case of WF (see Figure 4.10) or link $s_7 \rightarrow s_{15}$, shared with WP_1, in case of BF (see Figure 4.11). This behaviour is confirmed by Figure 4.12, as link $s_8 \rightarrow s_{15}$ belongs to WP_2 and RP_{WF}, but not to RP_{BF}.

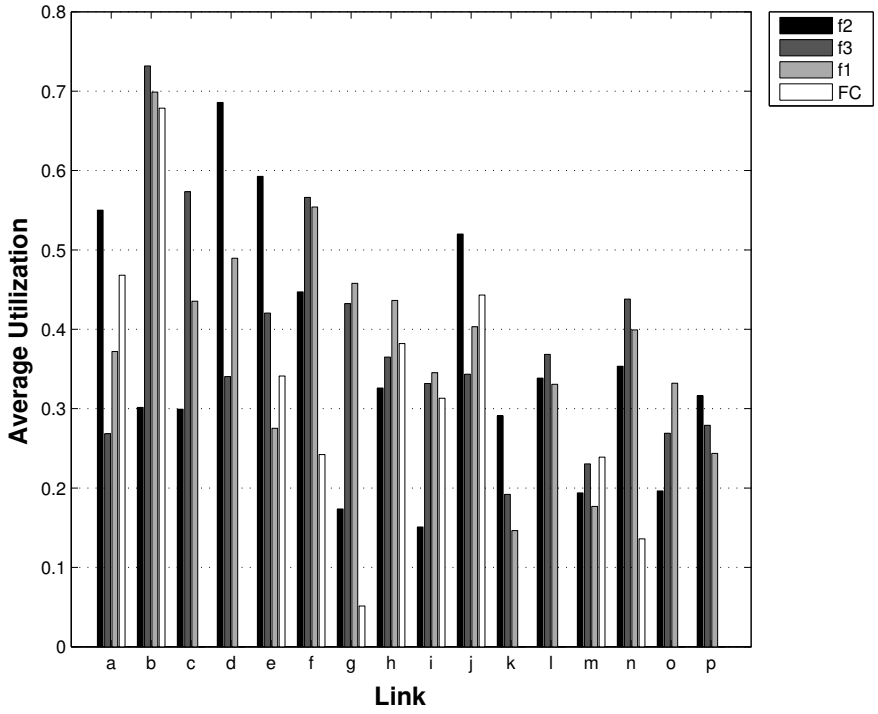

Figure 4.5: *TCP flows average link utilizations: Aggregation → Edge direction.*

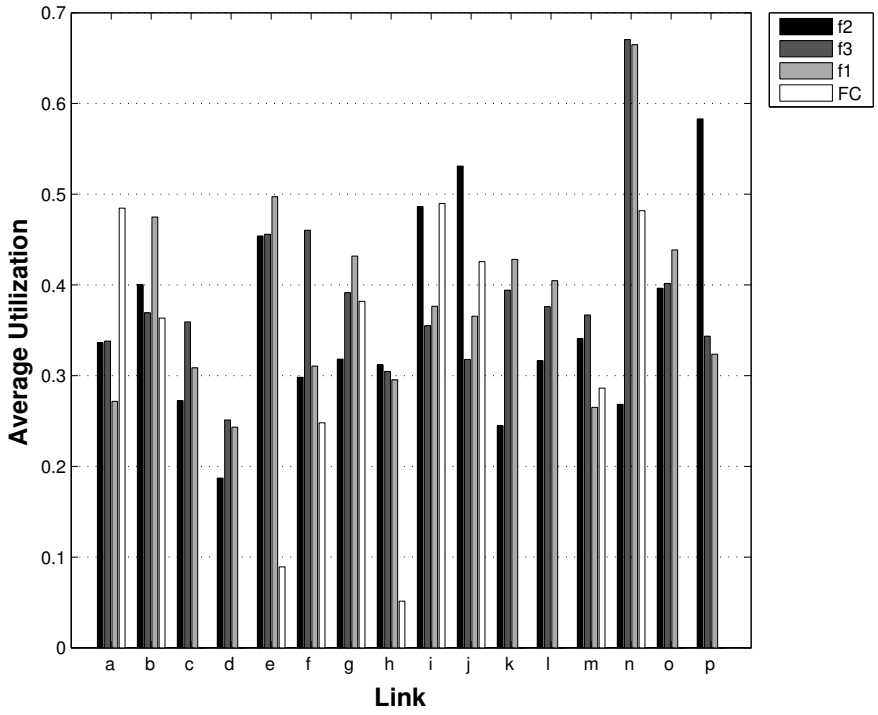

Figure 4.6: *TCP flows average link utilizations: Edge → Aggregation direction.*

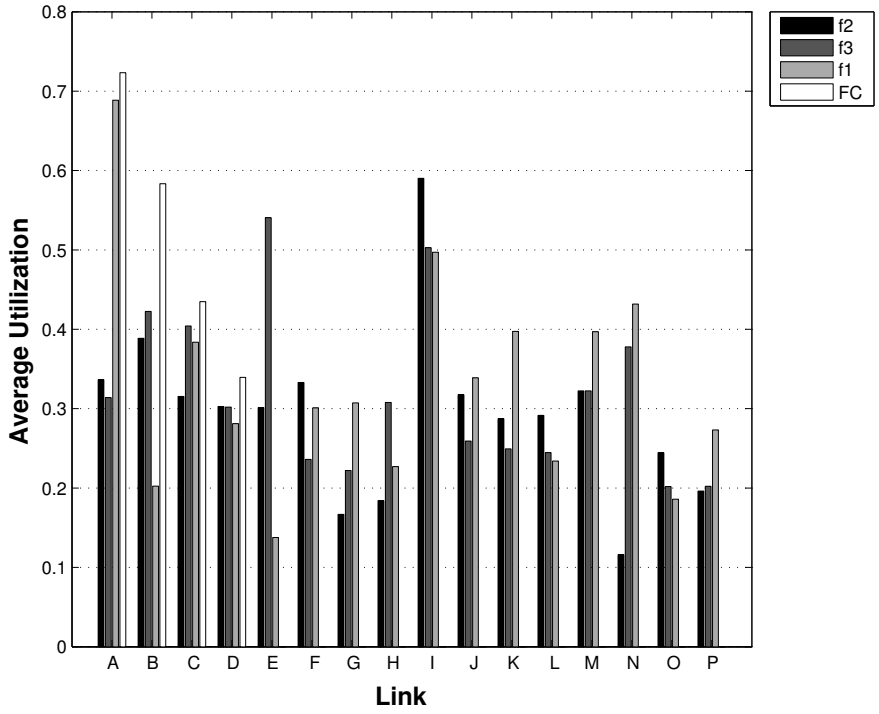

Figure 4.7: *TCP flows average link utilizations: Core → Aggregation direction.*

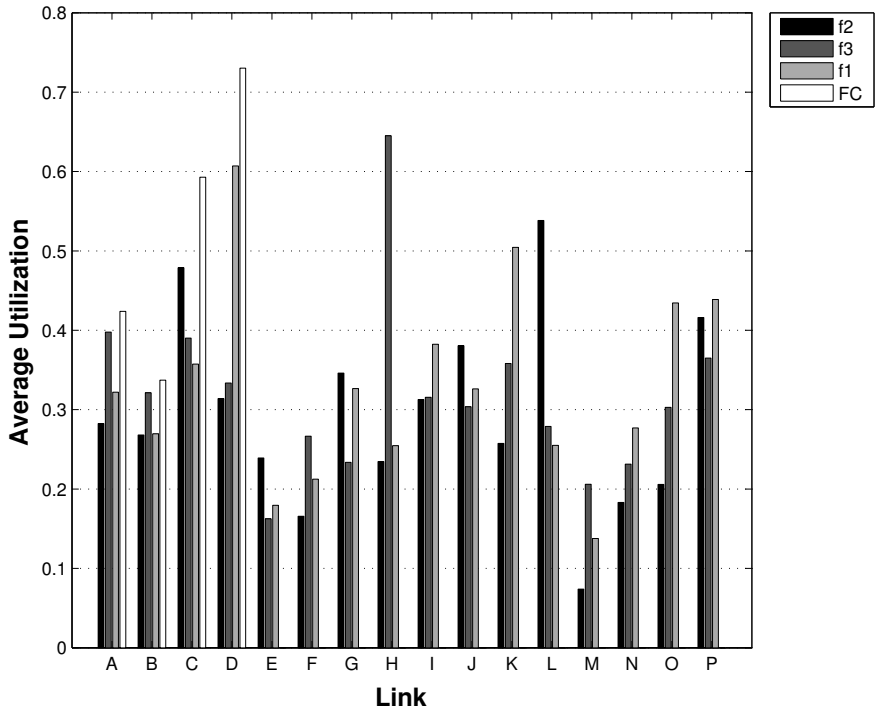

Figure 4.8: *TCP flows average link utilizations: Aggregation → Core direction.*

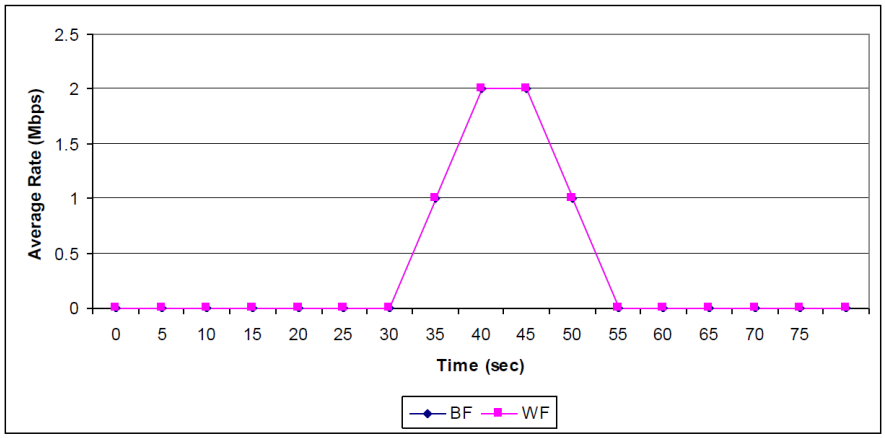

Figure 4.9: *Link* $s_3 - s_6$.

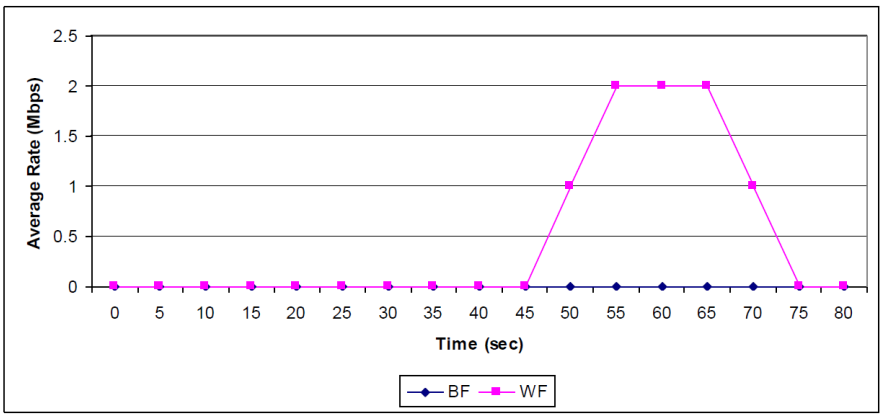

Figure 4.10: *Link* $s_4 - s_8$.

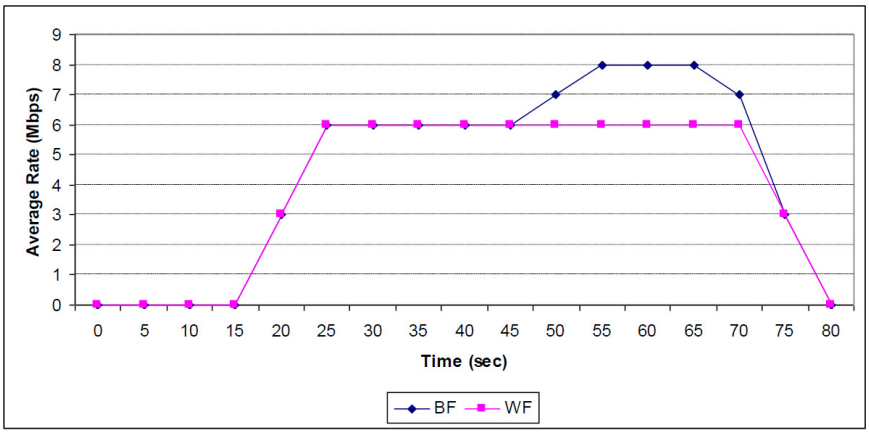

Figure 4.11: *Link* $s_7 - s_{15}$.

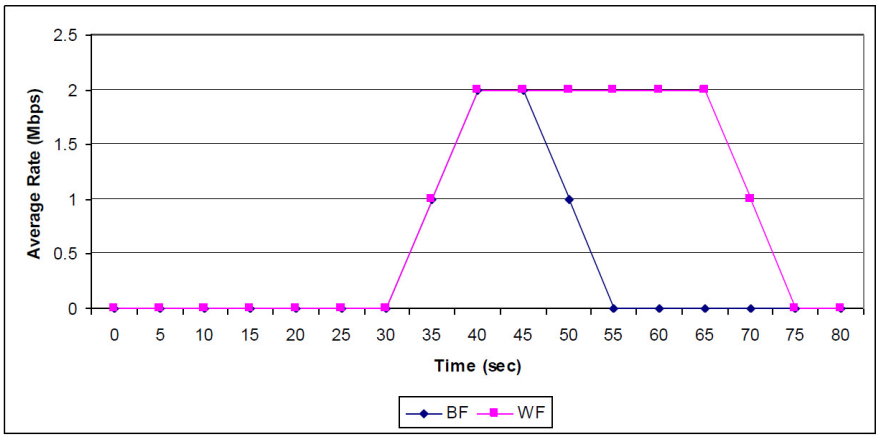

Figure 4.12: *Link* $s_8 - s_{15}$.

4.2 MPI Collective Scheduling

4.2.1 MPI Collective

Collectives (or collective communications) are particular kind of communications among MPI entities which involve more than two processes. Currently, the existing collective communications defined by the standard are:

- *multicast*, when a single sender forwards the same message to multiple destinations; this collective is named *broadcast* when all the possible destinations are involved;
- *reduce*, when multiple senders send the same message to a single destination process;
- *scatter*, a single sender spreads chunks of an array to different destinations;
- *gather*, multiple senders transmit chunks of an array to a single destination;
- *barrier*, that is used as a synchronization barrier for processes;
- *all-to-all*, where every process communicates with all the others.

In the current proposal, we focus on multicast/broadcast but in some cases our approach could be generalized for other collectives. The support for the remaining collectives will be subject of investigation of future researches.

4.2.2 LogGP Model

One of the most adopted model to evaluate MPI communication is the LogGP model [14] that is an extension of the LogP model [26]. LogGP model abstracts the communication performance of MPI communication considering four parameters: the communication latency incurred in sending a message from its source processor to its target processor (*L*), the overhead (*o*) or the length of time that a processor is engaged in the transmission or reception of each message, the minimum gap between two messages or the minimum time interval between consecutive message transmissions or consecutive message receptions at a processor (*g*), the bandwidth for long messages (*1/G*), and the number of processors (*P*). In 4.3 we report the values for two libraries running on the Meiko CS-2 super-computer. When considering TCP/IP communications, the network latency overcomes all the other parameters, so we might easily neglect them in this work without losing generality.

Parameter	Split-C	Elan lib
L	8.6 μs	10 μs
o	1.7 μs	3.8 μs
g	14.2 μs	13.8
$\frac{1}{G}$	33.3 MBps	43 MBps

Table 4.3: *LogGP parameters.*

Our proposal focuses on communications between processes of distributed applications. Each process could be a source or a destination in the collective communication. When a process receives a copy of the message, it may become a sender when the message is received and analysed. We name *software-routing overhead* the time spent to analyze the received message and decide the new possible destinations. Usually, the software-routing overhead is order of magnitude higher than the single hop communication between two switches. For this reason a receiver is not able to resend immediately the message after its reception, but it experiences a software delay.

4.2.3 Heuristic

General description

We represent the DC internal structure with a directed graph $\Gamma(V, E)$; servers and switches are the nodes while all the links are the edges of the graph. We exploit the graph topology to optimize the MPI scheduler increasing the parallelism degree of the communication. The scheduling procedure is iterated until the end of the collective communication. During each iteration, the heuristic first evaluates all the available senders and receivers (i.e., the ones which are not transmitting or receiving anything at that time), and it divides the graph in Reference Areas (RAs) used to map senders with receivers, avoid network congestion, and improve the scheduling phase; in the second phase, it generates the flow allocation demands for each source-destination pair; and in the last phase, it collects all the information about the link occupancy and network flows that are already allocated, and it solves a multi-commodity flow problem that determines which demands will be allocated. We describe the three phases below.

Recursive distance-based graph partition

As briefly described, the heuristic splits all the available destination nodes in RAs. We partite the graph Γ in a set of RAs such that:

$$RA_i \cap RA_j = \emptyset \quad \text{if } i \neq j \tag{4.4}$$

$$\bigcup_i RA_i = \Gamma \tag{4.5}$$

The assignment of nodes into RAs simplifies the transmission demands generations and reduces the number of flows that will be evaluated in the allocation phase.

Within a RA, a single process called *leader* owns the message to be transmitted or is receiving a copy of it; each RA contains exactly one leader, and when another process in the same RA is eligible to become a leader (i.e., is receiving or will receive soon a copy of the message) our heuristic furthermore splits the original RA in two new RA so as to each RA contains exactly one leader.

We adopt a *distance-based criterion* to split RAs, defining the *distance* between a pair of nodes α and β the length of the shortest path between α and β in terms of number of edges traversed. The distance-based criterion assigns each destination node to the less-distant leader's RA; we apply it both at the beginning of the computation and every time there is a process eligible as new leader. The splitting procedure, performed applying the distance-based criterion, is limited only to a single RA and never affects the limiting ones: even if some destination nodes in the limiting RAs are closer to

the new leader, those nodes are not assigned to the new RA. Following this procedure we are now able to assign an integer rank based on the number of splitting operation applied to each RA. We assign rank zero to the original graph Γ, rank equal to one to the very first RAs that are generated, and we increase the rank of each new RA after the partition procedure. We provide an example to better clarify what stated; if A is a RA having rank equal to r and it is split in n new RAss $B_1, B_2, ..., B_n$ we have that:

$$\text{rank}(B_i) = r + 1 \; \forall i \in [1, n]. \tag{4.6}$$

The previous example shows that we can organize all the RAs as a tree where A is the *parent* of each new *child* B RA.

Demand generation

The second phase is the demand generation. Each non-transmitting leader is in charge to forward a copy of the message within its RA generating all the necessary traffic demands. In order to increase the parallelism degree, the heuristic leverages on the distance-based criterion also in this phase: available leaders search in their own RA for the destination node having the highest distance. We call it*pivot*, and it is marked as an *eligible* leader. Then, we apply the partition procedure as described above, before performing the transmission, so two new partitions are created: the first which contains the original leader and the second that includes the pivot node. At this time, we generate demands from the leader towards each possible destination node in the pivot's RA, but only one of them will be allocated in the next phase. The new leader of the second partition will be the one which receives the incoming transmission, and it is worth noting that the pivot node may not become the leader of its RA.

If for some reasons in a RA there is no available or reachable destination node, the chosen leader will inspect bottom-up all the RAs in the RA tree until at least one destination node is found or the whole graph is explored. Then, we generate the traffic demands according to the result of this search.

Multi-commodity flow allocation

In the last phase, we allocate some traffic demands solving a multi-commodity allocation problem. Our goal is to maximize the number of allocated flows and their bit-rate avoiding network congestion. All the adopted symbols are listed in table 4.4.

Symbol	Description
d	total number of demands
l	total number of links in the graph
b_i	the bit-rate assigned to the i-th demand
c_i	the bit-rate requested by the i-th demand
r_j	the maximum capacity of the j-th link
ω_i	the weight assigned to the i-th demand
a	the allocation vector $\in \mathbb{R}^d$
u	the link-rate matrix $\in \mathbb{R}^{d*l}$

Table 4.4: *List of the adopted symbols.*

We define the allocation vector as:

$$a_i = \begin{cases} 1 & \text{if the i-th demand is allocated;} \\ 0 & \text{otherwise.} \end{cases} \tag{4.7}$$

and the link-rate matrix as:

$$u_{ij} = \begin{cases} b_i & \text{if the i-th demand is allocated on the j-th link;} \\ 0 & \text{otherwise.} \end{cases} \tag{4.8}$$

We assign different weights to communications that are done within the same RA and in this case the weight is higher, or within parent RAs that always have a lower priority. If a demand might be allocated on multiple paths, our algorithm chooses the shortest one. The heuristic solves the following linear problem:

$$\text{maximize :} \quad \sum_{i=1}^{d}(a_i * b_i * \omega_i - \sum_{j=1}^{l} u_i j); \tag{4.9a}$$

$$\text{s.t. :} \quad 0 \leq b_i \leq c_i \qquad \forall i = 1..d; \tag{4.9b}$$

$$a_i * b_i \geq b_i \qquad \forall i = 1..d; \tag{4.9c}$$

$$\sum_{i=0}^{d} b_i * u_i j \leq r_j \qquad \forall j = 1..l. \tag{4.9d}$$

We included some other constraints such as:

- the flow conservation principle;
- cycles-avoidance;
- multiple receptions or transmissions avoidance for each node at the same time;
- path integrity.

At the end of the computation, the results are collected and the iteration ends. The whole algorithm is repeated until the end of the collective communication.

4.2.4 Experiments and Results

In our simulations, we set the latency per hop equal to 1 ms and the software-routing overhead to 10 ms; we use CPLEX [44] to solve the demand allocation while the graph management framework is a self-programmed environment written in Java.

We simulate our approach for different network topologies: line, mesh/torus, and tree, but more topology can be easily implemented and tested. In our figures, we display the node types through different colors: the original sender is green, the destination processes are yellow, and the switches are white.

In a tree topology, our scheduling is optimal in terms of throughput and communication time: we computed the same scheduling obtained in [31] splitting each subtree in two parts (i.e., the left and right subtrees) starting from the root up to the highest internal nodes.

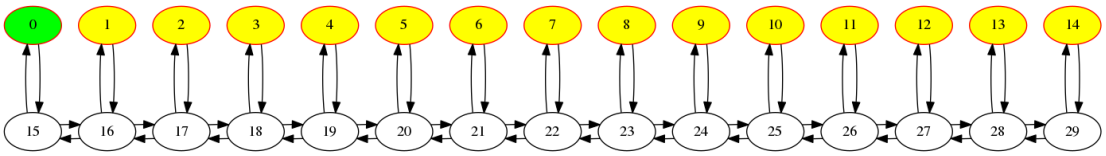

Figure 4.13: *Line topology.*

Source	Destination	Start time [ms]	End time [ms]
0	8	0	10
0	4	10	16
0	2	16	20
0	1	20	23
8	10	20	25
0	3	23	28
8	10	25	29
4	6	26	30
8	9	29	32
4	5	30	33
8	12	32	38
4	7	33	38
11	13	38	42
11	14	42	47

Table 4.5: *Results for the line topology.*

We tested our heuristic with 15 servers connected through the line topology depicted in fig. 4.13 reaching the maximum parallelism degree of the three processes $\{0, 4, 8\}$ in the time interval [26,28] as shown in table 4.5.

We generated a 4x4 2-D Mesh that is represented in fig. 4.14. As reported in table 4.6, we reach a maximum parallelism degree of five processes $\{0, 1, 2, 8, 9\}$ in the time interval [26,28]. We show the computed transmission scheduling in table 4.6.

The results we highlighted for both line and mesh topologies reach a high parallelism degree. If we neglect the routing overhead, the maximum parallelism degree p we can achieve is:

$$p = log_2(n); \tag{4.10}$$

where n is the number of processes involved in the collective. Considering the software-routing overhead, p is an upper-bound that sometimes can not be achieved. Our topology-aware scheduler reaches a maximum p that is very close to the ideal one.

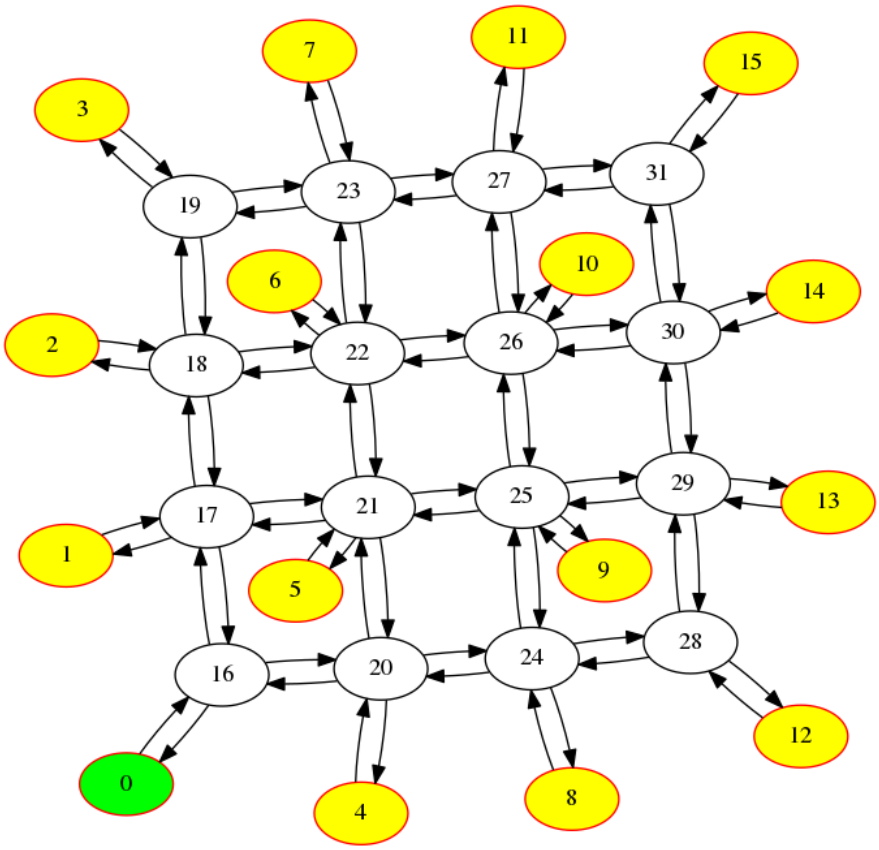

Figure 4.14: *Mesh topology.*

Source	Destination	Start time [ms]	End time [ms]
0	9	0	5
0	2	5	9
0	8	9	13
0	1	13	16
9	6	15	19
0	4	16	19
0	5	19	23
2	3	19	22
9	13	19	24
2	12	22	29
0	7	23	29
8	10	23	29
9	11	24	28
1	14	26	32
9	15	28	33

Table 4.6: *Results for the mesh topology.*

CHAPTER 5

Internet of Things

One of the leading standard in the IoT is the IEEE 802.15.4-2015 [7], which is the third revision of the IEEE 802.15.4 Standard for Low Rate Wireless Networks. IEEE 802.15-2015 [7] standard presents TSCH that provides high reliability and low power consumption to various industrial applications. The channel agility of TSCH is the key to its ultra-high reliability and it has been described as the future in making networks deterministic. Before the development of IEEE 802.15.4-2015 standard, energy efficiency and deterministic QoS application requirements were not adequately addressed by the previous IEEE 802.15.4 standard. TSCH paves the way for addressing this issue, which emerges from the proprietary Time Synchronized Mesh Protocol (TSMP) [64] and has been the heart of industrial low-power wireless standards such as WirelessHART [22], ISA100.11a [45] etc.

As the industrial environment presents a vast range of applications, adopting an adequate solution for each case is vital to obtain good performance of the system. In this context, the routing and scheduling schemes associated with these technologies have a direct impact on important features, like latency and energy consumption.

In industrial environments where TSCH is now relevant, changing batteries can be dramatically expensive and difficult, thus maximizing the energy efficiency is an important problem that needs to be tackled. Even though the problem of energy efficient scheduling and routing has been widely studied for traditional wireless networks, the strict reliability requirements in industrial networks bring new challenges. To the best of our knowledge, this is the first work that focuses on energy efficiency as a design criteria in IEEE 802.15.4-2015 TSCH scheduling. In this work, we focus on scheduling in TSCH networks from the energy efficiency perspective where the gateway makes frequency allocations and time slot assignments.

The main contributions of this chapter are as follows:

- the formulation of the scheduling problem in TSCH networks as an energy efficiency maximization problem, which is a non-linear programming problem;
- the introduction of an energy-consumption model of a TSCH node;
- the proposal of a heuristic energy-efficient scheduler to overcome the complexity of the problem formulation;
- the design of a scheduling algorithm based on Vogel's Approximation method is proposed to address the pitfalls of the greedy allocation;
- the performance comparison of the scheduling algorithms with Round Robin Scheduler is evaluated by extensive simulations.

5.1 Related Works

In recent times, several works has been tailored towards scheduling in TSCH networks. Palattella et. al. developed TASA [63], a centralized scheduler that exploits an innovative matching and colouring technique of graph theory to plan the distribution of time slots and channel offsets. Accettura et.al developed DeTAS [9] to construct optimum multi-hop schedules in a distributed fashion which uses neighbour-to-neighbour signaling for gathering information about the network and traffic features.

Other works address scheduling in TSCH networks using combinatorial properties based on Genetic Algorithms. The use of GAs has been proven to be quite successful for time slot and frequency assignment in both cellular networks [41] and cognitive radio networks [38]. In Industrial Internet of Things, particularly the industrial wireless sensor network, the studies about the usage of GAs mainly revolve around the configuration of various sensor parameters such as symbol rate, modulation etc. [47], [83].

5.2 Throughput Maximization Scheduling Algorithm in TSCH Networks

In TSCH networks, all the nodes are synchronized and maintain a slotframe structure. A slotframe is a sequence of timeslots which repeats over time. A typical time slot duration in IEEE 802.15.4-2015 is 15 ms [73], which is large enough to transmit a frame and to receive an acknowledgement. In addition, TSCH networks are deterministic in nature; the actions that occur in each time slot are well known, and its communication is orchestrated by a schedule. A schedule is defined by the timeslot and channel on which a node should transmit/receive data to/from its neighbours. The TSCH schedule can be represented by a 2-D matrix with the width as the number of slots in a slotframe, and the height as the channel offsets as shown in Figure 5.1. Each node in the network only cares about the cells it participates in. The two nodes at either end of the link communicate periodically once in every slotframe.

The communication links hop over a set of available channels in a pseudo-random pattern among a slot frame. For each scheduled cell, both the sender and receiver will use the following translation function shown below to determine which frequency f should be used for communication.

$$f = F_{MAP}(ASN + channeloffset) mod N_{ch} \tag{5.1}$$

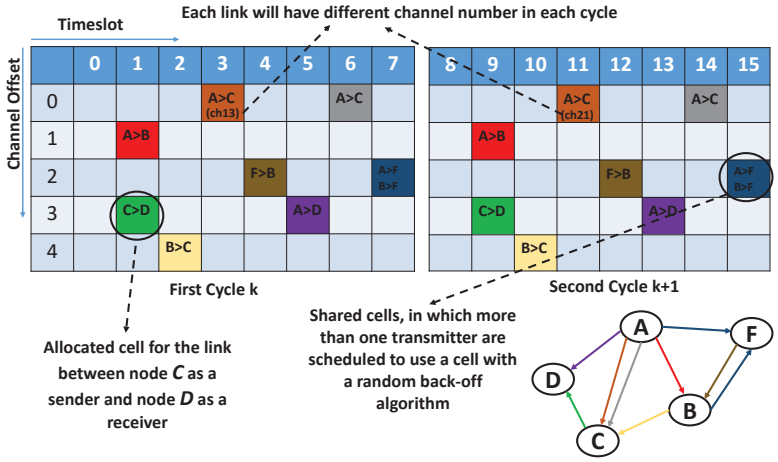

Figure 5.1: *TSCH slot-channel matrix with a simple network topology.*

where Absolute Slot Number (ASN) is a variable which counts the number of timeslots since the network was established. F_{MAP} is the mapping function to find the channel frequency from a channel lookup table. N_{ch} indicates the number of available physical channels.

Several MAC based scheduling approaches have been considered in the wireless networking literature [52], [33] but this approach cannot be applied to TSCH MAC. Nonetheless, TSCH paradigm brings this topic into the focus of research again due to the following reasons: IEEE 802.15.4-2015 standard defines the mechanism for a TSCH node to communicate, but the standard does not specify how to build an optimized schedule and to construct a schedule is policy specific; and TSCH brings new opportunities band challenges because of its time synchronization multiplexed in frequency to be able to scale up communications and to improve reliability. In this paper, we introduce the TSCH scheduling problem with deadline constraints which differs from other related works.

5.2.1 Network Model and Problem Formulation

Network Model

We model a time-slotted IEEE 802.15.4-2015 network as a graph $G = (V, E)$, where the vertices in $V = \{n_0, n_1, n_2, ..., n_{N-1}\}$ represent the network devices and the edges in E denote the devices pairs that can sustain a reliable communication. There is only one gateway denoted by n_0 and n_k, with $1 \leq k \leq N-1$ represents other nodes in the network. The scheduler resides at the gateway and determines how many packets and with which frequency each node will transmit in each time slot. Each node is equipped with a radio with communication range R_i, i.e., transmission is successful if $d_{i,j} \leq R_i$ condition is satisfied, where $d_{i,j}$ is the actual distance between node i and node j in the network. For the sake of clarity, we are given a set of nodes $k \in \{1, .., .N\}$, where each node is characterized by its frequency, timeslot and its deadline δ_k, where $f \in \{1, ..., F\}$ and $t \in \{1, ..., T\}$ denote the set of frequencies and timeslots respectively. We also denote P_k as the number of packets that can be transmitted. We also

assume that one-hop transmission of a packet consumes one time slot at most.

At the beginning of each slot frame, each node sends its state to the gateway. We denote by Q_k the number of packets in each node's buffer. We denote by $C_{k,f,t}$ the channel capacity of link $l_{k,f}$ at time t by assuming a time-varying wireless channel between each pair of nodes (sender and receiver) which depends on the channel bandwidth ***B*** and the signal-to-noise ratio ($SNR_{k,f,t}$) of the link based on Shannon's formula. $M_{k,f,t}$ is defined as the number of packets that can be sent during a slot frame.

$$M_{k,f,t} = C_{k,f,t}T \tag{5.2}$$

where T is the slot frame duration, $M_{k,f,t}$ is a function of the signal-to-noise ratio and shows the theoretical upper bound. $U_{k,f,t}$ represents the number of packets that n_k can send through f in timeslot t based on its queue size and transmission time. (i.e., $U_{k,f,t}$ is restricted by both $M_{k,f,t}$ and Q_k because we consider that node k cannot transmit more than the number of packets in its buffer). Therefore

$$U_{k,f,t} = min(M_{k,f,t}, Q_k) \tag{5.3}$$

Considering the binary variable $X_{k,f,t}$ defined as:

$$X_{k,f,t} = \begin{cases} 1 & \text{if node } n_k \text{ transmits using} \\ & \text{frequency } f \text{ in time slot } t; \\ 0 & \text{otherwise;} \end{cases} \tag{5.4}$$

$$\vec{x} = [X_{k,f,t}, k \in \{1, ...N\}; f \in \{1, ...F\}; t \in \{1, ...T\}] \tag{5.5}$$

is the scheduling decision vector with elements $X_{k,f,t}$. Note that $X_{k,f,t}$ is a function of the information available to n_k. The gateway needs to first collect network statistics (e.g. the topology information of the network and the traffic generated by each node). With the required information at the gateway, the scheduler which resides at the gateway now performs its scheduling approach. Moreover, we assume that all data transmissions are scheduled in dedicated[1] timeslots (i.e., only one transmitter is scheduled, and the cell is contention free).

We assume that each device is equipped with a half based transceiver radio, which implies that a device cannot transmit and receive at the same timeslot. We also assumed that a device cannot receive from multiple nodes at the same time. Consider node n_i, the transmission of node n_j will interfere with the transmission of n_i if

$$\mid n_p - n_j \mid \leq (1 + \triangle) \mid n_i - n_j \mid \tag{5.6}$$

for any node $n_p \neq n_i$., where $\mid n_i - n_j \mid$ is the Euclidean distance, and $\triangle$ is a guard zone specified by the protocol to prevent a neighbouring node from transmitting on the same channel at the same time. It also allows for imprecision in the achieved range of transmissions [40].

[1] In TSCH network, shared slots are allocated when multiple transmitters are scheduled for transmission to the same device simultaneously, and the standard defines a CSMA-CA algorithm to reduce the probability of repeated collisions. We do not use shared slots as we assume reliable link-layer communication

Throughput Maximization Problem

Our objective is to give each link $l \in G$ a transmission schedule, which is the list of time slots and frequencies it could send packets such that the schedule is interference-free and the overall throughput of the network is maximized. Given the values for $N, F, T, M_{k,f,t}$, the throughput maximization problem is to find the vector X given by the following binary integer linear programming (ILP) formulation to be executed by the gateway:

$$\textbf{P1:} \quad \max \sum_{k=1}^{N} \sum_{f=1}^{F} \sum_{t=1}^{T} U_{k,f,t} X_{k,f,t} \tag{5.7}$$

s.t

$$\sum_{f=1}^{F} \sum_{t=1}^{T} X_{k,f,t} \geq 1; \forall k \in \{1, ..., N\} \tag{5.8}$$

$$X_{k,f,t} + X_{k',f,t} \leq 1; \forall k, k' \in N, k \neq k', \forall f, \forall t \tag{5.9}$$

$$\sum_{t=1}^{d_k} \sum_{f=1}^{F} X_{k,f,t} = P_k; \forall k \in N \tag{5.10}$$

$$X_{k,f,t} \in \{0, 1\}; \forall k \in N; \forall f \in F; \forall t \in T \tag{5.11}$$

In the problem formulation, the objective function in (5.7) maximizes the total throughput of all the nodes in TSCH network governed by the gateway. The constraint in (5.8) ensures that each node is assigned at least one time slot and hence provides temporal notion of fairness. If there is a need to ensure more bandwidth to a link, this can be done by putting more slots to that link which allows the need to send more packets inside the frame. The constraint in (5.9) is used to avoid collision, by guaranteeing that at most one user can transmit in a certain slot and frequency offset. Constraint (5.10) is used to ensure that each node has to transmit a certain number of packets within a deadline frame. Constraint in (5.11) indicates a binary decision variable.

We assume in the simulations part of this work that the buffers of the nodes are continuously backlogged; i.e., there are always enough packets to transmit with the data rate determined by the scheduling algorithm. This situation is necessary in order to effectively evaluate the performance of the scheduling process by avoiding the possible influence of the traffic arrival process.

Generalization of the Deadline Constraints

In our work, we assume that each packet has a deadline of one timeslot. Most work focuses on dropping a packet from the system if it is not delivered within the estimated deadline. Conversely, our aim is not to drop the unacknowledged packet that is not delivered within the estimated deadline, but to minimize the number and the entity of the delay. Moreover, in real-time applications, data arrive at the node's buffer on a frame-by-frame structure. This means that multiple packets (belonging to the same frame) arrive simultaneously rather than one at a time. A slot frame consists of a fixed number of packets, and each packet fits into exactly one time slot of duration t (both transmission and acknowledgement).

Each slot frame has its own deadline, and therefore, packets belonging to the same frame have the same deadline [80]. This deadline represents the maximum number of time slots by which packets belonging to the same slot frame need to be transmitted, on average. ***P1*** can be thought of a special case, where the deadline was equal to one time slot, and each slot frame consists of packets.

5.2.2 Proposed GA Scheduler

Genetic Algorithms

Our motivation for using GAs in designing suboptimal schedulers for the throughput scheduling problem is manifold. First, GAs are suitable for problems with large search spaces. They are equipped with many tools to reduce computational complexity and also to produce a diverse set of solutions. This is possible because they can quickly focus in a specific area and diversify search to develop wide range of solutions to address unknown environments. Considering that the solution space in the throughput scheduling problem is huge (even for 5 nodes, 5 frequencies, and 5 time slots, the size of the solution space for the throughput optimal scheduler is 2^{125}). Secondly, the binary decision variables $X_{k,f,t}$ can be easily encoded to a binary string, and therefore, GAs can be suitably applied.

Encoding

The feasibility of using genetic algorithm is problem dependent. Its success strongly depends on whether the right encoding scheme can be adopted or not. We detail all the adopted symbols in Table 5.1 below. We associate with each source node three integer values: a unique identifier (n), the number of packets that should be transmitted (t), and the deadline (δ) as described in (5.12).

$$(n, t, \delta) \qquad n \in \{1, 2, \ldots, N\} \qquad t, \delta \in \mathbb{N}. \tag{5.12}$$

In our work, we encoded the solutions as an integer vector having length equal to M. We represent our solution in (5.13) and (5.14), and we detail the encoding scheme in (5.15).

Table 5.1: *Symbols.*

Symbol	Description
S	a generic solution
s_i	the i-th element of the solution
N	the number of source devices
M	the total number of slots that should be transmitted

$$M = \sum_{i=1}^{N} t_i \qquad \forall i = 1, 2, \ldots.., N. \tag{5.13}$$

$$S \in \mathbb{N}^M = \{s_1, s_2, \ldots, s_m\}, \tag{5.14}$$

$$s_i \in \{1, 2, \ldots, N\} \qquad \forall i \in \{1, 2, \ldots, M\}. \tag{5.15}$$

Solutions must guarantee to each source node the transmission of all the required time slots in order to be feasible.

Fitness Function Evaluation

The amount to which a chromosome satisfies a problem requirements depends on how much it maximizes or minimizes the objective function and how many problem constraints it violates. In view of this, our goal is to maximize the throughput and also minimize the amount of violated deadline for all the transmissions. In maximizing the throughput, we introduce two values FT_a as maximizing the throughput and FT_b as the problem constraints it violates. If any constraint is violated, FT_a equals zero and FT_b is non-zero. Likewise, if no constraint is violated, FT_b equals zero and FT_a is non-zero. We formulate the fitness value for maximizing the throughput as follows:

$$FT_a = \begin{cases} 0 & \text{if } W_1 + W_2 > 0; \\ \sum_{k=1}^{N} \sum_{f=1}^{F} \sum_{t=1}^{T} U_{k,f,t} X_{k,f,t} & \text{otherwise;} \end{cases} \tag{5.16}$$

$$FT_b = \begin{cases} 0 & \text{if } W_1 + W_2 = 0; \\ \dfrac{1}{W_1 + W_2} & \text{otherwise;} \end{cases} \tag{5.17}$$

where W_1 is the number of violations of constraint (5.8), and W_2 is the number of violations of constraints (5.9). If $W_1 + W_2 > 0$, it means that some constraints are violated. If $W_1 + W_2 = 0$, it means that none of the constraints is violated. On the other hand, we minimize the current fitness function:

$$f = \sum_{i=1}^{N} \sum_{j=1}^{t_i} (\delta_i - r_{ij}) W_j(\delta_i, r_i) \tag{5.18}$$

where

$$W_{ij}(\delta, r) = \begin{cases} 1 & \text{if } \delta > r_{ij} \\ 0 & \text{otherwise} \end{cases} \tag{5.19}$$

and r_i is the allocation vector of each timeslot of the i-th mote. If we are not able to obtain the best schedule, we always prefer small time violations of each deadline.

Operator

The mutation operator alters a single solution and generates a new one; this operator is used to re-introduce genes that are lost during the recombination phase with a certain mutation probability; the aim of this operator is to avoid establishing a uniform population unable to evolve. In our case, we randomly permutate the solution. Moreover, since our goal is to find the best ordering, we rely on the PMX Crossover that performs some permutations among the two parents.

Search Strategy

NSGAII Non dominated Sorting Genetic Algorithm-II [28] is based on the concept of Pareto dominance and optimality and uses a fast non-dominating sorting approach in classifying the solutions. Using sorting in NSGA-II, i.e., rapid and non-dominated solution can reduce computational complexity. NSGA-II selection operator uses non-dominated rank and a crowding distance [66] value to choose a winner between two feasible individuals from the parent population.

NSGAIII Non dominated Sorting Genetic Algorithm-III [29], is an improved version of NSGA-II, able to efficiently evolve solutions where problems include a large number of objectives. Since we are dealing with a single objective problem, NSGA-III will pick a random solution for its recombination and mutation operators. The key behind the success of NSGA-III is the niching based selection operator, which adapts pressure automatically according to the dimensionality of the problem in hand. The major difference between NSGA-II and NSGA-III is replacing the crowding-distance-based niching [66] with reference-directions-based niching [29].

MOCell The multi-objective cellular genetic algorithm [57] uses a decentralized population structure called cellular whereby individuals are isolated by distance and only solutions in neighbouring cells are allowed to interact, which prevents good solutions from spreading quickly and hence maintains solution diversity.

SPEA2 Approach SPEA2 [85] uses an archive to store non-dominated solutions. SPEA2 maintains an external set of individuals that take into account the number of solutions for each solution it dominates and the number of solutions by which it is dominated. Individuals in population and external set are evaluated interdependently.

Round Robin Scheduler and Random Scheduler

In this section, we introduce another scheduler called round robin scheduler. The round robin scheduler considers all the source nodes and allocates a single slot to each until all the transmissions are performed. More into the details, the scheduler allocates the first $n-1$ timeslots to source nodes $s_1, s_2, ..., s_{n-1}$, then it performs another round allocating slots to source nodes having a second packet to transmit. The total number of rounds is equal to the maximum number of packets that should be transmitted among all the source nodes, and the procedure ends when all the packets are scheduled. Differently from a classical round robin, the scheduler assigns timeslots only to source nodes that should transmit at least one packet. This feature improves the global throughput of the system.

On the other hand, the random scheduler which is based on the round robin paradigm allocates all the timeslots randomly and it does not follow a precise strategy. Also in this case, the random scheduler assigns timeslots only to source nodes that should transmit at least one packet; source nodes that have already transmitted all their packets are not taken into account.

Performance Evaluation In this section, we implemented the GA-based suboptimal schedulers with the jmetal framework, whereas we solved the optimization problems using

CPLEX. In our simulation, we consider a TSCH network comprising of sensor nodes randomly deployed in a square area where a star topology is considered i.e., each device is randomly deployed around the gateway. We assume that a slotframe contains 75 timeslots and each slot is 15 ms long, and we obtained $U_{k,f,t}$ values for the slotframe period mentioned above. The simulation parameters are given in table 5.2. The simulation is performed with 50 runs and the results are given taking average of those 50 runs.

Table 5.2: *Basic Simulation Parameters.*

Parameters	Value
Radio band	2.4 GHz
Bit rate	250 kbps
Channel bandwidth B	2 MHz
Number of nodes N	[10,100]
Channel numbers	11-26
Timeslot duration t	15 ms
Slotframe duration T	750 ms
Deadline	d_k
Transmitted power P_{tx}	mW

In Section 5.2.2, we used the original parameters proposed for GAs, and we implemented the classical crossover and mutation operators for combinatorial problems. The chromosome size is equal to $N \times F \times T$. Various experiments were performed. We analyse the best solutions taking the two objectives separately. Employing a very large experiment directly in the first steps is usually considered a waste of time [56]. In line with this approach, we initially make a series of experiments using N = 10, and then observe the impact with various method combinations. Afterwards, we evaluate the scalability by increasing the number of nodes to 100 in steps of 10. The platform used to execute these experiments was an i7 CPU 860@2.80GHz x 8 equipped with 12GB RAM running Ubuntu OS 16.04 LTS.

One of the objective is to minimize the violated deadlines, so we propose GA-based algorithms to solve for the objective function. At first, we compare the execution times

Table 5.3: *Genetic Algorithm Parameters.*

Parameter	Value
Population size	100
Evaluations number	100
Crossover operator	one point
Crossover probability	0.9
Mutation operator	Random mutation
Mutation probability	$\frac{1}{\text{task number}}$
Selection operator	Binary tournament
Search heuristic	NSGA-II, NSGA-III, SPEA2, MOCell

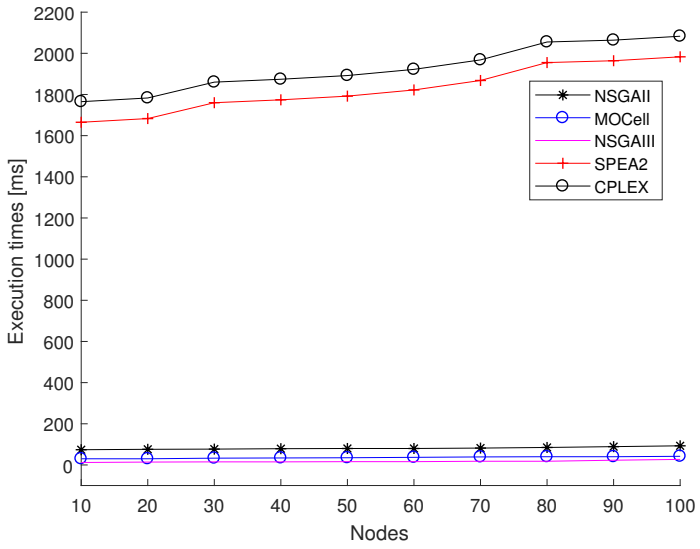

Figure 5.2: *Execution Times of the Scheduling Algorithm.*

Table 5.4: *Average Execution times in ms of the various Genetic Algorithm Search Strategies.*

Number of Nodes	NSGAII	NSGAIII	MOCell	SPEA2
10	74	12	30	1665
20	76	14	30	1683
30	77	15	33	1760
40	79	15	34	1774
50	80	16	35	1792
60	80	16	37	1822
70	82	18	39	1868
80	85	18	40	1955
90	89	23	40	1964
100	93	27	42	1983

of the search heuristic indicated in section IV-H. We use the genetic algorithm parameters indicated in Section 5.2.2. As shown in Figure 5.2, GAs solves the allocated problem faster than CPLEX. NSGA-III performs less compare to others in terms of the execution time. It performs faster because it focuses on multiple predefined reference points rather than multiple search directions as seen in other GAs. Each reference point can be emphasized to find the set of pareto-optimal points. The worst among the GAs is SPEA2 because it has a complexity of $\mathcal{O}(L \log L)$, where L is related to the population size and the pareto front instance. Table 5.4 gives a detailed values for the execution times of the search heuristics.

In Figure 5.3, with the parameter settings in Table 5.2, we compare the result of GA with CPLEX, round robin scheduler and random scheduler with respect to the objective of minimizing the violated deadlines. We implemented the GA using various search strategies presented in the previous sections, but they have the same result since we are considering a single-objective optimization problem. We indicate the search heuristics as GA. As shown in Figure 5.3, we can see that GA outperforms both round robin and

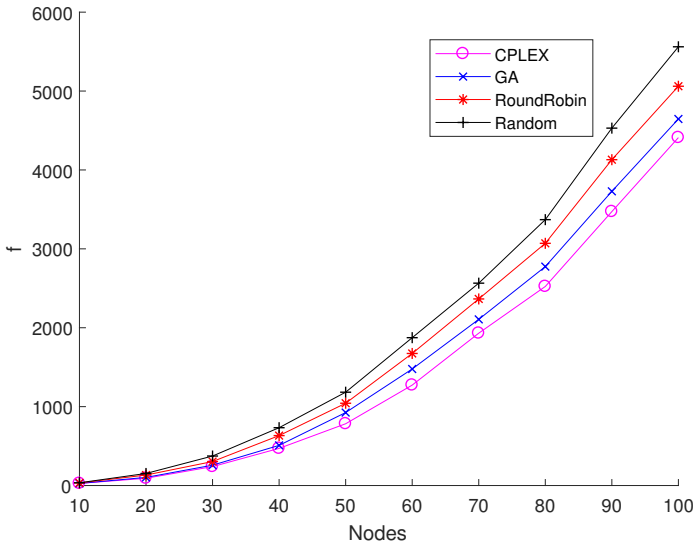

Figure 5.3: *Minimizing the violated deadlines.*

random schedulers, and at the same time yielding very close result to the optimal solution obtained through CPLEX. Moreover, we consider 10 nodes with mutation rate = 0.01, and a crossover to be 0.9. Figure 5.3 presents the average throughput optimal as well as the throughput maximizing GA-based scheduling scheme with the parameters in Section 5.2.2 while varying the population size. It can be shown that increasing the population size increases the throughput, however, the computational cost of a single iteration increases since more chromosomes have to be processed at each iteration. In addition, the performance improvement decreases as the population size increases. The results indicate that setting the population size to 100 is a sensible choice in terms of both criteria. Furthermore, the GA-based approach outperforms the round robin scheduler while at the same time yielding a close performance to the throughput optimal derived by CPLEX. Figure 5.4 presents the average network throughput performance for the throughput optimal, Round Robin and the GA-based suboptimal scheduler with population size of 100, mutation rate of 0.01 and crossover to be 0.9, where the number of TSCH nodes varies between 10 and 100. For all the schemes mentioned above, the GA-based suboptimal scheduler outperforms the round robin scheduler in terms of throughput with several orders of magnitude, while at the same time yielding a very close performance to the throughput optimal scheduler. The throughput increases as the number of nodes increases.

Notice here that if we ignore constraints (5.8), (5.10) and (5.11) in the problem formulation, the optimal solution is achieved when each frequency and timeslots are assigned to the node n_k that has the maximum $U_{k,f,t}$ value for the frequency and timeslot. In other words, $X_{k,f,t} = 1$ if $k = argmax_k(U_{k,f,t})$ and 0 otherwise, $\forall f \in F, \forall t \in T$.

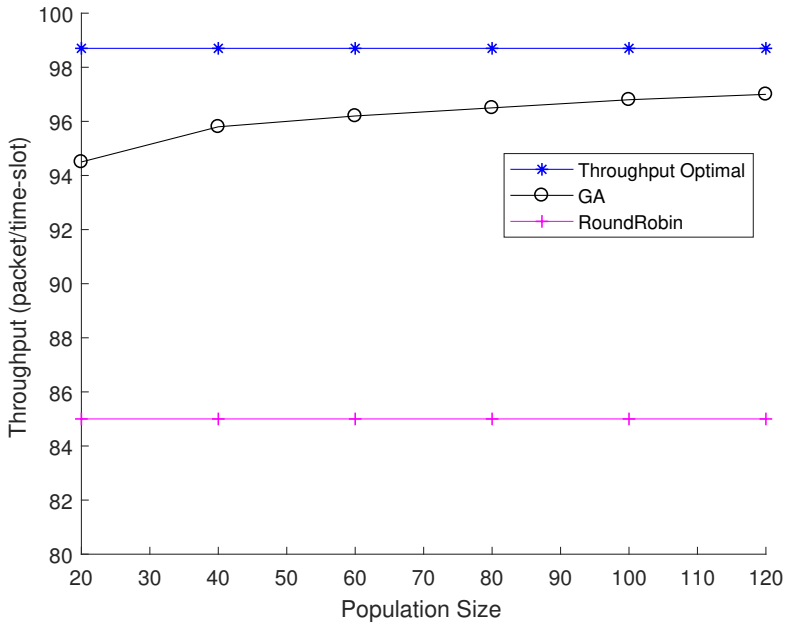

Figure 5.4: *Average network throughput for the GA-based scheduling scheme with $N = 10$, crossover = 0.9, mutation probability = 0.01 and varying population size.*

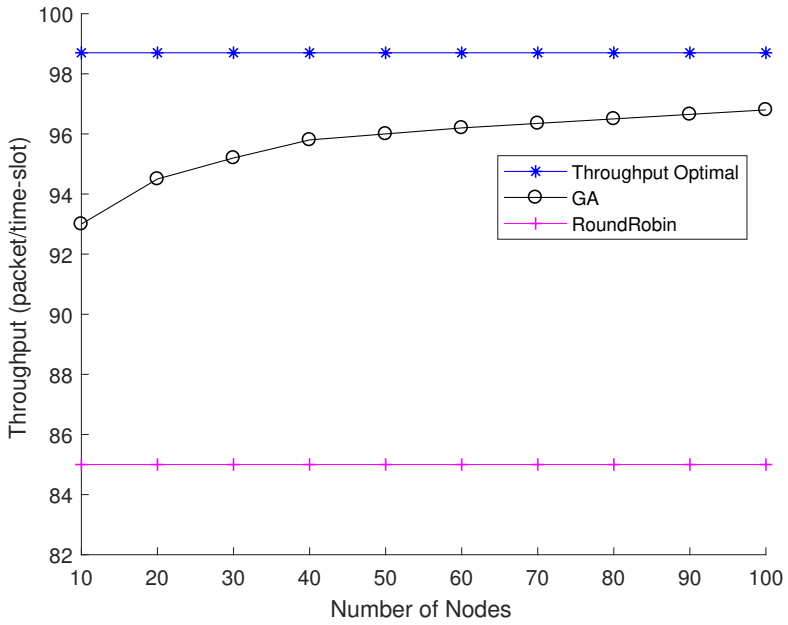

Figure 5.5: *Average network throughput for the GA-based scheduling scheme with population size = 100, crossover = 0.9, mutation probability = 0.01 and varying number of nodes.*

CHAPTER 6

Conclusions

This thesis proposed novel power-efficient resource allocation strategies in DCs. The highest expenses for Cloud providers concern electricity costs and hardware management, and the introduction of DC management strategies improving the resource utilization is useful both to mitigate electricity costs and to increase provider's RoI. We analyzed the allocation problem from two different perspectives: the former deals with the improvement of computing and network resource utilization defining new placement strategies for tasks and VMs; and the latter handles the internal management of network flows through the application of load balancing techniques, resilience mechanisms, and network demands scheduling.

For what concerns the first aspect, we distinguished the static allocation, where all the requests are known in advance, from the dynamic one, which considers each request one by one. More thoroughly, we formulated a multi-objective optimization problem for static task and VM placement designing a fast and scalable approach based on a customized implementation of MOGA. We aimed to solve a complex problem both minimizing server and switch power consumption while improving DC overall performance by drastically reducing the computation length; the allocation patterns found by MOGA were compared with others computed by SA and CPLEX. Results show that MOGA reaches the same performance of CPLEX reducing the computation time from hundreds to less than ten seconds. Moreover, MOGA performs better with respect to other heuristics such as multi-objective SA implemented to perform another comparison.

The main goal of dynamic VM allocators is to place as many VMs as possible limiting resource fragmentation and increasing resource availability for future allocations. We compared several allocators based on two classical allocation strategies: BF and WF. Among the allocators based on single-objective, multi-objective, and fuzzy logic

solvers, we reached slightly better results using the WF multi-objective allocator. Since WF typically fragments the available resources, we investigated on how to further improve it in order to overcome WF limitations. For this reason, we designed the Enhanced Worst Fit (EWF) strategy combining the strength point of both BF and WF. One of the key aspects of EWF is the reduction of resource fragmentation using the history of the previous requests. Indeed, EWF predicts future allocation patterns using statistical information collected from the previous requests. Results show that EWF places more VM with respect to BF and WF, hence satisfying a higher number of allocation requests.

We addressed the issues related to the design and implementation of a new controller for class-based traffic recovery and load balancing in SDN networks. We carried out functional and performance tests aimed at verifying first the correct operation of our controller in presence of multiple flows and link failures, and successively the performance of load balancing with different cost strategies. The results highlighted that the inter-working between the controller logic and the traffic monitoring/estimation module allow to take advantage of the network redundancy for better resource utilization and efficient recovery from failures.

The last improvement of DC network flow management concerns the design of a topology-aware scheduling strategy for MPI collective communications that avoids network congestions and minimizes the overall communication time. The proposed scheduler operates on a graph representing the whole network which is functionally partitioned to increase the communication parallelism while decreasing the communication latency. We tested our scheduler on several topologies, and the results are promising since we achieved the optimality in terms of communication latency when servers are organized according to a tree topology, and we reached a good parallelism degree for servers interconnected through mesh and line topologies.

In the last part of the thesis, we pointed out our contributions to the development of power-efficient communication strategies for IoT devices. We designed a new scheduling strategy based on GA for IEEE 802.15.4 devices using a centralized gateway and TSCH. We considered a real-time scenario in which communications should be transmitted within a deadline while reducing the device power-consumption and the overall communication latency.

Our contribution aimed, on the one hand, at the improvement of QoS and QoE for 5G customers, on the other at the increment of providers and mobile operators revenues. The joint optimization of the power consumed by DCs and the improvement of resources utilization allow a reduction of the operational costs while increasing the number of customers. In addition, the integration of Cloud functionality with the mobile network will be challenging in the next years: it will be mandatory to implement an orchestrator both in DC and IoT scenarios in order to meet the very strict QoS requirements of the 5G standard.

Bibliography

[1] Cisco Data Center Spine-and-Leaf Architecture: Design Overview White Paper - white-paper-c11-737022.pdf.

[2] Iperf. http://iperf.fr/.

[3] Memory management reference. http://www.memorymanagement.org/.

[4] Mininet. http://mininet.org/.

[5] Pox. http://www.noxrepo.org/pox/about-pox/.

[6] Pox, openflow.discovery. https://github.com/noxrepo/pox/blob/betta/pox/openflow/discovery.py.

[7] IEEE Standard for Low-Rate Wireless Networks. *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, April 2016.

[8] Draft new report itu-r m.[imt-2020.tech perf req] - minimum requirements related to technical performance for imt-2020 radio interface(s). https://www.itu.int/md/R15-SG05-C-0040/en, 2017.

[9] Nicola Accettura, Maria Rita Palattella, Gennaro Boggia, Luigi Alfredo Grieco, and Mischa Dohler. Decentralized Traffic Aware Scheduling for Multi-Hop Low Power Lossy Networks in the Internet of Things. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops*, pages 1–6. IEEE, 2013.

[10] D. Adami, A. Gabbrielli, S. Giordano, M. Pagano, and G. Portaluri. A fuzzy logic approach for resources allocation in cloud data center. In *2015 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6, Dec 2015.

[11] D. Adami, S. Giordano, M. Pagano, and G. Portaluri. A novel sdn controller for traffic recovery and load balancing in data centers. In *2016 IEEE 21st International Workshop on Computer Aided Modelling and Design of Communication Links and Networks (CAMAD)*, pages 77–82, Oct 2016.

[12] Davide Adami, Stefano Giordano, Michele Pagano, and Nicola Santinelli. Class-based traffic recovery with load balancing in software-defined networks. In *GLOBECOM Workshops*, pages 161–165. IEEE, 2014.

[13] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 63–74, New York, NY, USA, 2008. ACM.

[14] Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. Loggp: Incorporating long messages into the logp model — one step closer towards a realistic model for parallel computation. Technical report, Santa Barbara, CA, USA, 1995.

[15] M. Alicherry and T. V. Lakshman. Network aware resource allocation in distributed clouds. In *INFOCOM, 2012 Proceedings IEEE*, pages 963–971, March 2012.

[16] Silvio Roberto Martins Amarante, Filipe Maciel Roberto, Andre Ribeiro Cardoso, and Joaquim Celestino. Using the Multiple Knapsack Problem to Model the Problem of Virtual Machine Allocation in Cloud Computing. pages 476–483. IEEE, December 2013.

[17] P. Aruna and S. Vasantha. A particle swarm optimization algorithm for power-aware virtual machine allocation. In *2015 6th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–6, July 2015.

[18] Shapour Azarm and Jin Wu. Metrics for Quality Assessment of a Multiobjective Design Optimization Solution Set. *Journal of Mechanical Design*, 123(1):18–25, 2001.

[19] Luis Andre Barroso, Jimmy Clidaras, and Urs Hoelzle. *The Datacenter as a Computer:An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, 2013.

[20] Anton Beloglazov and Rajkumar Buyya. Energy efficient allocation of virtual machines in cloud data centers. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 577–578, Washington, DC, USA, 2010. IEEE Computer Society.

[21] Dejene Boru, Dzmitry Kliazovich, Fabrizio Granelli, Pascal Bouvry, and AlbertY. Zomaya. Energy-efficient data replication in cloud computing datacenters. *Cluster Computing*, 18(1):385–402, 2015.

[22] Deji Chen, Mark Nixon, Song Han, Aloysius K Mok, and Xiuming Zhu. Wirelesshart and IEEE 802.15. 4e. In *Industrial Technology (ICIT), 2014 IEEE International Conference on*, pages 760–765. IEEE, 2014.

[23] Y. C. Chou, S. S. Nestinger, and H. H. Cheng. Ch mpi: Interpretive parallel computing in c. *Computing in Science Engineering*, 12(2):54–67, March 2010.

[24] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca. The dynamic placement of virtual network functions. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–9, May 2014.

[25] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[26] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. Logp: Towards a realistic model of parallel computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPOPP '93, pages 1–12, New York, NY, USA, 1993. ACM.

[27] R. Cziva, S. Jouët, D. Stapleton, F. P. Tso, and D. P. Pezaros. Sdn-based virtual machine management for cloud data centers. *IEEE Transactions on Network and Service Management*, 13(2):212–225, June 2016.

[28] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, Apr 2002.

[29] Kalyanmoy Deb and Himanshu Jain. An Evolutionary Many-Objective Optimization Algorithm using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems with Box Constraints. *IEEE Trans. Evolutionary Computation*, 18(4):577–601, 2014.

[30] J.L. Díaz, J. Entrialgo, M. García, J. García, and D.F. García. Optimal allocation of virtual machines in multi-cloud environments with reserved and on-demand pricing. *Future Generation Computer Systems*, 71:129–144, 2017.

[31] Vassilios V. Dimakopoulos and Nikitas J. Dimopoulos. Communications in binary fat trees. In *In IEEE ICDCS*, 1995.

[32] Juan J. Durillo and Antonio J. Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760 – 771, 2011.

[33] Sinem Coleri Ergen and Pravin Varaiya. TDMA Scheduling Algorithms for Wireless Sensor Networks. *Wireless Networks*, 16(4):985–997, 2010.

[34] Weiwei Fang, Xiangmin Liang, Shengxin Li, Luca Chiaraviglio, and Naixue Xiong. Vmplanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers. *Computer Networks*, 57(1):179 – 196, 2013.

[35] Robert Fuller. *Introduction to Neuro-Fuzzy Systems*. Advances in Soft Computing Series. Physica-Verlag Heidelberg, 2000.

[36] C. Ghribi, M. Hadji, and D. Zeghlache. Energy efficient vm scheduling for cloud data centers: Exact allocation and migration algorithms. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 671–678, May 2013.

[37] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[38] Didem Gözüpek and Fatih Alagöz. Genetic Algorithm-Based Scheduling in Cognitive Radio Networks under Interference Temperature Constraints. *International Journal of Communication Systems*, 24(2):239–257, 2011.

[39] T. Guérout, Y. Gaoua, C. Artigues, G. Da Costa, P. Lopez, and T. Monteil. Mixed integer linear programming for quality of service optimization in clouds. *Future Generation Computer Systems*, 71:1–17, 2017.

[40] Piyush Gupta and Panganmala R Kumar. The Capacity of Wireless Networks. *Information Theory, IEEE Transactions on*, 46(2):388–404, 2000.

[41] Ibrahim Habib, Mahmoud Sherif, Mahmoud Naghshineh, and Parviz Kermani. An Adaptive Quality of Service Channel Borrowing Algorithm for Cellular Networks. *International Journal of Communication Systems*, 16(8):759–777, 2003.

[42] Dimitris Hatzopoulos, Iordanis Koutsopoulos, George Koutitas, and Ward Van Heddeghem. Dynamic virtual machine allocation in cloud server facility systems with renewable energy sources. In *Communications (ICC), 2013 IEEE International Conference on*, pages 4217–4221. IEEE, 2013.

[43] Jerry L. Hintze and Ray D. Nelson. Violin plots: A box plot-density trace synergism. *The American Statistician*, 52(2):181–184, May 1998.

[44] ILOG. Efficient Modeling in ILOG OPL-CPLEX Development System. *White Paper*, June 2007.

[45] ISA-100.11a-2011. Wireless Systems for Industrial Automation: Process Control and Related Applications. *ISA.100.11a-2011*, May 2011.

[46] H. Ishibuchi, Naoya Akedo, H. Ohyanagi, and Yusuke Nojima. Behavior of emo algorithms on many-objective optimization problems with correlated objectives. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 1465–1472, June 2011.

[47] LI Jielin and Ming Chen. Multi-Objective Topology Optimization Based on Mapping Matrix and NSGA-II for Switched Industrial Internet of Things. *IEEE Internet of Things Journal*, 2016.

[48] D. Kliazovich, P. Bouvry, Y. Audzevich, and S. U. Khan. Greencloud: A packet-level simulator of energy-aware cloud computing data centers. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–5, Dec 2010.

[49] A Lara, A Kolasani, and B. Ramamurthy. Network innovation using OpenFlow: A survey. *Communications Surveys Tutorials, IEEE*, 16(1):493–512, First 2014.

[50] C. C. Lee. Fuzzy logic in control systems: fuzzy logic controller. i. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(2):404–418, Mar 1990.

[51] C. C. Lee. Fuzzy logic in control systems: fuzzy logic controller. ii. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(2):419–435, Mar 1990.

[52] Junchao Ma, Wei Lou, Yanwei Wu, X-Y Li, and Guihai Chen. Energy Efficient TDMA Sleep Scheduling in Wireless Sensor Networks. In *INFOCOM 2009, IEEE*, pages 630–638. IEEE, 2009.

[53] B. Martini, D. Adami, M. Gharbaoui, P. Castoldi, L. Donatini, and S. Giordano. Design and evaluation of sdn-based orchestration system for cloud data centers. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6, May 2016.

[54] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, 2008.

[55] D. Minarolli and B. Freisleben. Virtual machine resource allocation in cloud computing via multi-agent fuzzy control. In *Cloud and Green Computing (CGC), 2013 Third International Conference on*, pages 188–194, Sept 2013.

[56] Douglas C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, 2006.

[57] Antonio J Nebro, Juan J Durillo, Francisco Luna, Bernabé Dorronsoro, and Enrique Alba. Mocell: A Cellular Genetic Algorithm for Multiobjective Optimization. *International Journal of Intelligent Systems*, 24(7):726–746, 2009.

[58] Sergio Nesmachnow, Bernabé Dorronsoro, JohnatanE. Pecero, and Pascal Bouvry. Energy-aware scheduling on multicore heterogeneous grid computing systems. *Journal of Grid Computing*, 11(4):653–680, 2013.

[59] Quyet Thang Nguyen, Nguyen Quang-Hung, Nguyen Huynh Tuong, Van Hoai Tran, and Nam Thoai. Virtual machine allocation in cloud computing for minimizing total execution time on each machine. In *Computing, Management and Telecommunications (ComManTel), 2013 International Conference on*, pages 241–245, Jan 2013.

[60] B.AA Nunes, M. Mendonca, Xuan-Nam Nguyen, K. Obraczka, and T. Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *Communications Surveys Tutorials, IEEE*, 16(3):1617–1634, Third 2014.

[61] ONF. Software-defined networking: The new norm for networks. Technical report, Open Networking Foundation, April 2012.

[62] ONF. OpenFlow switch specification. Technical report, Open Networking Foundation, October 2013.

[63] Maria Rita Palattella, Nicola Accettura, Mischa Dohler, Luigi Alfredo Grieco, and Gennaro Boggia. Traffic Aware Scheduling Algorithm for Reliable Low-Power Multi-Hop IEEE 802.15.4e Networks. In *Personal Indoor and Mobile Radio Communications (PIMRC), 2012 IEEE 23rd International Symposium on*, pages 327–332. IEEE, 2012.

[64] K Pister and Lance Doherty. TSMP: Time Synchronized Mesh Protocol. *IASTED Distributed Sensor Networks*, pages 391–398, 2008.

[65] G. Portaluri, S. Giordano, D. Kliazovich, and B. Dorronsoro. A power efficient genetic algorithm for resource allocation in cloud computing data centers. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 58–63, Oct 2014.

[66] Carlo R. Raquel and Prospero C. Naval, Jr. An effective use of crowding distance in multiobjective particle swarm optimization. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, GECCO '05, pages 257–264, New York, NY, USA, 2005. ACM.

[67] P. Ruiu, A. Bianco, C. Fiandrino, P. Giaccone, and D. Kliazovich. Power comparison of cloud data center architectures. In *IEEE ICC 2016 - Communication QoS, Reliability and Modeling Symposium*, 2016.

[68] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester. Enabling fast failure recovery in openflow networks. In *2011 8th International Workshop on the Design of Reliable Communication Networks (DRCN)*, pages 164–171, Oct 2011.

[69] B. Suman and P. Kumar. A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society*, 57(10):1143–1160, Oct 2006.

[70] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(1):116–132, Jan 1985.

[71] J.P. Vasseur, M. Pickavet, and P. Demeester. *Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS*. Morgan Kaufmann series in networking. Morgan Kaufmann, 2004.

[72] Abhishek Vichare, Zenia P. Gomes, Noella Fernandes, and Flavin Cardoza. Cloud computing using OCRP and virtual machines for dynamic allocation of resources. In *Technologies for sustainable development (ICTSD), 2015 International Conference on*, pages 1–5. IEEE, 2015.

[73] X Vilajosana and K Pister. Minimal 6TiSCH Configuration-draft-ietf-6tisch-minimal-00. *IETF: Fremont, CA, USA*, 2013.

[74] J. V. Wang, C. T. Cheng, and C. K. Tse. A power and thermal-aware virtual machine allocation mechanism for cloud data centers. In *2015 IEEE International Conference on Communication Workshop (ICCW)*, pages 2850–2855, June 2015.

[75] Lizhe Wang, Gregor von Laszewski, Jay Dayal, and Fugang Wang. Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with dvfs. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 368–377, Washington, DC, USA, 2010. IEEE Computer Society.

[76] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: statistical analysis of ethernet lan traffic at the source level. *IEEE/ACM Transactions on Networking*, 5(1):71–86, Feb 1997.

[77] Z. Xiao, W. Song, and Q. Chen. Dynamic resource allocation using virtual machines for cloud computing environment. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1107–1117, June 2013.

[78] J. Xu and J. A. B. Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 179–188, Dec 2010.

[79] Chao-Tung Yang, Kuan-Chieh Wang, Hsiang-Yao Cheng, Cheng-Ta Kuo, and William Cheng C. Chu. Green Power Management with Dynamic Resource Allocation for Cloud Virtual Machines. pages 726–733. IEEE, September 2011.

[80] Lei Yang, Yalin Evren Sagduyu, and Jason Hongjun Li. Adaptive Network Coding for Scheduling Real-time Traffic with Hard Deadlines. In *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*, pages 105–114. ACM, 2012.

[81] Y. Yu. Mobile edge computing towards 5g: Vision, recent progress, and open challenges. *China Communications*, 13(Supplement2):89–99, N 2016.

[82] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338 – 353, 1965.

[83] Yaqiang Zhang, Zhangbing Zhou, Deng Zhao, Mahmoud Barhamgi, and Taj Rahman. Graph-Based Mechanism for Scheduling Mobile Sensors in Time-Sensitive WSNs Applications. *IEEE Access*, 5:1559–1569, 2017.

[84] E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca, and V.G. da Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *Evolutionary Computation, IEEE Transactions on*, 7(2):117–132, April 2003.

[85] Eckart Zitzler, Marco Laumanns, Lothar Thiele, et al. SPEA2: Improving the Strength Pareto Evolutionary Algorithm, 2001.