Pós-Graduação em Ciência da Computação

Leylane Graziele Ferreira da Silva

Utilizing Optimization Algorithms to Maximize the Availability of Composable Data Center

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2020

Leylane Graziele Ferreira da Silva

Utilizing Optimization Algorithms to Maximize the Availability of Composable Data Center

A M.Sc. Dissertation presented to the Center for Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Concentration Area: Computer Network
Advisor: Prof. Dr. Djamel Fawzi Hadj Sadok
Co-advisor: Prof. Dr. Patricia Takako Endo

Recife
2020

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

S586u Silva, Leylane Graziele Ferreira da
Utilizing optimization algorithms to maximize the availibility of composable data center / Leylane Graziele Ferreira da Silva. – 2020.
85 f.: il., fig., tab.

Orientador: Djamel Fawzi Hadj Sadok.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2020.
Inclui referências.

1. Redes de computadores. 2. Algoritmos de otimização. I. Sadok, Djamel Fawzi Hadj (orientador). II. Título.

004.6 CDD (23. ed.) UFPE - CCEN 2020 - 129

Leylane Graziele Ferreira da Silva

“Utilizing Optimization Algorithms to Maximize the Availability of Composable Data Center”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação

Aprovado em: 17/02/2020.

BANCA EXAMINADORA

_____

Prof. Dr. Aluizio Fausto Ribeiro Araújo
Centro de Informática / UFPE

_____

Prof. Dr. Carmelo José Albanez Bastos Filho
Escola Politécnica/UPE

_____

Prof. Dr. Djamel Fawzi Hadj Sadok
Centro de Informática / UFPE
(**Orientador**)

I dedicate this work to parents, for all suport and love.

ACKNOWLEDGEMENTS

I want to thank God for the opportunity to do this work and for all overcome challenges. To my parents, Graciete and Ligerson, and close relatives for the motivational support and their love. I love you, guys.

I want to thank my boyfriend, Matheus, for his patience, support, and love. You are an essential part of my life <3

In the same away, to my old and new friends to become these two years more fun. Especially to Guto, Edoarda, Demis, Élisson, Pedro Henrique, Iago, Arthur Flôr, Daniel Bezerra, Carolina Cani, and Diego.

To my advisor, Djamel Sadok, and Judith Kelner, for the opportunity to do this work and to make part of GPRT. I am forever grateful.

To my co-adviser, Patricia Endo, for the opportunity to work with her one more time. For all advice, motivation, friendship, and believing in me. You have a special place in my heart.

Finally, I would like to thank the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior Brasil (CAPES) - Finance Code 001 for funding this work.

"If it doesn't challenge you, it won't change you".

Fred Devito (DAVIS; BAKER; SPAJIĆ, 2015).

ABSTRACT

The cloud computing paradigm has performed, for years, the fundamental role in delivering IT resources, typically available in data centers, allowing cost reduction and providing services such as high availability, scalability, and elasticity. Despite many advantages, the data center infrastructure suffers from inefficiency problems due to factors such as excessive redundancy usage and infrastructure sub-utilization. The Composable Data Center paradigm aims to mitigate such problems, proposing the disaggregation of resources distributed in racks with different chassis configurations. In this context, different resource arrangements, allocated via software (called Composable Infrastructure) may directly affect the system availability. Thus, this work presents an optimization problem to allocate Composable Infrastructures in Composable Data Center, taking into account budget constraints, obeying application requirements to maximizing availability in those data centers. For such, some optimization algorithms utilized in two approaches: mono-objective approach and multi-objective approach. From the results, it is possible to identify the best configurations and understand how each component affects availability. In the mono-objective approach, the Dynamic Programming algorithm obtained the best results in balancing cost and availability. In the multi-objective approach, both GDE3 and NSGA-II algorithms were useful in finding attractive solutions, and GDE3 presented the most solution number in most of the cases.

Keywords: Composable Data Center. Composable Infrastructure. Data Center Availability. Resource Allocation. Optimization Algorithms.

RESUMO

O paradigma de computação em nuvem tem desempenhado, durante anos, um papel fundamental na entrega de recursos de tecnologia da informação (TI), normalmente disponíveis em *data centers*, permitindo redução de custos e provendo serviços como alta disponibilidade, escalabilidade e elasticidade. Apesar das inúmeras vantagens, as infraestruturas de data centers sofrem com alguns problemas de ineficiência devido a fatores como o uso excessivo de redundância e a sub-utilização da infraestrutura. O paradigma de *Composable data center* é uma solução idealizada de modo a mitigar tais problemas, o qual propõe a desagregação de recursos computacionais, distribuídos em *racks* com diferentes configurações de chassi. Neste contexto, as diferentes combinações de recursos, alocadas via *software* (denominadas Composable Infrastructure), podem afetar diretamente na disponibilidade do sistema. Deste modo, este trabalho apresenta um problema de otimização para realizar alocação de *Composable Infrastrucutures* em *Composable Data Centers* levando em consideração restrições orçamentárias, de modo a atender requisitos da aplicação para maximizar a disponibilidade nestes Data Centers. Para tal, foram utilizados diferentes algoritmos de otimização em duas principais abordagens: mono-objetivo e multi-objetivo. A partir dos resultados é possível identificar as melhores configurações e como cada recurso pode impactar na disponibilidade. Na abordagem mono-objetivo, o algoritmos *Dynamic Programming* apresentou os melhores resultados ao equilibrar disponibilidade e custo. Na abordagem multi-objetiva, os algoritmos GDE3 e NSGA-II foram efetivos para encontrar boas soluções e o algoritmo GDE3 apresentou uma maior quantidade de soluções na maioria dos casos.

Palavras-chaves: *Composable Data Center*. *Composable Infrastructure*. Disponibilidade de *Data Center*. Alocação de Recursos. Algoritmos de Otimização.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

IT	Information Technology
GA	Genetic Algorithms
ILP	Integer Linear Programming
MILP	Mixed-Integer Linear Programming
VNFs	Virtualized Network Functions
CD	Crowding Distance
CI	Composable Infrastructure
MTTF	Mean Time to Failure
MTTR	Mean Time to Repair
MOEAs	Multi-objective Evolutionary Algorithms
EAs	Evolutionary Algorithms
RBD	Reliability Block Diagrams
DP	Dynamic Programming
DE	Differential Evolution
PSO	Particle Swarm Optimization
VMs	Virtual Machines
PDSO	Parallel Discrete Particle Swarm Optimization
CAPEX	Capital Expenditure
POD	Performance Optimized Data Center
CAHCM	Cloud Memory Model
2DA	Dynamic Data Allocation Advances
NSGA-II	Non-dominated Sorting Genetic Algorithm
GDE3	Generalized Differential Evolution
vPOD	Virtual Performance Optimized Data Center

DMTF	Distributed Management Task Force
Intel RSD	Intel Rack Scale Design
IDC	International Data Corporation
SBX	Simulated Binary Crossover
HV	Hypervolume

CONTENTS

1 INTRODUCTION

Cloud computing has been accepted as a dominant computing paradigm in enterprise Information Technology (IT). Over 73% of organizations have at least one application or a portion of their computing infrastructure already hosted in the cloud (IDG, 2018). By and large, these enterprises are attracted by the convergence of IT efficiencies and business agility, enabled by scalability, rapid deployment, and high parallelization (KIM, 2009).

> *The fast adoption of cloud computing can be mostly attributed to its ability to quickly deploy applications with no upfront capital investment and to scale up or down the infrastructure capacity based on the real time workload requirement* (LIN et al., 2018).

But despite the promise of cloud computing, existing enterprise data center infrastructure often is subject to technology-driven inefficiencies. Recent research suggests that a lack of automation, overprovisioning for redundancy and resiliency, and infrastructure underutilization are resulting in significant people, process, and technology inefficiencies (NADKAMI, 2017). IDC (NADKAMI, 2017) suggests that in most enterprise data centers, infrastructure is 45% provisioned – which means infrastructure available, but not necessarily utilized – 45% utilized, 30% agile and 40% compliant with stated service level agreements (SLA) for infrastructure and its components, that should be established by cloud providers and their costumers. Much of this inefficiency can be explained by enterprise use of resource redundancy to improve availability (DUMITRESCU et al., 2018). However it is clear that this not only increases costs and management complexity but may also introduce new potential points of failure and vulnerabilities (SHARMA et al., 2016; WAGNER; SOOD, 2016).

As more and more enterprises embrace digital transformation, they are migrating their workloads to the cloud, requiring high availability moving as close as possible to 24/7 uptime (GUNAWI et al., 2016). However, they are faced with a new challenge caused by the increasing duality of legacy applications, on the one hand, and next generation cloud-native applications, on the other. The former requires infrastructure resiliency and exploits virtualization and clustering for portability and application state preservation, while the latter is designed to be horizontally scalable, containerized, and continuously updated (NADKAMI, 2017). Traditional enterprise data centers are not designed to accommodate the differing infrastructure requirements of both legacy and next generation applications.

To achieve improved IT staff productivity, increased utilization of compute and storage resources, faster provisioning, higher availability and greater business agility, enterprises are increasingly looking to a new data center paradigm called **disaggregated or composable data center infrastructure** (IDC, 2016). Composable data center infras-

tructure disaggregates and refactors compute, storage, network and other infrastructure resources into shared resource pools that can be dynamically and automatically assembled and re-assembled to meet changing workload requirements (NADKAMI, 2017; KUMBHARE et al., 2016). The design principles for such composable infrastructure is fundamentally different from that of traditional data centers as the hardware underlying the data center must support full or partial dynamic disaggregation while some control software must be available to logically assemble the hardware required by the application from the resource pool (NADKAMI, 2017; KUMBHARE et al., 2016). In this way, not only are inefficiency issues mitigated, but application duality is addressed by being able to deploy heterogeneous resources that support specific application requirements. Composable data center infrastructure "*offers the potential advantage of enabling continuous peak workload performance while minimizing resource fragmentation for fast evolving heterogeneous workloads.*" (LI et al., 2017).

> *The vision of disaggregation is to depart from the traditional paradigm of the mainboard-as-a-unit (server-centric model) and to enable the creation of function block-as-a-unit (resource-centric model) having a baseline disaggregated pool of components including compute, memory, storage, network, and accelerators* (ZERVAS et al., 2018).

The forcast for composable infrastructure is expected to reach $4,7 billion in 2023 (NADKARNI; SHEPPARD; STOLARKSI, 2018). Given the economic opportunity and the potential benefits, it is not surprising that a number of the major global IT vendors are active in this area. Thus, to turn the composable data center a concrete and applicable paradigm, several industry initiatives have emerged. For instance, Intel proposed an architecture called Intel Rack Scale Design (Intel RSD), based on Redfish a Distributed Management Task Force (DMTF) industry standard. Next other companies started to offer their own compatible solutions. Ericsson proposed the Hyperscale Datacenter 8000 that provides configuration and control of a composable data center by using performance-optimized resources (named Virtual Performance-Optimized Data Center - vPOD).

Furthermore, composable data center infrastructure brings new challenges for data center vendors. Examples of the challenges present in this context are mentioned by (ZERVAS et al., 2017): latency overhead minimization; network system-specific performance and services according to communication type between components; high bandwidth and bandwidth density associated to low cost and power consumption; maximum flexibility provisioning; and maximization of resource utilization delivering workload performance at minimum cost through orchestration of components. As a result, this area is seen as very attractive for developing different studies on various aspects.

1.1 MOTIVATION

In the data center planning context, there are three main limitations mentioned by (KATRINIS et al., 2016): the first limitation is about **resource proportionality of a system**, which follows the proportionality of the basic mainboard, considering only initial procurement time and upgrade cycles and excluding the future need of doubling the component capacity; the second limitation is related to the **allocation of resources to processes or virtual machines (VMs)** due to resources being available within mainboard boundary which consequently faces fragmentation and inefficiency; finally, the third limitation is about the need of **technology upgrades** in all server boards, even if the upgrade is directed into a specific component.

All these problems may be addressed utilizing composable infrastructure. As discussed in the literature, VMs can be allocated utilizing computing resources that meet their needs, reducing capital expenditure (CAPEX) and, in the same way, involving fewer resources to achieve a demanding set. It explains that "*disaggregation brings modularity to systems, enabling easier hardware upgrades when desired.*" (PAGÈS et al., 2017), the authors concluding that these facts show that composable data centers infrastructure is a brilliant technology for data center future.

Among the characteristics that may turn a composable infrastructure feasible, there is the ability to aggregate the resources to form a temporal system (AJIBOLA; EL-GORASHI; ELMIRGHANI, 2018). Starting with this flexibility provided by resource aggregation, it is possible to build several different arrangements of a composable infrastructure in order to delivery the best indicators to the costumers. An important fact to note is that the distribution, configuration and arrangement of hardware resources may impact the availability of a composable infrastructure at different levels (KUMBHARE et al., 2016).

As mentioned in (ROOZBEH et al., 2018), high availability can be provided on a component level utilizing composable infrastructure. Redundancy may be introduced on logical servers to take over from failed components, which means that a failure of a single component on a logical server does not result in the failure of the whole system, and this logical server can continue to operate normally. Authors also argue a logical server that operates on a composable infrastructure handles CPU, memory, storage, or network failures without incurring an extended downtime, hence reducing application complexity.

The analysis of components applied to the availability context on composable data center infrastructure is the focus of this work. Thus, we have the following research problem "how to maximize the availability for allocating a composable infrastructure within a composable data center infrastructure while meeting the minimum requirements in terms of compute, memory, network and storage under budget constraints?". In the next section, we define the general and specific objectives of this work to answer the research question.

1.2 OBJECTIVES

Considering the motivation presented in the previous section, the main objective of this work is to propose optimization models to achieve availability maximization in composable a data center infrastructure, while meeting several cost constraints. The proposed models should meet application requirements to achieve workload execution and satisfy budget constraints. The specific goals of this dissertation are:

- Establish the availability estimate method, comprising the complexity of the system.
- Propose optimization problems modeleing using different approaches: mono-objective and multi-objective.
- Analyse and compare algorithms results with each other in respective approaches.
- Discuss the results obtained with the experiments, comparing the performance of both mono-objective and multi-objective solutions.
- Understand how hardware components may effect the availability of composable Infrastructure.

1.3 ORGANIZATION OF THE DISSERTATION

The reminder of this document is organized as follow:

Chapter 2 presents basic concepts of composable data centers. This chapter also presents concepts related to reliability block diagrams and optimization algorithms, exploring Dynamic Programming, Differential Evolution, Particle Swarm Optimization, Non-dominated Sorting Genetic Algorithm, and Generalized Differential Evolution operation. In chapter we also present in 2.4 the main related work found in literature and we perform a comparison with this dissertation.

Chapter 3 presents the formulation of the problem with a mono-objective view, defining its parameters and constraints. This chapter also shows the automatic availability model formulation based on the RBD approach and the implementation of optimization algorithms fitness functions applied in this approach. Additionally, we present the algorithms´ parametrization for a mono-objective approach, the scenarios definition used in this work, and specifies the experiments. Furthermore, this chapter presents component importance analysis considered in this work and shows the main results obtained though experiments applied to mono-objective approach.

Chapter 4 presents the formulation of the problem with a multi-objective view, defining its parameters and constraints. This chapter also shows the implementation of optimization algorithms fitness functions applied in this approach. Additionally,we present the algorithms parametrization for the multi-objective approach, and also presents the main results obtained through experiments, as well as, algorithms convergence analysis.

Finally, Chapter 5 presents an overview of this work and the main contributions, presenting its limitation, future works, the contributions of this work and the main publication related with this dissertation.

2 BACKGROUND

In this chapter, we will present the main concepts addressed in this work to clarify the context used in this research. The sections of this chapter are divided in: The composable data center (Section 2.1), Reliability Block Diagrams (Section 2.2) and Optimization Algorithms (Section 2.3).

2.1 THE COMPOSABLE DATA CENTER

Historically, cloud data centers were characterised by hardware and system software platform homogeneity, a common systems management layer, a greater degree of proprietary software use, single organisation control, and a focus on cost efficiency (LYNN, 2018). For multi-tenant hyperscale clouds, the clouds, per se, appear on top of the physical data centre infrastructure and are abstracted from end-user applications, end users, and software developers exploiting the cloud. In addition, they may operate across multiple physical data centres typically organised by geographic region (LYNN, 2018). This provides cloud service providers with cost efficiencies and deployment flexibility by allowing them to maintain, enhance, and expand the underlying cloud infrastructure without requiring changes to software (CRAGO; WALTERS, 2015). As such, infrastructure performance was typically improved through a combination of scale-out and advances in microprocessor capability, while service availability is assured through over-provisioning (LYNN, 2018). Despite being effective, this resulted in massive high-density data centers comprising hundreds if not thousands of servers, a significant amount of which may be under-utilised relatively to their peak load capability, with frequent idle times resulting in disproportionate energy consumption (BARROSO; HÖLZLE, 2007; UCHECHUKWU; LI; SHEN, 2014).

While traditional cloud architectures was based on hardware and software homogeneity, in recent times the so-called heterogeneous cloud has emerged. Hardware heterogeneity assumes the use of different or dissimilar hardware resources that may have specialized processing capabilities to handle specific tasks (SHAN, 2006). This may include specialized co-processors or networking infrastructure that can support higher throughput and lower latency (SHAN, 2006; YEO; LEE, 2011). At the same time, clouds need to be able to support an ever-widening application domain with greater heterogeneity and specifically the need to support both legacy and next generation software applications.

Establishing the right balance between efficiency, resilience and resource utilization while hosting heterogeneous workloads represents a significant challenge for data center managers. A way to reach this objective is to arrange the data center infrastructure in modules called Performance Optimized Data Center (POD) (Figure 1.a). PODs can be defined as a collection of racks, tightly connected through a fast local network, shar-

ing common resources, such as monitoring management, aisle containment, and a Power Distribution Unit (PDU) (SPECIFYING..., 2018; ROSENDO et al., 2019). With this configuration, data center operators can divide up their infrastructure to allocate different workloads to PODs with specific configurations, decoupling the data center space into different demands and facilitating monitoring and management (ROSENDO et al., 2019).

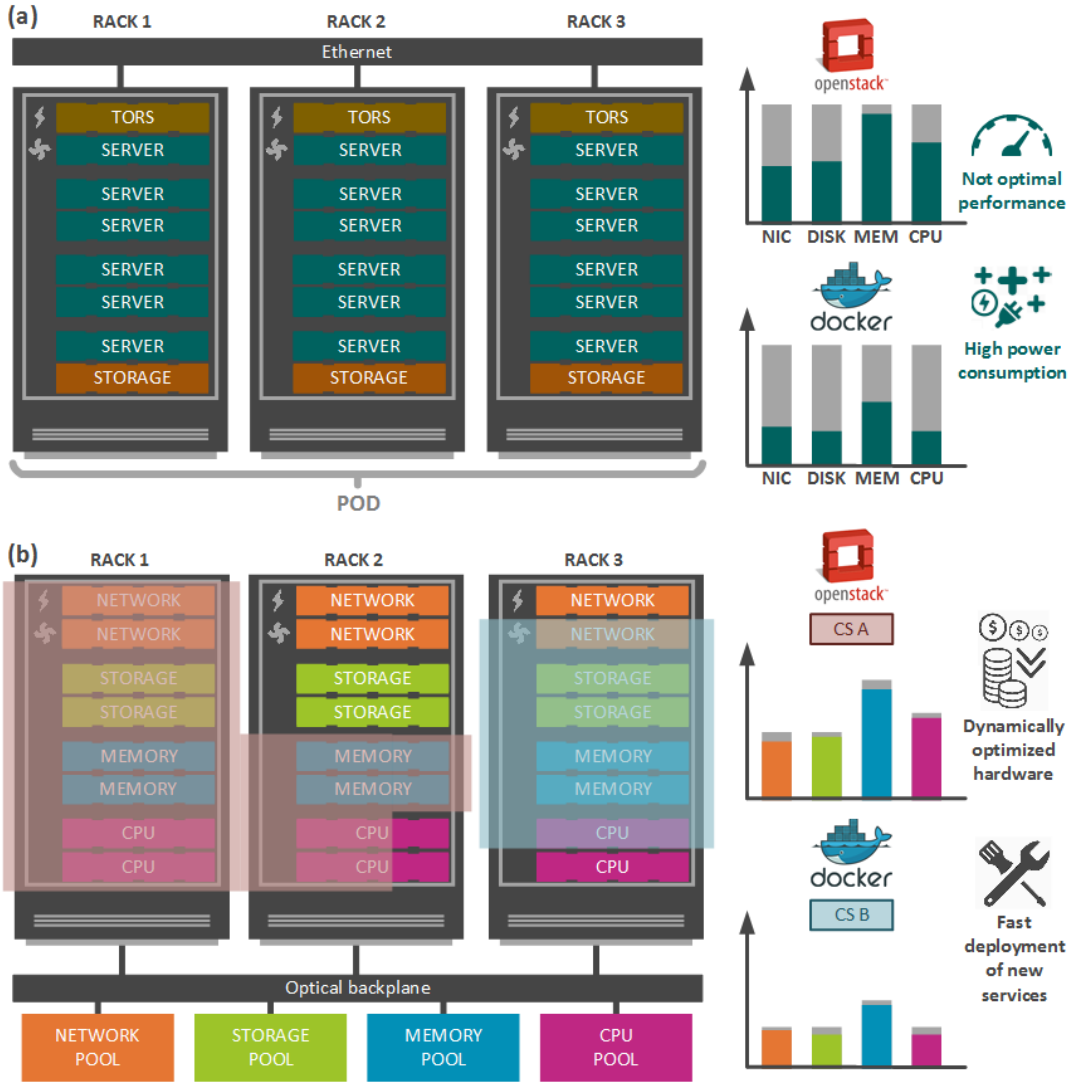


Figure 1 – Differences between POD (a) and (b) composable infrastructure (b) (ROSENDO et al., 2019).

With a POD architecture, it is common to have more hardware resources allocated than what is really required leading to sub-optimal resource utilization which ultimately translates into energy costs (SCHWARTZ; PRIES; TRAN-GIA, 2012). A composable data center infrastructure is based on the concept of a composable systems (Figure 1.b) where compute, memory, storage and network resources are disaggregated, and are assembled and re-assembled using a control software layer and through the deployment of fast optical fiber links (CHUNG et al., 2018). As the hardware resources' allocation is software-based, configurations can be changed dynamically in order to meet the requirements of a given

application or workload. As soon as a hardware resource is no longer required, it can be released for use by another application(s), hence reducing energy consumption and maintenance time (LI et al., 2017). Thus, data centers using composable infrastructure avoid the issue of having idle resources which often occurs in POD configurations (CHENG; GRINNEMO, 2017).

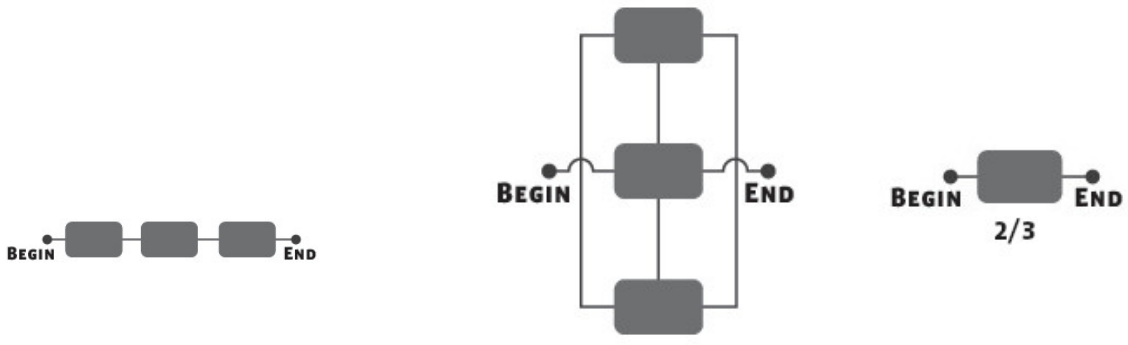
2.2 RELIABILITY BLOCK DIAGRAM (RBD)

A Reliability Block Diagram (RBD) model is a graphical representation of a system components arrangement, providing information on how components relate to each other (ZHANG et al., 2013). The model illustrates the functioning state of the whole system (i.e. success or failure) and captures the network relationships of the system components. By analytically solving RBD models, it is possible to analyze the availability and reliability of complex systems (FAZLOLLAHTABAR; NIAKI, 2017).

There are three main ways to arrange the block in a RBD model: in series, in parallel, or in k-out-of-n configurations (GUIMARAES et al., 2011) (Figure 2). In the series configuration (2.a), if any component fails, the whole system becomes unavailable. In other words, each component must be working in order for the system to be operational.

In a parallel configuration (Figure 2.b), as long as at least one component is working, the system is operational. This kind of configuration can be used to represent redundant components of a system.

Finally, in *k-out-of-n* configurations, the system is operational if k components out of the total n components are working (MACIEL et al., 2010). For example in Figure 2.c), two components must be working for the whole system to be operational.



(a) (b) (c)

Figure 2 – Block configurations in RBD. (a) Series, (b) Parallel and (c) *k-out-of-n* (ROSENDO et al., 2019).

It is possible to solve RBD models analytically and therefore estimate system availability by calculating the availability of each component i of the system as per Equation

2.1.

$$A_i = \frac{MTTF_i}{MTTF_i + MTTR_i} \tag{2.1}$$

The Mean Time To Failure (MTTF) represents the average time length in which the system is available during the period of study while the Mean Time To Repair (MTTR) represents the average time length in which the system was unavailable, and under repair.

The formulas presented in Equations 2.2 and 2.3 can then be used to calculate the availability of series (A_s) and parallel components (A_p), respectively.

$$A_s = \prod_{i=0}^{N} A_i \tag{2.2}$$

$$A_p = 1 - \prod_{i=0}^{N} (1 - A_i) \tag{2.3}$$

In this work, we use a special type of *k-out-of-n* configuration, called nonidentical *k-out-of-n* independent components, in order to represent heterogeneous components performing the same function on the system (GURLER; BAIRAMOV, 2009). To calculate the availability of this type of RBD model, it is necessary to consider all possible alternative values of k. For instance, for a *2-out-of-3* system which has three components with different configurations and the respective availability, A_1, A_2, and A_3, the availability of the entire system can be calculated as follows:

$$A_k = A_1A_2A_3 + (1 - A_1)A_2A_3 + A_1(1 - A_2)A_3 + A_1A_2(1 - A_3) \tag{2.4}$$

2.3 OPTIMIZATION ALGORITHMS

In this section, we provide more details of the algorithms used in our optimization experiments, divided into Mono-objective and Multi-objective problem. The optimization problem, modeled in these two approaches, utilized in this work are presented in 3.1 and 4.1.

2.3.1 Mono-objective Optimization Methods

In the mono-objective problem, the problem is mathematically described through a unique objective function composed of variables of the problem. The best solution represents the minimum or maximum value obtained from the objective function (POIAN et al., 2008). In this section, we present three mono-objective methods utilized in this work.

2.3.1.1 Dynamic Programming

Dynamic Programming (DP) is a mathematical optimization method commonly used to solve multi-stage allocation problems (BELLMAN, 2013), problems which decisions are made at several stages. DP is often used to solve problems with a large number of decision variables and may handle both probabilistic and deterministic variables (SHENOY; SRIVASTAVA; SHARMA, 1986). It was the first exact method proposed to solve the knapsack problem (BOYER; BAZ; ELKIHEL, 2012), but its is also used in different classes of problems.

The rationale of DP is to decompose a whole optimization problem into low-order sub-problems, as a multi-stage decision process. In other words, for each stage of a problem, there are states of the process which keep a solution of the sub-problem, helping in future decision making (RONG; FIGUEIRA, 2014). Thus, these more straightforward sub-problems, are sequentially solved and their stored solutions are recursively used to solve higher-order sub-problems (RONG; FIGUEIRA; KLAMROTH, 2012); this way, DP can avoid re-computing.

To achieve the computational efficiency, especially in terms of the evaluation of all possible sequences of decision, DP needs to provide common sub-problems such as sub-problems of a problem that are also sub-problems to another. Thus, a specific solution from a sub-problem requires to be found only one time, and can then be used as often as necessary (LEW; MAUCH, 2006).

In this work, we have adapted the traditional DP algorithm to fit our optimization problem. The resulting adapted approach is described in Section 3.3.1. The DP algorithm was selected for this study because it is the ease of implementation algorithm, frequently utilizing for Knapsack problem (FAWZY; SANGADJI; AS'AD, 2017); it presents efficiency in NP-hard problems, and can significantly reduce the search space, efficiently find an optimal solution (YANG et al., 2018).

2.3.1.2 Differential Evolution

An efficient and also simple technique is the Differential Evolution algorithm (DE), which is one of the most relevant evolutionary algorithms (STORN; PRICE, 1996). Some characteristics that made DE a simple algorithm include its use of a reduced number of parameters, offering a good convergence, and working with inexpensive and straightforward arithmetic operators (TANG; ZHAO; LIU, 2014).

DE adopts the strategy of greedy selection and is less of a stochastic approach in comparison with other algorithms towards solving optimization problems. Among evolutionary algorithms, DE stands out due to the use of mutation and recombination phases. Furthermore, this algorithm implements the weighted differences approach between solution vectors to disturb a given population (DRAA; BOUZOUBIA; BOUKHALFA, 2015).

A DE procedure can be summarized in four steps described in (STORN; PRICE, 1997):

the first step refers to the initialization of the population where the vector, which represents the population, is defined randomly. The second step comprises the mutation. This step occurs when there is a preliminary solution available, and the initial population generated through the addition of random deviations. These are generally distributed to the nominal solution. The addition of the weighted difference between the two population vectors in a third vector generates new parameter vectors. After that, the crossover (or recombination) is performed. In this step, the mutated vector's parameters are combined with parameters of another predetermined vector, called the target vector, to provide the trial vector. Finally, in the selection step, the trial vector replaces the target vector, in case of the trial vector offers a lower cost function value.

In this algorithm, repetitions only take place in the subsequent DE iterations, in the last three steps. These iterations proceed until a termination criterion is satisfied (DAS; MULLICK; SUGANTHAN, 2016). DE was selected for this study due to its success applied to solve a large number of real-world problems from different domains of science and technology (DAS; MULLICK; SUGANTHAN, 2016); it is a robust and simple structure algorithm; besides its parallel search ability and quick convergence (WU; CHE, 2019).

2.3.1.3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is an optimization algorithm that takes advantage of the swarm intelligence mechanisms and has been widely adopted for dealing with population-based optimization problems.

This algorithm is biologically inspired by a flock of birds behavior, where a set of solutions to a problem (swarm) is called population (GHAMISI; BENEDIKTSSON, 2015) (GHAMISI et al., 2014). A population is a set of parameters and represents a point in the solution space of the problem. As mentioned in the literature, "*In nature, a bird usually adjusts its movement to find a better position in the flocking according to its own experience and the experience of birds nearby.*" (LIU; DU; WANG, 2014).

Following this flocking birds' behavior, a PSO algorithm starts with random particles in a solution space, and each particle begins with its associated position and velocity. Then, the particles move through a solution space with the purpose of finding the optimal solution. The position and the velocity are updated based on a particle's own experience and that of its neighbors. Also, the positions distinguish between global and personal best, based on the fitness function. Each movement performed by particles is entirely influenced by its current position, its parameters, and group knowledge of the swarm (GHAMISI; BENEDIKTSSON, 2015) (DU et al., 2015).

According to (DU; SWAMY, 2016), PSO became popular due to two main factors: the first is related to its simple implementation and the second reason refers to its fast convergence. When compared with other algorithms such as ant colony and evolutionary algorithms (EAs), PSO is advantageous in terms of its requirement of primitive math-

ematical operators only, smaller computational bookkeeping, and use of fewer lines of code.

PSO was selected for this study due to its behavior to use a swarm method that simultaneously searches vast region in the solution space of the optimized objective function (WANG; TAN; LIU, 2018), it is a robust algorithm, with fast convergence, besides may be applied in different problems.

2.3.2 Multi-objective Optimization Methods

In multi-objective optimization, several conflicting objectives are optimized simultaneously, does not restrict to find a unique solution. Instead, a set of solutions, know as Pareto Optimal or Pareto Front, are presented (LAURENT et al., 2014). In this section, we present two multi-objective methods utilized in this work.

2.3.2.1 Non-dominated Sorting Genetic Algorithm

The Non-dominated Sorting Genetic Algorithm (NSGA-II) is considered one of the most potent multi-objective evolutionary algorithms (MOEAs) (VO-DUY et al., 2017) and has been developed by combining NSGA and the non-dominated sorting concept proposed by Goldberg (GOLDBERG; HOLLAND, 1988).

A characteristic that makes NSGA-II an algorithm with good convergence towards the Pareto-optimal front keeping the solution diversity is its procedure of maintaining the elitism in the population, where the best individuals receive the opportunity to be transferred to the next generation due to their condition (KAMJOO et al., 2016).

When implementing NSGA-II, the number of population (N) and the initial population (Pt) are generated randomly. Objective functions are initially calculated for the initial population. Then, when members are sorted, the rush criterion function is determined. Parents are chosen using binary tournament selection based on a degree – which means that between two solutions in the current population, the best solution is selected – and crowding distance (CD) – a measure of the density in neighbors solutions; parents and off-springs compete with each other to be included in the next generation. Next, mutation and combination operators also are implemented. The new population is then combined with the initial one, and the sorting operation continues. In this new population, N is chosen for the next generation according to the top members of the population (ESFE et al., 2017; BOUACHA; TERRAB, 2016).

NSGA-II was selected for this study due to present better speed, spread of the solution and convergence in comparison to another algorithms (SINGH; SINGH, 2017) and computational simplicity. Additionally, this algorithm is largely used in literature to solve real-world and engineering problems.

2.3.2.2 Generalized Differential Evolution

Generalized Differential Evolution (GDE3) is a multi-objective approach based on the classic Differential Evolution (DE) algorithm. It is a successor of the GDE and GDE2 versions that use non-dominated sorting (JESUS; RIVERA, 2018). The main difference of this version to the previous ones in in the extension of DE/rand/1 strategy that introduces problems with M objectives and K constraints functions (KUKKONEN; LAMPINEN, 2005).

This algorithm shows better-distributed solutions when compared with its previous versions. Furthermore, its performance demonstrated in a different set of tests problems in the literature presented good results when dealing with multi-objective problems (SAFI; UCAN; BAYAT, 2018; TSARMPOPOULOS et al., 2019).

In this algorithm, the DE selection operator introduces the Pareto dominance approach. In case the trial vector weakly dominates the decision vector, the decision vector is replaced by the trial vector in the next generation. Both vectors may be preserved in the population if there is no dominant vector. The size of the population may increase at the end of a generation; therefore, the size of the current population is truncated to the original size through the use of the CD-based selection method. In this method, it is posssible to approximate the crowdiness of the vector in its non-dominated set (KUKKONEN; LAMPINEN, 2005). Non-dominance and crowdiness approaches are used to sort the vector; the worst solutions must be removed (CREMENE et al., 2016).

GDE3 was selected for this study due to characteristics to improve the ability to handle multi-objective problems by providing a better-distributed set of solutions, besides be less sensitive to the selection of control parameter values in comparison to the earlier GDE versions (ADEKANMBI; GREEN, 2015).

2.4 RELATED WORK

While there is a significant body of literature on resource allocation in cloud computing using evolutionary algorithms, resource allocation in composable data center infrastructures is a relatively nascent area. The use of evolutionary algorithms for this purpose is at an even earlier stage. Notwithstanding that, there are several related work worth noting. We divided this section into evolutionary algorithms and composable data infrastructures works.

About evolutionary algorithm works, Rankothge et al. (RANKOTHGE et al., 2017) present a proposal for a resource allocation algorithm for Virtualized Network Functions (VNFs) based on Genetic Programming (GP). GP performance was compared against traditional Integer Linear Programming (ILP). The comparison was performed under two scenarios: *(1)* initial VNFs were instantiated, and (2) additional VNFs were instantiated to satisfy changes in traffic with minimal impact on network performance (scaling). In this context, the authors aim to minimize required resources, such as number of servers,

number of links, and average link utilization adjusting the resources to satisfy traffic changes and minimizing the number of configuration changes to reduce potential service disruptions, and performance degradation. The results showed that GP took less time to generate a VNF configuration, more specifically, GP spent a few milliseconds while the ILP, took several hours. Authors also conducted evaluation of the performance of the NFC Management System when using the proposed GP approach. They established three network architectures to evaluate: k fat tree, VL2 and BCube. GP results showed a reduced the average link utilization for the three architectures.

Gai et al. (GAI; QIU; ZHAO, 2016) solved the problem regarding the constraints faced by heterogeneous cloud memories, as well as the costs of hardware distribution. A solution was proposed to provide high performance cloud-based heterogeneous memory service offerings. In addition, they addressed some sets of factors that impact cloud memory performance, such as communication and operating cost, data and energy performance, and time constraints. Experiments were performed using Cost-Aware Heterogeneous Cloud Memory Model (CAHCM), and Dynamic Data Allocation Advanced (2DA) that uses genetic programming to determine the data allocations on the cloud-based memories and to find the optimal solutions and sub-optimal data allocation. The obtained results reached the level desired by the authors. Using CAHCM, it was possible to allocate data within 5.1 ms using 5 cloud memories, which is much smaller compared to the traditional method which required 22903.3 ms.

Ficco et al. (FICCO et al., 2018) proposed a metaheuristic approach for cloud resource allocation based on the bio-inspired coral-reefs optimization paradigm to model cloud elasticity in a cloud data center, and on the classic Game Theory to optimize the resource reallocation schema with respect to cloud provider's optimization objectives. In the model, authors considered a scenario with a cloud data center composed of several computing nodes, such as CPU, memory and storage capacity and also considered network bandwidth. In each step of the allocation process, a set of virtual machines could be involved. The solution used evolutionary algorithms based on the observation of the reefs structure and coral reproduction to simulate the continuous requests for new cloud resources. Combined with game theory-based solutions, the algorithm used was able to satisfy the conflicting interests between requests and servers to find a better solution in terms of adaptability and elasticity and improve convergence time performance.

Relevant to the topic of composable data center infrastructure works, is the work by Lin et al. (LIN et al., 2018) who developed queuing models based on network losses to measure the performance of composable infrastructure in terms of utilization of each type of resource pools and drop probabilities with a focus on each type of incoming workloads. The models are extended to consider random workloads demands and the impact of network latency of the composable infrastructure. The authors considered workload requests received by orchestration. This orchestration verifies if there is enough hardware in the

resource pool to satisfy the workload requirement and may accept or reject it. After a workload is completed, the resources return to the workload orchestration, and the workload leaves the system. Three sets of the simulation are considered: (a) a simulation to confirm the utilization and drop probability from the models; (b) a simulation to compare the maximum throughput between the traditional system and the composable infrastructure and; (c) a simulation to measure performance penalty in composable infrastructure when the delay is considering. Their simulations showed that a composable infrastructure could sustain nearly 1.6 times stronger workload intensity than conventional systems while being sensitive to workload distribution demands; it can achieve high system utilization, and; the composable infrastructure data transmission length is relatively smaller compared to traditional data centers.

In (FARIAS et al., 2017), authors developed a simulator to evaluate the behavior of scheduling algorithms applied in warehouse-scale clusters built using the composable infrastructure. The simulator receives as input a workload and its associated infrastructure to be scheduled. The model responds to the job submissions that comprise tasks composed of priority, submission time, placement constraints and resource demands in terms of CPU and RAM. The authors evaluated two scheduling algorithms in equivalent server-based and composable-based infrastructures. These approaches are compared in terms of component size grains and the maximum size of the logical servers. Authors discuss that the scheduler that considers composable infrastructure can allocate, on average, a more significant fraction of workload, in terms of CPU and RAM demand, this number is, respectively, 4.6% and 5.7% larger than in server-based infrastructure. Besides, the scheduler running in composable infrastructure is also able to allocate all tasks of the production system.

In work (AJIBOLA; EL-GORASHI; ELMIRGHANI, 2018), the authors used a mixed-integer linear programming (MILP) model to evaluate and compare the performance of composable rack-scale and pod-scale infrastructures to deduce the optimal scale in this system. Authors considered a scenario with a rack in the data center as a pool of resources, where each rack accommodates many CPU or Memory resources. A traditional data center server is also considered as pool, where there are a single CPU and memory modules. They also considered IO resources integrated into data center communication fabric. In the MILP model, authors addressed workload placement regarding CPU and memory resources demands, resource network traffic and north-south traffic, while minimizing the total CPU, memory and network power consumption. Authors analyzed how composable infrastructure impacted the throughput and energy efficiency of network infrastructure. Among the benefits of disaggregation presented in results are better performance in terms of total network power consumption values and the same CPU power consumption of all data centers architectures. Authors also evaluated the feasible gains in energy efficiency to apply silicon photonic technologies in this infrastructure.

Finally, in (CHUNG et al., 2018) authors demonstrated the operational efficiencies of a composable system through the simulation of the dynamic composition of servers in the cloud, based on traces of GPU-full server clusters. In this work, authors considered a simulation scenario where customers are utilizing GPUs, in increments of considered times units, from a shared pool, which is logically attached to servers via PCIe switches. The results showed that the clusters may support the same amount of workloads with 60% fewer GPUs, which demonstrate the efficiency of composable infrastructure. The authors also defined a composable system concept presenting a real prototype implementation, in addition to describing composable compute and storage accelerator system.

The approaches utilized by works presented in this section search for understanding how to allocate workloads taking into account hardware resources, but without considering their different characteristics or different arrangements. These characteristics become the optimization problem presented in this work a combinatorial problem, and also an NP-hard problem. Aiming to solve this problem, we choose to use metaheuristics approaches. Furthermore, our use of mono-objective and multi-objective approaches, expressed by the use of five different optimization algorithms, to optimize resource allocation in the composable data center infrastructure differs significantly from the studies presented above. This novelty emphasized through our heterogeneity and variability scenarios exploration and the impact of these factors on availability within a constrained budget.

3 OPTIMIZING THE INFRASTRUCTURE ALLOCATION IN COMPOSABLE DATA CENTER: MONO-OBJECTIVE APPROACH

In this chapter, we will present the formulation of mono-objective problem definition, the formulation details of our availability model and its implementation taking into account the three algorithms used in this experiment: Dynamic Programmming, Particle Swarm Optimization and Differential Evolution. We also will present the main results obtained with mono-objective experiments. These results are divided into two scenarios that will be present in this chapter. We will present the parametrization of the algorithms utilized in this approach, and the component importance analysis.

3.1 PROBLEM DEFINITION

We modeled the composable infrastructure allocation problem as follows: we considered four main computing resources to compose a composable infrastructure; they are: compute, memory, network, and storage, that are modeled as types defined by the following set $T = \{cpu, mem, net, sto\}$, all of them are considered in objective function and they do not present any conflict. Each resource type has sub-types indexed; being $i \in \{1, 2, ...I\}$, $j \in \{1, 2, ...J\}$, $k \in \{1, 2, ...K\}$, and $l \in \{1, 2, ...L\}$, where I, J, K, and L are the number of sub-types of *cpu*, *mem*, *net*, and *sto* available at the data center, respectively. Associated with those sub-types there is a range of parameters defined in Tables 1.

Table 1 – Parameters of the optimization model

PARAMETERS	DESCRIPTION
$cost\_cpu_i$	cost of each item of the i^{th} sub-type of the *cpu* resource type
$cost\_mem_j$	cost of each item of the j^{th} sub-type of the *mem* resource type
$cost\_net_k$	cost of each item of the k^{th} sub-type of the *net* resource type
$cost\_sto_l$	cost of each item of the l^{th} sub-type of the *sto* resource type
$mttf\_cpu_i$	MTTF of each item of the i^{th} sub-type of the *cpu* resource type
$mttf\_mem_j$	MTTF of each item of the j^{th} sub-type of the *mem* resource type
$mttf\_net_k$	MTTF of each item of the k^{th} sub-type of the *net* resource type
$mttf\_sto_l$	MTTF of each item of the l^{th} sub-type of the *sto* resource type
$mttr\_cpu_i$	MTTR of each item of the i^{th} sub-type of the *cpu* resource type
$mttr\_mem_j$	MTTR of each item of the j^{th} sub-type of the *mem* resource type
$mttr\_net_k$	MTTR of each item of the k^{th} sub-type of the *net* resource type
$mttr\_sto_l$	MTTR of each item of the l^{th} sub-type of the *sto* resource type
$b\_mem_i \in N^*$	number of items available of the j^{th} sub-type of the *mem* resource type
$b\_net_k \in N^*$	number of items available of the k^{th} sub-type of the *net* resource type
$b\_sto_l \in N^*$	number of items available of the l^{th} sub-type of the *sto* resource type
$b\_cpu_i \in N^*$	number of items available of the i^{th} sub-type of the *cpu* resource type

The range of these parameters is specified in 3.3.4. Considering those parameters, we define a set of decision variables that describe the number of selected items of each sub-type of a resource type, they are x_i, y_j, w_k, z_l, where $x_i \in \{0, ..., b\_cpu_i\}$ for each sub-type i of the *cpu* type; $y_j \in \{0, ..., b\_mem_j\}$ for each sub-type j of the *mem* type; $w_k \in \{0, ..., b\_net_k\}$ for each sub-type k of the *net* type; and $z_l \in \{0, ..., b\_sto_l\}$ for each sub-type l of the *sto* type.

This way, we define the following Integer Program in order to find an composable infrastructure allocation obeying a given budget B. The application requirement is composed of a set of computing resources.

$$
\begin{aligned}
\text{maximize} \quad & availability(x_1, ..., x_I, y_1, ..., y_J, \\
& w_1, ..., w_K, z_1, ..., z_L, \\
& mttf\_cpu_1, ..., mttf\_cpu_I, \\
& mttr\_cpu_1, ..., mttr\_cpu_I, \\
& mttf\_mem_1, ..., mttf\_mem_J, \\
& mttr\_mem_1, ..., mttr\_mem_J, \\
& mttf\_net_1, ..., mttf\_net_K, \\
& mttr\_net_1, ..., mttr\_net_J, \\
& mttf\_sto_1, ..., mttf\_sto_L, \\
& mttr\_sto_1, ..., mttr\_sto_L, \\
& R_{cpu}, R_{mem}, R_{net}, R_{sto}) \\
\text{subject to} \quad & \sum_{i=1}^{I} cost\_cpu_i x_i + \sum_{j=1}^{J} cost\_mem_j y_j +, \\
& \sum_{k=1}^{K} cost\_net_k w_k + \sum_{l=1}^{L} cost\_sto_l z_l < B, \\
& \sum_{i=1}^{I} x_i \geq R_{cpu}, \\
& \sum_{j=1}^{J} y_j \geq R_{mem}, \\
& \sum_{k=1}^{K} w_k \geq R_{net}, \\
& \sum_{l=1}^{L} z_l \geq R_{sto}
\end{aligned}
$$

In the four last constraints, the terms $R_{cpu} \in N^*$, $R_{mem} \in N^*$, $R_{net} \in N^*$, and $R_{cpu} \in N^*$ are the application requirements defined by the user, which can be defined,

respectively, as the minimum number of items of the *cpu* resource type required by the application; the minimum number of items of the *mem* resource type; the minimum number of items of the *net* resource type; and the minimum number of items of the *sto* resource type.

The function *availability*() estimates the availability of a specific candidate solution from an RBD model that is parameterized by the decision variables, the application requirements(which are related to the selection of items), MTTF values, and MTTR values (which are directly correlated with each other in the availability calculation) and are solved through numerical algorithms. The arguments of availability function are parameters and variables. Depending on the values assumed by the decision variables in the candidate solution, a specific RBD model is composed, as final product of availability function, and solved in order to estimate the corresponding availability.

The problem proposed in this approach is a bounded multidimensional knapsack problem (KELLERER; PFERSCHY; PISINGER, 2004), where a set of *cpu*, *mem*, *net* and *sto* represents the elements to be inserted in the knapsack. The problem can be considered multidimensional because each resource type adds a different dimension and new constraints that define the minimum required resources (given by *Rcpu*, *Rmem*, *Rnet*, and *Rsto*). Moreover, the problem is bounded because for each sub-type of a resource type, a certain number of resources which are limited by the number of available resources (given by b_cpu_i, b_mem_j, b_net_k, and b_sto_l) must be chosen.

The problem proposed here can be proven to be NP-hard, since the classical 0-1 knapsack problem is a particular case of this problem when there is only one dimension, the objective is a linear function, there are no minimum requirements (i.e. $Rcpu = Rmem = Rnet = Rsto = 0$), and that each sub-type has only one resource available (i.e., $b\_cpu_i = b\_mem_j = b\_net_k = b\_sto_l = 1, \forall i, j, k, l$) (MARTELLO; TOTH, 1990).

3.2 AUTOMATIC GENERATION OF THE AVAILABILITY MODEL

As described previously, the *availability*() function is implemented to estimate the availability of each candidate solution through the use of an RBD model. For each candidate solution an RBD model is automatically generated, which changes at each iteration of the optimization algorithm.

The RBD approach was adopted to calculate the availability of each possible solution as this technique *(a)* provides a measure of the dependability of a logical system (VERMA; AJIT; KARANKI, 2010), and *(b)* allows the derivation of closed-form equations to calculate the dependability metrics and therefore provide faster results compared to other methods (MACIEL et al., 2017).

To automatically generate RBD models, the independent components are modeled using a nonidentical *k-out-of-n* approach, where k represents the hardware requirement for a specific type of resource and n represents the number of components with the same

type and different sub-type in a particular set of resources assembled in a composable infrastructure.

Four hardware types found in a composable infrastructure were considered i.e. compute, memory, network and storage. Each hardware type is modeled by an RBD block arranged in series which means that the failure of all hardware components of the same type implies a failure of the whole system. The redundancy of hardware components and the redundancy of each sub-type of hardware are arranged in parallel as nonidentical *k-out-of-n* independent components.

In the case of a given components' redundancy, the failures considered are those that do not adhere to the *k-out-of-n* restriction.

As discussed, we consider different hardware configurations. For instance, we could have processor A (8 cores) and processor B (4 cores), each with different characteristics in terms of manufacturer, cost, MTTF and MTTR values.

3.3 ALGORITHMS IMPLEMENTATION

In this section, we will present the different fitness function implementations applied for Dynamic Programming and for evolutionary algorithms, respectively. All algorithms represent the same problem formulation. For evolutionary algorithms, we consider the implementations present in the NMOF[1] package from R. The implementation present in 3.3.2 represents only the fitness function.

3.3.1 Dynamic Programming

In this work, we implemented the DP algorithm using a bottom-up approach, as shown in Figure 3. A matrix M of dimension $M[w]$x$[B]$, where $w = b\_cpu_i + b\_mem_i + b\_net_i + b\_sto_i$, is populated sequentially, accessing directly the calculated entries from the matrix itself. This approach avoids the recursion overhead, improving the code performance. The available budget and all costs have real values, but those values can be converted to an integer value.

Each entry of the matrix M consists in the availability result of adding a new element (hardware resource) in the knapsack (in the RBD model). Moreover, our implementation only considers solutions that obey the budget constraint, as well as, the application requirements (in terms of compute, memory, network, and storage).

Due to this constraint, before running the DP, we should organize the hardware resources (the lines of matrix M) as follows: $cpu_1, mem_1, net_1, sto_1, cpu_2, mem_2, net_2, sto_2, ...,$ $cpu_n, mem_n, net_n, sto_n$. In other words, the items of each type are sorted in the decreasing order according to their cost. This way, cpu_1, for example, is the most expensive *cpu* item and cpu_n the least expensive. Finally, these sorted items are interleaved combining

[1] https://cran.r-project.org/web/packages/NMOF/index.html

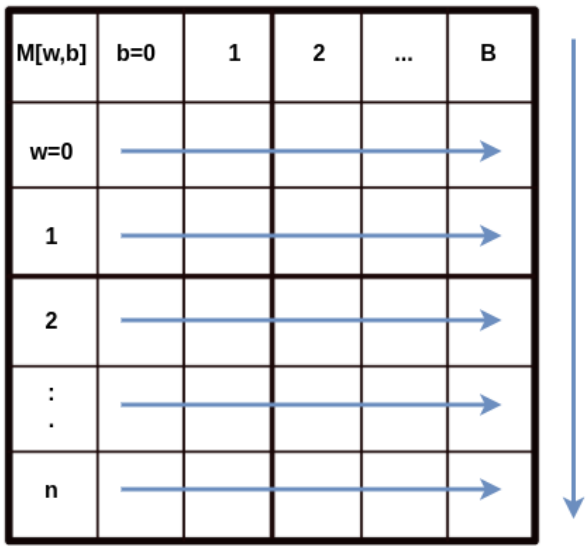


Figure 3 – An example of Bottom-Up DP approach

one item of each resource in a sequence. This item sequencing avoids the fact that the algorithm does not meet the application requirement or finds bad availability results. It is important to note that DP is an algorithm that does not include any random process in its implementation. Due to this fact, this algorithm is executed only one time for each experiment.

3.3.2 Evolutionary Algorithms

The evolutionary algorithms are specified through some components, procedures, or operators, indicated by Eiben et al. (EIBEN; SMITH et al., 2003) and cited as follows:

- **Representation**: The link between the real-world and the evolutionary world. The objects that compose possible solutions in a problem context are phenotypes, and the individuals (phenotypes encoding) are genotypes.
- **Fitness Function**: A function that designates a quality measure to genotypes, representing the requirements the population should adapt to meet, the basis for selection.
- **Population:** It represents a multi-set (set where multiple copies of an element possible) of genotypes, that forms the unit of evolution.
- **Parent Selection Mechanism:** To distinguish individuals based on defined quality to allow the best of them to become parents in the next generation. An individual becomes a parent when it is selected to undergo variation to create offspring.
- **Variation Operators:** Responsible for creating new individuals from the old ones; They are mutation and recombination.

1. **Mutation:** A unary variation operator. This variation is applied to one genotype and generates a slightly modified mutant, called child or offspring.
2. **Recombination:** A unary variation operator. This variation merges information from two parents' genotypes into one or two offspring genotypes.

- **Survivor Selection Mechanism:** To distinguish individuals based on their quality where the survivor selection mechanism runs after the creation of the offspring from selected parents. It may be based on the age-biased approach to choose the next generation.

The fitness function implementation used for both DE and PSO algorithms is presented in this section: For each candidate solution given by the algorithm we calculate infrastructure availability and cost. We determine infrastructure availability through the RBD model utilizing *K-out-of-N* method, based on MTTF and MTTR values of components. The cost is calculated based on individual cost of each component selected in the candidate solution (as shown in 3.1). After that, we applied a penalty function (3.1) about the availability value generated. The penalty function is described follow:

$$p = \frac{B - C_{component}}{C_{infrastructure}} \tag{3.1}$$

$$min(0, p) \tag{3.2}$$

where B is the Budget, $C_{component}$ is the cost of the candidate solution and $C_{infrastructure}$ is the cost of all available components in the data center. However, this penalty function only must be applied when the cost of the candidate solution is greater then B value. To guarantee this, we also use the minimum value between 0 and penalty function (Equation 3.2). Thus, the result of this must be added to the candidate solution availability.

3.3.3 Algorithms Parametrization

To solve the composable infrastructure allocation problem using DE, we defined a set of parameters for this algorithm based on (PEDERSEN, 2010a) and presented in Table 2. For PSO, we defined its parameters based on (PEDERSEN, 2010b) and presented in Table 3. In these two studies, the authors performed several experiments for determining the best set of DE and PSO parameters, respectively, based on the number of variables of the problem. Taking into account these studies, the set of parameters used in the present work is based on the maximum number of variables that the problem achieves, corresponding to 64 variables. We only adapted the number of generations for each algorithm to 5,000 through empirical tests.

We defined a fitness function, that it is applied for both algorithms. This function receives as parameter an integer vector that contains the number of components for each

sub-type. As the original DE and PSO implementations work with real vectors, we needed to convert the representation vector to the integer vector.

Table 2 – DE parameters

Probability for crossover	95.65%
Step Size	58.24%
Population	46
Number of generations	5,000

Table 3 – PSO parameters

Inertia Weight	-0.2089
Deviation of the initial velocities	2
Population	161
Weight towards the individuals best solution	-0.0787
Weight towards the population best solution	3.7637
Number of generations	5,000

3.3.4 Scenarios

We explored two different scenarios for the optimization of composable infrastructure allocation. In the first one, we maintained constant the number of items per sub-types and varied only the number of sub-types as shown in (Table 4) in order to evaluate the **Heterogeneity**. Figure 4 represents the Heterogeneity scenario. This scenario aims to represent how less component sub-type variation affects the compose of a high availability associated with a low cost in composable Infrastructure. The resource pool presented in these scenarios is based on the number of sub-types of all available types of this scenario. The budget is defined as follows: we take the price of the application requirements and increase 10% of this value for each scenario, varying from 10% to 40%. Each experiment was executed 10 times for each evolutionary algorithm.

The goal of the second scenario is to evaluate **Variability**. To achieve this, we set the number of sub-types to 8, and we varied the number of items per sub-type (Table 5). The budget follows the same setting of the Heterogeneity scenario. The resource pool is based

Table 4 – Heterogeneity Scenario Settings

Experiment	Number of Sub-Types	Items per Sub-Type	Application Requirement	Resource Pool	Available Budget ($)
1	2	1	1	8	629.00
2	4	1	1	16	686.00
3	8	1	1	32	743.00
4	16	1	1	64	800.00

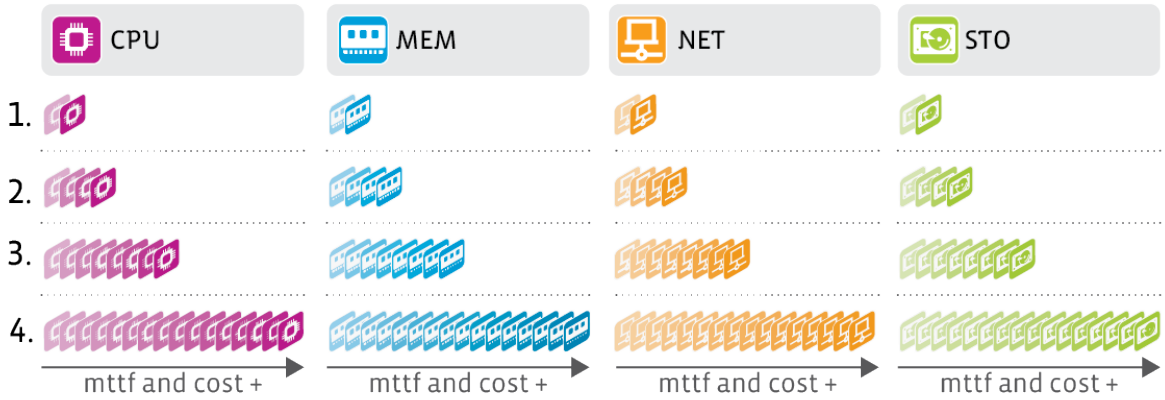


Figure 4 – Resource pool from Heterogeneity Scenario

on the number of sub-types of all available types of this scenario. Figure 5 shows the Variability scenario. This scenario aims to represents how a higher component sub-type variation affects the compose of a high availability associated with low cost in composable Infrastructure. Each experiment was executed 10 times for each evolutionary algorithm.

Table 5 – Variability Scenario Settings

Experiment	Number of Sub-Types	Items per Sub-Type	Application requirement	Resource Pool	Available Budget ($)
1	8	1	1	32	629.00
2	8	2	1	64	686.00
3	8	3	1	96	743.00
4	8	4	1	128	800.00

The MTTF and MTTR base values used in this paper and the cost of each component are presented in Table 6. There are variations in MTTF and cost of hardware resources for each sub-type from these base values, that are expressed in equations 3.3 and 3.4.

$$random(mttf, mttf + mttf * 0.1) \tag{3.3}$$

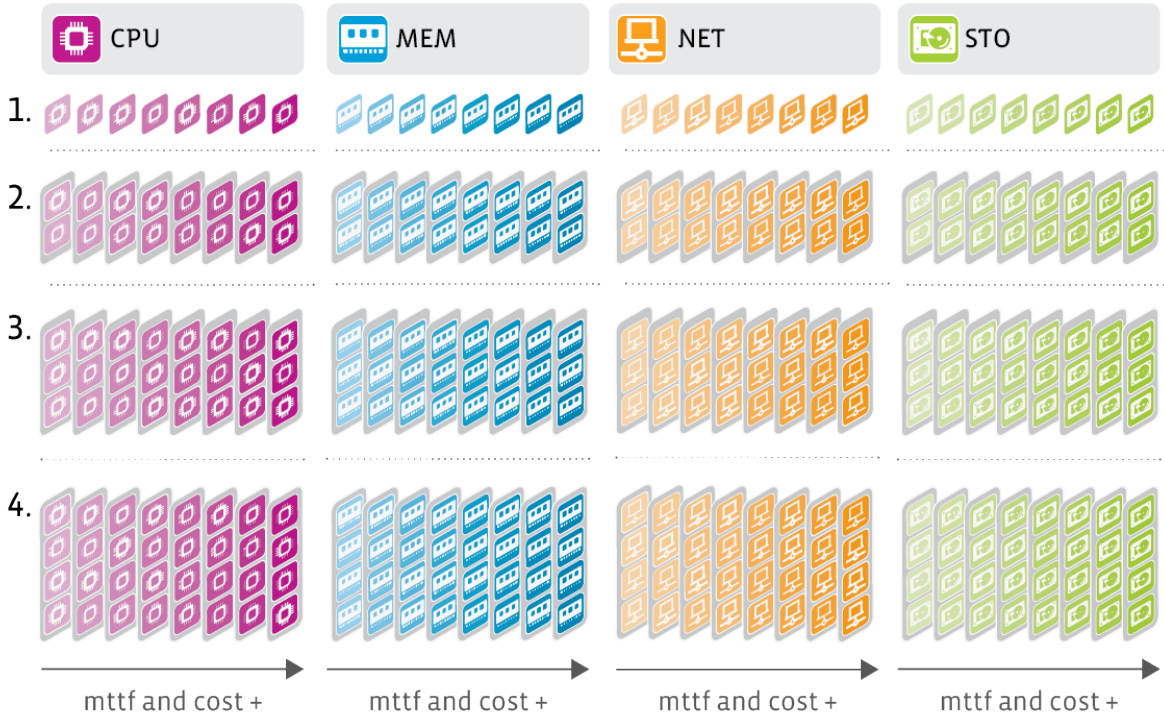


Figure 5 – Resource pool from Variability Scenario

$$random(cost, cost + cost * 0.1) \tag{3.4}$$

Table 6 – MTTF, MTTR, and cost values of the hardware components (from (SMITH et al., 2008; ARAUJO et al., 2014; BROSCH et al., 2010))

Component	**MTTF (in hours)**	**MTTR (in hours)**	**Component Cost ($)**
Compute	292,000.00	6.0	370.00
Memory	480,000.00	2.5	130.00
Storage	200,000.00	2.5	45.00
Network	120,000.00	2.5	27.00

3.4 COMPONENT IMPORTANCE ANALYSIS

The importance of system components can be estimated based on two aspects: reliability and availability. Component reliability is directly related to the MTTF parameter, while the availability component is related to MTTF and MTTR parameters. It is desirable to improve system availability and reliability by making a small investment. Thus, the cost of the component must be taken into account to estimate its importance (FIGUEIRÊDO et al., 2011).

We use the RBD model presented in Figure 6 to calculate the availability and reliability importance of each component present in our scenarios (GUIMARAES et al., 2011). We have one block for each hardware resource (CPU, memory, network and storage), which receives the MTTF and MTTR parameters and the respective hardware resource cost, as shown in Table 6. We use the software Mercury[2] to calculate these importance values, based on Birnbaum Importance (I_i^B) equation. A reliability importance (RI) is shown in Equation 3.5 as example, where p^i is the component reliability vector with ith component removed; 0_i is the condition when component i failed; and 1_i is the condition when i is working. The availability importance equation follows the same structure.

$$I_i^B = R_s(1_i, p^i) - R_s(0_i, p^i) \tag{3.5}$$

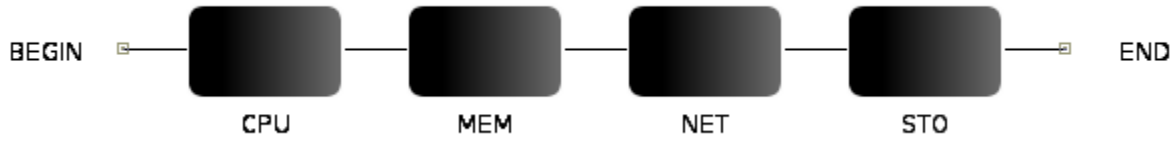


Figure 6 – Composable Infrastructue RBD model

Table 7 illustrates reliability (considering one year) and availability importance values of the hardware resource. The network is the component with the highest value of both importance values, followed by the storage, while the memory and compute have the lowest ones. Therefore, a little investment in the network results in a greater increase of the composable infrastructure availability, while a greater investment in the compute is needed to result in a similar improvement on availability.

Table 7 – Importance values

Component	**Availability Importance**	**Reliability Importance**
Network	0.95276	0.8690
Storage	0.92128	0.8161
Memory	0.77268	0.6672
Compute	0.35313	0.6672

3.5 MONO-OBJECTIVE OPTIMIZATION ANALYSIS

In this section, we will present the main results obtained in Heterogeneity and Variability scenarios of mono-objective approach in terms of availability, cost and arrangement of solutions, as well as, a discussion about them. Taking into account the fact that we

[2] http://www.modcs.org/?page_id=2392

performed several executions for each experiment of the evolutionary algorithms, in this section, we discuss the results from an random independent execution. The summary from all executions are present in section 3.5.1.

3.5.1 Evolutionary Algorithms Executions

In this section, we discuss the results obtained from 10 executions of each experiment, from both scenarios of the evolutionary algorithms. We calculated the mean and standard deviation obtained from availability and cost values. To analyze the different solutions obtained in each experiment for DE and PSO algorithms, we applied the non-parametric Kruskal-Wallis Test (VARGHA; DELANEY, 1998), to verify statistical equality in the models.

In Table 8, we present the summary of executions obtained from Heterogeneity Scenario. In Experiment 1, we noted that PSO obtained a higher mean availability and a lower mean cost, but applying the statistical test, we obtained a p-value of 0.4853, so there was no statistical difference between the sets obtained by PSO, and DE.

Similar behavior was obtained for other experiments: in Experiment 2, DE obtained a higher mean availability, and a lower mean cost, but a p-value of 0.7001 established the statistical equality for both models; in Experiments 3 and 4, the means availability were closer to each other, but the means cost were better for Experiment 3 for DE a better for PSO in Experiment 4. In both cases, the models presented, respectively, a p-value of 0.2961 and 0.6954, which indicates the statistical equality.

Table 8 – Heterogeneity Summary from 10 execution of DE and PSO Algorithms

DE | Heterogeneity				
Experiment	**Mean Availability**	**Standard Deviation**	**Mean Cost**	**Standard Deviation**
1	99.99567447779822	8.290920590492531e-06	618.5	10.229858259037611
2	99.99684108573146	1.0407618914247076e-05	672.7	8.307225770376053
3	99.99742441440498	2.7794627992550146e-15	733.2	9.421252570651102
4	99.99742442243051	6.552724635342446e-11	788.2	9.119210492142399
PSO | Heterogeneity				
Experiment	**Mean Availability**	**Standard Deviation**	**Mean Cost**	**Standard Deviation**
1	99.99600778638271	3.333041259567793e-06	615.3	9.869650449737316
2	99.99671608442029	1.0382624210616162e-05	676.9	13.209466302617983
3	99.99742440697396	1.486197831654857e-10	737.8	3.6
4	99.99742442109303	6.687871373367664e-11	786.0	17.332051234634633

In Table 9, we present the summary of executions obtained from Variability Scenario.

In Experiment 1, DE obtained better results than PSO. These models had a statistical difference demonstrated to a p-value of 0.0377. Other experiments demonstrated statistical equality to each other. In Experiment 2, DE obtained a better mean availability and a mean cost, but comparing with PSO model, a p-value of 0.6114 was obtained. In Experiment 3, PSO obtained a better mean availability and the same mean cost obtained to DE models. In this case, the models demonstrated a p-value of 0.59048. Finally, in Experiment 4, PSO obtained better mean availability, and DE obtained better cost. These models demonstrated a p-value of 0.4151.

Table 9 – Variability Summary from 10 execution of DE and PSO Algorithms

DE | Variability				
Experiment	**Mean Availability**	**Standard Deviation**	**Mean Cost**	**Standard Deviation**
1	99.99575780382971	8.332692319545919e-06	620.2	6.954135460285484
2	99.99729939406633	3.749728223967141e-06	681.5	3.0740852297878796
3	99.99717443199442	4.999781971234736e-06	739.3	2.794637722496424
4	99.99752857977846	2.0830733880522285e-06	789.9	6.024118192731613
PSO | Variability				
Experiment	**Mean Availability**	**Standard Deviation**	**Mean Cost**	**Standard Deviation**
1	99.84550572909624	0.0030720270547191634	633.9	24.50081631293129
2	99.98253733181036	0.00029055065906329465	684.5	7.074602462329597
3	99.99742441633689	1.9939956596359852e-10	739.6	3.0397368307141326
4	99.99773688377324	2.551274404057683e-06	794.9	3.726929030716845

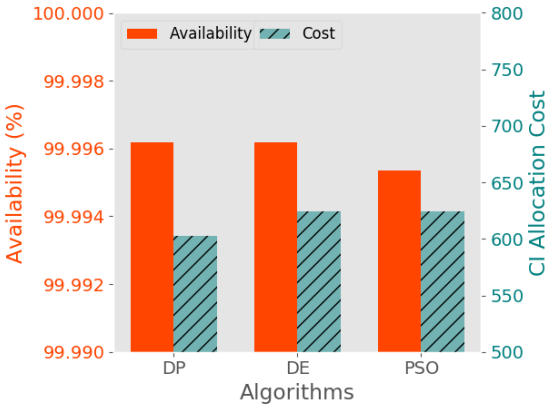
3.5.2 Heterogeneity scenario results

In Experiment 1 corresponding to the heterogeneity scenario (see Figure 7.a), the DP and the DE presented about the same availability (99.9961%, achieving an annual downtime of about 33.51 hours), however DE found a more expensive solution ($624 from DE against $603 from DP). PSO obtained an availability level of 99.9953% (pointing to an annual downtime of about 41.17 hours) and a cost of $624.

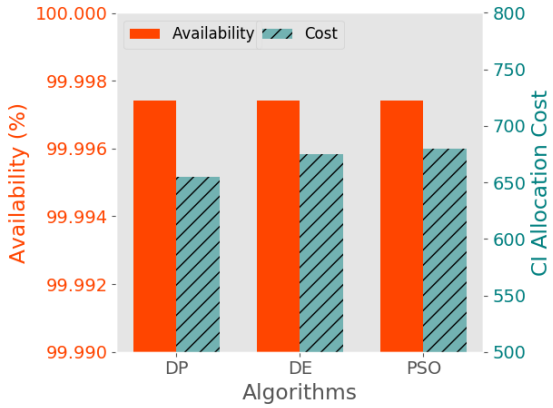
In Experiment 2 (Figure 7.b), the three algorithms obtained roughly the same availability (99.9974%, equivalent to an annual downtime of about 22.56 hours), but with different costs; DP found the best cost ($655), DE found the cost of $622 and the PSO found the worst one ($680).

In the Experiments 3 (Figure 7.c) and 4 (Figure 7.d), we observed an interesting behavior: the availability level stabilizes at around 99.9974% (only DP in Experiment 4 obtained a slightly higher availability of 99.9977%) but the costs of the solutions keep

increasing. In experiment 3, DP found the solution with the best cost ($655) compared to PSO and DE ($737 and $739, respectively). And in Experiment 4, again DP found the best solution (with a higher cost compared to experiment 3 of $757), and DE found the worst solution with the highest cost of $798 while PSO obtained a cost of $782.



(a) Experiment 1



(b) Experiment 2

(c) Experiment 3

(d) Experiment 4

Figure 7 – Results of the heterogeneity scenarios using DP, DE and PSO algorithms of the (a) experiment 1, (b) experiment 2, (c) experiment 3, and (d) experiment 4.

From the algorithms results in the heterogeneity scenario, one can note that the availability is improved from experiment 1 to experiment 2. After that, the availability levels keep almost the same (there are changes only after the 7th decimal places), even when we increase the budget.

In Table 10, we can observe the solutions found by each algorithm (and the cost of each component) in the experiment 1. Regarding this Experiment, we can state that DP found the best solution because it decided to have two redundant network interfaces (the most critical component according to the importance analysis results; see Table 7) and opted for having the cheapest memory. DE also selected two network components, but selected a more expensive memory. Therefore, despite DP and DE achieving about the

same availability (99.99617%, see Figure 7(a)), the DE solution was more expensive. PSO presented the worst result, since it decided for having two redundant storage components (the second critical component, see Table 7).

In Experiment 2 (see Table 11), in order to increase availability, taking in account the increase in the budget value, the algorithms adopted some strategies for choosing their solutions. For example, DP kept the same solution of Experiment 1 but added redundancy in the storage resource which is the second largest critical component according to importance analysis. This configuration allowed DP to establish the best cost-benefit solution. DE chose the cheapest memory and also added redundancy in the storage resource, ensuring the same availability of DP. And PSO added redundancy in the network resource, choosing the second and the fourth more expensive resources, besides, this algorithm chose the more expensive and more available storage resource (was greedy), which justified PSO has obtained the highest cost in this Experiment.

Regarding the availability behaviour across the experiments, we can see that after the second experiment, even with a higher available budget, the algorithms, in general, did not find a solution that increases substantially the composable infrastructure availability. It is interesting to note that although the solutions are more expensive, availability stays practically the same (from experiment 2 to 4) for all algorithms. It is due to the fact that even with the addition of redundant components (which increases the cost of the solution), the availability does not increase considerably since other components also impact availability. In addition, it is not suitable to add redundant components that do not have a major impact on the availability due to limited budget. Thus, it is important to add redundant components that have the greatest impact on availability.

For example, In Experiment 3 (Table 12), DP kept the same solution and cost of Experiment 2 despite the budget increase. Both DE and PSO chose a more expensive and more available memory resource and increased the redundancy of network resources for 3 units in comparison of Experiment 2. These facts only increased the cost of each solution.

Finally, in Experiment 4 (Table 13), only DP achieved a slight increase in availability due to choosing more expensive and more available network and storage resources. DE and PSO tried to increase the availability by adding a 3 units redundancy to the storage resource in comparison to Experiment 3, but availability remained practically the same.

Overall, the DP algorithm obtained the best results (lowest cost and highest availability) due to the redundancy priority in the network, storage, and memory components respectively (most critical components according to the importance analysis results, see Table 7). The DP algorithm solved our problem efficiently due to its nature of decomposing the problem into low-order sub-problems and discarding solutions that are not suitable after verification (BELLMAN, 2013).

Table 10 – Solutions found by DP, DE and PSO for the experiment 1 of the heterogeneity scenario. Considering that there are 2 sub-types in this experiment, the Solution column presents the solution vector with the respective amount of components selected per sub-type.

Heterogeneity | Experiment 1				
Algorithm	**Hardware Resource Type**	**Solution**	**MTTF (in hours)**	**Allocated Hardware Cost ($)**
DP	Compute	[1 0]	292.000	370.00
	Memory	[1 0]	480.000	130.00
	Network	[1 1]	120.000	27.00
			140.168	31.00
	Storage	[1 0]	200.000	45.00
DE	Compute	[1 0]	292.000	370.00
	Memory	[0 1]	560.652	151.00
	Network	[1 1]	120.000	27.00
			140.168	31.00
	Storage	[1 0]	200.000	45.00
PSO	Compute	[1 0]	292.000	370.00
	Memory	[1 0]	480.000	130.00
	Network	[1 0]	120.000	27.00
	Storage	[1 1]	200.000	45.00
			233.613	52.00

3.5.3 Variability scenario results

In experiment 1 of the variability scenario (Figure 8.a), DP and DE obtained the best availability level (both about 99.99617%, meaning an annual downtime of 33.51 hours) while PSO achieved 99.99409% (annual downtime of 51.76 hours). In this case, DP presented the best result because its solution has the lowest cost of $603 against $620 from PSO and $622 from DE.

In experiment 2 (Figure 8.b), the three algorithms achieved about 99.9974% of availability (annual downtime of 22.562), and again DP presented the best cost-benefit solution. While DP obtained a cost of $655, DE and PSO reached $682 and $686 respectively.

In Experiments 3 (Figure 8.c) and 4 (Figure 8.d), we can state that the availability

Table 11 – Solutions found by DP, DE and PSO for the experiment 2 of the heterogeneity scenario. Considering that there are 2 sub-types in this experiment, the Solution column presents the solution vector with the respective amount of components selected per sub-type.

Heterogeneity | Experiment 2				
Algorithm	**Component Type**	**Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**
DP	Compute	[1 0 0 0]	292.000	370.00
	Memory	[1 0 0 0]	480.000	130.00
	Network	[1 1 0 0]	120.000	27.00
			140.168	31.00
	Storage	[1 1 0 0]	200.000	45.00
			233.613	52.00
DE	Compute	[1 0 0 0]	292.000	370.00
	Memory	[1 0 0 0]	480.000	130.00
	Network	[1 1 0 0]	120.000	27.00
			140.168	31.00
	Storage	[0 1 0 1]	233.613	52.00
			289.856	65.00
PSO	Compute	[1 0 0 0]	292.000	370.00
	Memory	[1 0 0 0]	480.000	130.00
	Network	[0 1 0 1]	140.168	31.00
			173.914	39.00
	Storage	[1 0 0 1]	200.000	45.00
			289.856	65.00

stabilized around 99.9979% (annual downtime of 18 hours), and that DP obtained the best cost ($655), followed by the PSO ($737) and DE ($739); And finally in Experiment 4, DP was the best again ($774) followed by PSO ($788) and DE ($793).

The solutions found by each algorithm (and the cost of each component) in Experiment 1 of the variability scenario are presented in Table 14. Again, DP outperformed the others because it was able to select the components that balance better availability and cost. DP

Table 12 – Solutions found by DP, DE and PSO for the experiment 3 of the heterogeneity scenario. Considering that there are 2 sub-types in this experiment, the Solution column presents the solution vector with the respective amount of components selected per sub-type.

Heterogeneity | Experiment 3				
Algorithm	**Component Type**	**Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**
DP	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[1 0 0 0 0 0 0 0]	480.000	130.00
	Network	[1 1 0 0 0 0 0 0]	120.000	27.00
			140.168	31.00
	Storage	[1 1 0 0 0 0 0 0]	200.000	45.00
			233.613	52.00
DE	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[0 0 1 0 0 0 0 0]	615.161	166.00
	Network	[1 0 1 0 0 1 0 0]	120.000	27.00
			153.790	34.00
			202.104	45.00
	Storage	[1 1 0 0 0 0 0 0]	200.000	45.00
			233.613	52.00
PSO	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[0 0 1 0 0 0 0 0]	615.161	166.00
	Network	[1 1 1 0 0 0 0 0]	120.000	27.00
			140.168	31.00
			153.790	34.00
	Storage	[0 1 1 0 0 0 0 0]	233.613	52.00
			256.317	57.00

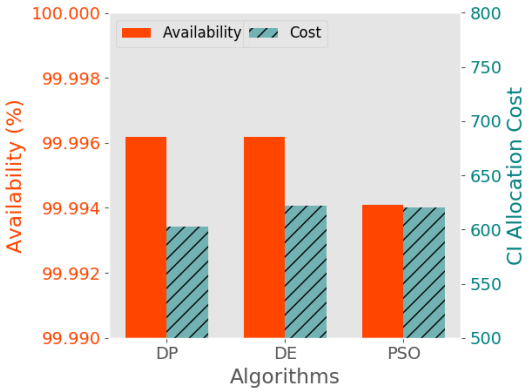
and DE gave priority to allocate redundant network (lower and higher cost respectively, since it is the component that present the highest importance value (see Table 7). PSO presented the worst availability result (Experiment 1), once its solution does not include redundant components. We noted that the three algorithms did not choose any solution

Table 13 – Solutions found by DP, DE and PSO for the experiment 4 of the heterogeneity scenario. Considering that there are 2 sub-types in this experiment, the Solution column presents the solution vector with the respective amount of components selected per sub-type.

Heterogeneity | Experiment 4				
Algorithm	**Component Type**	**Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**
DP	Compute	[0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	341.075	432.00
	Memory	[0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	560.672	151.00
	Network	[0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0]	140.168	31.00
			153.790	34.00
	Storage	[0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0]	233.613	52.00
			256.317	57.00
DE	Compute	[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	480.000	130.00
	Network	[1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0]	120.000	27.00
			153.790	34.00
			173.914	39.00
	Storage	[0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0]	233.613	52.00
			318.929	71.00
			336.840	75.00
PSO	Compute	[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	480.000	130.00
	Network	[1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0]	120.000	27.00
			140.168	31.00
			173.914	39.00
	Storage	[1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0]	200.000	45.00
			289.856	65.00
			336.840	75.00

with component redundancy of the same sub-type.

In Experiment 2 (Table 15), we noted that DP basically maintained the same solution from Experiment 1 but added a redundancy in the storage resource. This redundancy increases availability in comparison to Experiment 1. In this Experiment, DE and PSO



(a) Experiment 1

(b) Experiment 2

(c) Experiment 3

(d) Experiment 4

Figure 8 – Results of the variability scenarios using DP, DE and PSO algorithms in (a) experiment 1, (b) experiment 2, (c) experiment 3, and (d) experiment 4.

achieved the same availability as that of DP but presented more expensive solutions. In the case of DE, what makes the solution more expensive was the choice of 3 network hardware resources rather than 2 (as in Experiment 1), and 2 storage resources of the same sub-type. While in PSO case, this difference of cost is due selecting 3 network hardware resources rather than one, as in Experiment 1. Besides, PSO also added 2 storage resources of the same sub-type.

In Experiment 3 (Table 16), DP kept the same solution of Experiment 2. On the other hand, DE and PSO found different solutions, but these solutions kept about the same availability already found in Experiment 2, but increased in terms of cost. In the DE solution, this fact may be justified due to selecting a network resource that is more expensive and adding a storage resource in comparison to Experiment 2. Regarding the PSO solution, the difference is justified by the use of three storage resources and a more expensive memory resource.

In Experiment 4 (Table 17), the three algorithms showed redundancy for the same sub-type of resources. With DP, the algorithm chose 2 units of memory, network and storage resources, respectively. Only the compute resource did not present redundancy, due to its

higher cost. Furthermore, the algorithm selected only the cheapest resources to apply the redundancy, which guaranteed the best cost but delivered a slightly greater availability in the scenario. DE and PSO presented similar solutions, which applied redundancy of 2 units of the same sub-type of memory whereas network and storage resources presented redundancy in different sub-types.

Table 14 – Solutions found by DP, DE and PSO for the experiment 1 of the variability scenario. Considering that there are 8 sub-types in this experiment, the Solution column presents the solution vector with the respective amount of components selected per sub-type.

Variability | Experiment 1				
Algorithm	**Hardware Resource Type**	**Solution**	**MTTF (in hours)**	**Allocated Hardware Cost ($)**
DP	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[1 0 0 0 0 0 0 0]	480.000	130.00
	Network	[1 1 0 0 0 0 0 0]	120.000	27.00
			140.168	31.00
	Storage	[1 0 0 0 0 0 0 0]	200.000	45.00
DE	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[1 0 0 0 0 0 0 0]	480.000	130.00
	Network	[1 0 0 0 1 0 0 0]	120.000	27.00
			191.357	43.00
	Storage	[0 1 0 0 0 0 0 0]	233.613	52.00
PSO	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[0 1 0 0 0 0 0 0]	560.652	151.00
	Network	[0 0 0 0 0 0 0 1]	240.091	54.00
	Storage	[1 0 0 0 0 0 0 0]	200.000	45.00

Table 15 – Solutions found by DP, DE and PSO for the experiment 2 of the variability scenario. Considering that there are 8 sub-types in this experiment, the Solution column presents the solution vector with the respective amount of components selected per sub-type.

Variability | Experiment 2				
Algorithm	**Component Type**	**Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**
DP	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[1 0 0 0 0 0 0 0]	480.000	130.00
	Network	[1 1 0 0 0 0 0 0]	120.000	27.00
			140.168	31.00
	Storage	[1 1 0 0 0 0 0 0]	200.000	45.00
			233.613	52.00
DE	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[1 0 0 0 0 0 0 0]	480.000	130.00
	Network	[1 1 1 0 0 0 0 0]	120.000	27.00
			140.168	31.00
			153.790	34.00
	Storage	[2 0 0 0 0 0 0 0]	200.000	45.00
PSO	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[1 0 0 0 0 0 0 0]	480.000	130.00
	Network	[0 2 1 0 0 0 0 0]	140.168	31.00
			153.790	34.00
	Storage	[2 0 0 0 0 0 0 0]	200.000	45.00

Table 16 – Solutions found by DP, DE and PSO for the experiment 3 of the variability scenario. Considering that there are 8 sub-types in this experiment, the Solution column presents the solution vector with the respective amount of components selected per sub-type.

Variability | Experiment 3				
Algorithm	**Component Type**	**Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**
DP	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[1 0 0 0 0 0 0 0]	480.000	130.00
	Network	[1 1 0 0 0 0 0 0]	120.000	27.00
			140.168	31.00
	Storage	[1 1 0 0 0 0 0 0]	200.000	45.00
			233.613	52.00
DE	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[1 0 0 0 0 0 0 0]	480.000	130.00
	Network	[1 1 0 1 0 0 0 0]	120.000	27.00
			140.168	31.00
			173.914	39.00
	Storage	[2 1 0 0 0 0 0 0]	200.000	45.00
			233.613	52.00
PSO	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[0 1 0 0 0 0 0 0]	480.000	130.00
	Network	[3 0 0 0 0 0 0 0]	120.000	27.00
	Storage	[3 0 0 0 0 0 0 0]	200.000	45.00

Table 17 – Solutions found by DP, DE and PSO for the experiment 4 of the variability scenario. Considering that there are 8 sub-types in this experiment, the Solution column presents the solution vector with the respective amount of components selected per sub-type.

Variability | Experiment 4				
Algorithm	**Component Type**	**Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**
DP	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[2 0 0 0 0 0 0 0]	480.000	130.00
	Network	[2 0 0 0 0 0 0 0]	120.000	27.00
	Storage	[2 0 0 0 0 0 0 0]	200.000	45.00
DE	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[2 0 0 0 0 0 0 0]	480.000	130.00
	Network	[1 0 1 0 0 0 0 0]	120.000	27.00
			153.790	34.00
	Storage	[1 0 1 0 0 0 0 0]	200.000	45.00
			256.317	57.00
PSO	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[2 0 0 0 0 0 0 0]	480.000	130.00
	Network	[1 0 1 0 0 0 0 0]	120.000	27.00
			153.790	34.00
	Storage	[1 1 0 0 0 0 0 0]	200.000	45.00
			233.613	52.00

4 OPTIMIZING THE INFRASTRUCTURE ALLOCATION IN COMPOSABLE DATA CENTER: MULTI-OBJECTIVE APPROACH

Since maximizing the availability and minimizing the cost are conflicting goals, a traditional mono-objective approach could not express the trade-off properly. Thus, we also present our problem from a multi-objective view to analyze and understand the behavior of different solutions expressed as Pareto front.

In this chapter, we present the formulation of the multi-objective problem definition and its implementation taking in account the two algorithms used in this experiment: Non-dominated Sorting Genetic Algorithm and Generalized Differential Evolution. Furthermore, we present the main results obtained with multi-objective experiments. These results are also divided into two scenarios, previously presented in 3.3.4. We also show the parametrization of the algorithms utilized in this approach and the convergence analysis of them.

4.1 PROBLEM DEFINITION

We define the following multi-objective problem in order to find a set of computing resources from the shared pool of resources in a composable infrastructure that maximizes availability (A) of an application and minimizes budget (B).

$$
\begin{aligned}
\text{maximize} \quad & Y = availability(x_1, ..., x_I, y_1, ..., y_J, \\
& w_1, ..., w_K, z_1, ..., z_L, \\
& mttf\_cpu_1, ..., mttf\_cpu_I, \\
& mttr\_cpu_1, ..., mttr\_cpu_I, \\
& mttf\_mem_1, ..., mttf\_mem_J, \\
& mttr\_mem_1, ..., mttr\_mem_J, \\
& mttf\_net_1, ..., mttf\_net_K, \\
& mttr\_net_1, ..., mttr\_net_J, \\
& mttf\_sto_1, ..., mttf\_sto_L, \\
& mttr\_sto_1, ..., mttr\_sto_L, \\
& R_{cpu}, R_{mem}, R_{net}, R_{sto}),
\end{aligned}
$$

$$
\begin{aligned}
\text{minimize} \quad & W = cost(x_1, ..., x_I, y_1, ..., y_J, w_1, ..., w_K, z_1, ..., z_L, \\
& R_{cpu}, R_{mem}, R_{net}, R_{sto}, \\
& \sum_{i=1}^{I} cost\_cpu_i x_i + \sum_{j=1}^{J} cost\_mem_j y_j + \\
& \sum_{k=1}^{K} cost\_net_k w_k + \sum_{l=1}^{L} cost\_sto_l z_l) \\
\text{subject to} \quad & \sum_{i=1}^{I} cost\_cpu_i x_i + \sum_{j=1}^{J} cost\_mem_j y_j +, \\
& \sum_{k=1}^{K} cost\_net_k w_k + \sum_{l=1}^{L} cost\_sto_l z_l < B, \\
& \sum_{i=1}^{I} x_i \geq R_{cpu}, \sum_{j=1}^{J} y_j \geq R_{mem}, \\
& \sum_{k=1}^{K} w_k \geq R_{net}, \sum_{l=1}^{L} z_l \geq R_{sto}
\end{aligned}
$$

The function *availability*(), that corresponds to the same function presented in 3.1, estimates the availability of a possible solution represented by an RBD model that is parameterized by the decision variables, application requirements, MTTF values, and MTTR values, and solved through numerical algorithms. Depending on the values assumed by the decision variables for a possible solution, a specific RBD model is composed and solved in order to estimate the corresponding availability.

The function *cost*() calculates the cost associated with a given allocation that is parameterized by the components of a possible solution.

The first constraint ensures that the overall cost of the set of resources allocated by the composable infrastructure does not breach the maximum budget (B). The other four constraints (i.e. the terms $R_{cpu} \in N^*$, $R_{mem} \in N^*$, $R_{net} \in N^*$, and $R_{cpu} \in N^*$) are the application requirements defined by the user i.e. the minimum number of *cpu*, *mem*, *net* and *sto* units required by the application.

In this problem formulation, we have considered the same set of parameters presented in Table 1. Furthermore, we also considered the same automatic generation of the availability model presented in section 3.2 in this multi-objective approach.

4.2 ALGORITHMS IMPLEMENTATION

In order to develop the multi-objective approach, we divided the implementation in two fitness functions and a constraint function. This implementation while based on this two

fitness functions is applied for both NSGA-II and GDE3. The NSGA-II and GDE3 implementations considered in this work are part of JMetalPy[1] framework.

The first fitness function is responsible for calculating infrastructure availability for each candidate solution provided by the algorithm through a *K-out-of-N* RBD model taking into account the MTTF and MTTR values of respective components. This first fitness function only considers solutions that adhere to the application requirements.

The second fitness function calculates the cost for each candidate solution based on selected hardware resources, that means, the cost of each hardware resource is considered to estimate the overall cost of the proposed solution. This fitness function retains solutions that adhere to the application requirements.

Finally, the constraint function is responsible for verifying if the cost of the candidate solution meets the available budget.

In the next chapter, we will present the main results obtained for experiments utilizing these algorithms.

4.2.1 Algorithms Parametrization

For NSGA-II, we defined parameters as follows (Table 18). We also defined a fitness function to maximizing availability, a fitness function to minimizing cost and a constraint function. All functions receive as a input a integer vector containing the number of hardware resources for each sub-type. As the original NSGA-II implementation works with real vectors (as well as GDE3 implementation), we needed to convert the representation vector to the integer vector.

Table 18 – NSGA-II parameters

Mutation	Polynomial
Crossover	SBX
Mutation Rate	1/m
Mutation Distribution Index	20
Crossover Rate	1
Crossover Distribution Index	20
Population Size	N = 100
Selection for Reproduction	Binary Tournament
Fitness Evaluation	25,000

The parameters for GDE3 were defined as follows (Table 19). The fitness used and the constraint functions follow the same settings as the ones for the NSGA-II algorithm.

[1] https://github.com/jMetal/jMetalPy

Table 19 – GDE3 parameters

Population Size	N = 100
Crossover Rate	0.5
Step Size	0.5
Fitness evaluation	25,000

The most parameters used for NSGA-II and GDE3 are based on (DURILLO et al., 2010). In this study, the authors performed a parameterization study to analyze the scalability of sets of problems taking into account high numbers of problem variables. For NSGA-II, we only adapted empirically the probability of crossover rate for the higher value. And for GDE3, we adapted the crossover rate.

Each experiment of each scenarios, for both GDE3 and NSGA-II, was executed 10 times.

4.3 MULTI-OBJECTIVE OPTIMIZATION ANALYSIS

In this section, we will present the main results obtained in the Heterogeneity and Variability scenarios by the multi-objective approach in terms of availability, cost and arrangement of solutions, as well as, a provide a discussion about them. Additionally, we will present the convergence analysis results. Taking into account the fact that we performed several executions for each experiment of the evolutionary algorithms, in this section, we discuss the results from an independent execution. The summary from the all executions are present in section 4.3.3.

4.3.1 Heterogeneity scenario results

In Experiment 1 of the heterogeneity scenario (Figures 9 (a),(b)), NSGA-II and GDE3 found the same 5 solutions, with a maximum availability of 99.9963% (annual downtime of 0.3193 hours) at an estimated cost of $610; and minimum availability of 99.9940% (annual downtime of 0.5176 hours) at an estimated cost of $572.

In Experiment 2 (Figures 9 (c),(d)), NSGA-II and GDE3 obtained the same solution for maximum and minimum, with maximum availability of 99.9974% at an estimated cost of $683, and minimum availability of 99.9940% at an estimated cost of $572. Furthermore, the number of solutions increased to 31 for both algorithms.

In Experiment 3 (Figures 9 (e),(f)), NSGA-II and GDE3 found closer solutions. However, GDE3 found more solutions (60) and reached NSGA-II (50). Maximum availability for GDE3 was 99.9977% (annual downtime of 0.1931 hours) at an estimated cost of $738. Maximum availability for NSGA-II was 99.9976% (annual downtime of 0.2086 hours) at an

estimated cost of $739. Both algorithms show the same minimum availability (99.9940%) at the same estimated cost ($572).

Finally, in Experiment 4 (Figures 9 (g),(h)), GDE3 shows a maximum availability of 99.9976% at an estimated cost of $800 while NSGA-II shows a maximum availability of 99.9974% at an estimated cost of $798. In terms of minimum availability, GDE3 reached 99.9940% at an estimated cost of $572; NSGA-II is at 99.9941% at an estimated cost of $593. Overall, GDE3 found 63 solutions while NSGA-II found 62.

From these results, we can note an increase in the number of solutions in line with the configuration of each experiment for NSGA-II and GDE3. In Experiment 1, for example, these algorithms found only 5 solutions. When increasing the number of sub-types in the resource pool (Experiment 2), both NSGA-II and GDE3 more than tripled the number of solutions (31 solutions of each algorithm). However, Experiments 3 and 4 behave differently. From Experiment 2 to Experiment 3, the number of solutions found by NSGA-II increased by 61,29% , while from Experiment 3 to Experiment 4, the number of solutions only increased by 24%. This is understandable as, despite the increase in the number of sub-types, the budget and application requirement constraints limited the algorithms and the search space.

In both Experiments 1 and 2, the algorithms found the same maximum and minimum solutions (see Tables 23 and 24). For Experiment 3, as a minimum solution, GDE3 achieved better results due to selexting network resource cheaper comparing with NSGA-II. While for its maximum solution, GDE3 chose a compute and a network resource that were more expensive. Despite this, the difference between GDE3 and NSGA-II availability and cost are a mere 0,0001% and $1, respectively.

Regarding availability and cost results, we noted that NSGA-II and GDE3 in Experiment 2 resulted in similar availability to NSGA-II in Experiment 4, only differing at the 8th place decimal. Due to budget constraints and the size of the resource pool for the different experiments, the solution presented different number of hardware resources, and consequently different costs. NSGA-II and GDE3 in experiment 2 found a cost of $683, while NSGA-II in Experiment 4 found a cost of $798. While both NSGA-II and GDE3 found a solution composed of 6 hardware resources in experiment 2, NSGA-II in Experiment 4 (see Table 26) found a solution with 8 hardware resources. This suggests it is possible to assemble a set of resources in a composable infrastructure that meets the same requirement using less hardware resources, at a lower cost while delivering effectively the same availability.

4.3.2 Variability scenario results

In Experiment 1 for the variability scenario (Figures 10 (a), (b)), the maximum availability of the two algorithms was close, with 99.9966% for GDE3 (annual downtime of 0,2942 hours) and 99.9964% (annual downtime of 0,3110 hours) for NSGA-II, while the

correspondent cost was $629 for GDE3, and $627 for NSGA-II. The minimum availability of NSGA-II and GDE3 was the same (99.9940% at a cost of $572). In this experiment, NSGA-II and GDE3 found 18 and 23 solutions, respectively.

In Experiment 2 (Figures 10 (c), (d)), NSGA-II and GDE3 found 33 and 44 solutions, respectively. The two algorithms found effectively the same maximum availability of 99.9974% (with a difference at the 5th decimal place), with the same cost, $685, for both algorithms. GDE3 obtained minimum availability of 99.9940% and a cost of $572, while NSGA-II obtained availability of 99.9943% and a cost of $576.

In Experiment 3 (Figures 10 (e), (f)), even though GDE3 and NSGA-II found the same maximum cost, $743, GDE3 achieved the best maximum availability of 99.9976% while NSGA-II reached 99.9968%. Regarding the minimum availability and the respective cost, NSGA-II obtained 99.9947% and $682, while GDE3 kept the same result of previous experiment. In this experiment, NSGA-II and GDE3 listed 21 and 66 solutions, respectively.

Lastly, in Experiment 4 (Figures 10 (g), (h)), GDE3 and NSGA-II obtained the same maximum availability of 99.9979% but GDE3 reached the best cost, $788 against $799 from NSGA-II. The minimum availability of NSGA-II was 99.9945% costing $579, whereas GDE3 kept the result of previous experiment. In this experiment, NSGA-II and GDE3 visited 59 and 63 solutions, respectively.

From these results, we noted that in Experiment 1, NSGA-II and GDE3 found more solutions in variability scenarios than in heterogeneity ones. This can be explained by the size of the resource pool that presents redundant hardware resources. In this experiment, there is some variation in the number of solutions found by the algorithms. For instance, NSGA-II found 33 solutions in Experiment 2, and 21 in Experiment 3.

About solutions of In Experiment 1 (Table 27), we noted that both minimum and maximum solutions of NSGA-II and GDE3 were similar and consequently presented similar availability values. The only difference may be found in the maximum solution of NSGA-II where a different storage hardware resource, $2 more expensive, were chose.

In Experiment 2 (Table 28), both the minimum and maximum solutions of NSGA-II and GDE3 also were similar. The difference may be found in minimum solution of NSGA-II where a different network hardware resource, more expensive, were chosen. However, this change only increased in $4 the cost of solution, delivering similiar availability.

We noted in Experiment 3 (Table 29) that NSGA-II and GDE3 presented the same costs but different levels of availability. Despite GDE3 selecting more resources than NSGA-II, 6 *vs* 5, it selected a compute with better individual availability.

For Experiment 4 (Table 30), there is an interesting observation: GDE3 and NSGA-II obtained the same maximum availability with different costs. The solutions for these algorithms also impacted one of the results. While NSGA-II selected 7 hardware resources, GDE selected 6. Furthermore, NSGA-II selected 2 distinct resources of the same sub-type

in two cases (2 memory and 2 storage), while GDE3 only selected 2 resources of the same sub-type (2 storage). We also noted that GDE3 selected 1 memory resource for redundancy that was more expensive than the memory chosen by NSGA-II. This suggests that it is possible to assemble a less expensive set of resources using composable infrastructure that uses less redundant hardware resources while delivering the same availability of a set of resources with more redundancy.

4.3.3 Convergence Analysis

In order to analyse the performance of the optimization algorithms, we use a widely-used quality metric, the hypervolume (HV) measure. HV measures both convergence and diversity (FIGUEIREDO; LUDERMIR; BASTOS-FILHO, 2016; LI et al., 2015; SINGH; ISAACS; RAY, 2011). When calculating the HV, a reference point is required; in this work, each experiment of each scenario has a different vector that represents the set of reference points. We consider the worst values for availability and cost as reference points. For all cases, we consider the availability of 0%, and cost to allocate the whole resource pool, which varying according each experiment, as shown in Table 20.

Table 20 – Reference point variation

Heterogeneity	
Experiment	**Cost ($)**
1	1238.0
2	2797.0
3	6903.0
4	21670.0
Variability	
Experiment	**Cost**
1	6903.0
2	13806.0
3	20709.0
4	27612.0

We calculated the Hypervolume from 10 executions of each experiment, from both scenarios of NSGA-II and GDE3 algorithms. Following, we calculated the mean and standard deviation obtained from these values. To analyze the different solutions obtained in each experiment for GDE3 and NSGA-II algorithms, we also applied the non-parametric Kruskal-Wallis Test, to verify statistical equality in the models.

In Table 21, we present the summary of executions obtained from Heterogeneity scenario. In Experiment 1, we noted that mean and standard deviation values obtained for both algorithms are the same. It indicates the low variation of the algorithms due to

the limitations of this Experiment. In Experiment 2, despite algorithms obtained a close Hypervolume mean values, the models presented statistical difference with a p-value of 2.760041810417523e-05, indicating a better GDE3 result due to lower standard deviation. In Experiment 3, the models presented statistical difference with a p-value of 0.0001. In this case, NSGA-II obtained a better result due to lower standard deviation value. In Experiment 4, the algorithms obtained statistical equality with a p-value of 0.0796.

Table 21 – Heterogeneity Hypervolume Summary

GDE3 | Heterogeneity		
Experiment	**Hypervolume Mean**	**Standard Deviation**
1	665.9751223166036	0.0
2	2224.942944985391	0.0
3	6330.858600486061	1.186222298097961e-06
4	21097.564225994243	0.019171919886313414
NSGA-II | Heterogeneity		
Experiment	**Hypervolume Mean**	**Standard Deviation**
1	665.9751223166036	0.0
2	2224.9429403595213	9.765314754186369e-06
3	6330.848459293013	0.0009318107085997483
4	21090.95470461667	19.83241190715805

In Table 22, we present the summary of executions obtained from Variability scenario. In this scenario, all models presented statistical difference to each other. In Experiment 1, GDE3 obtained better result than NSGA-II, with lower standard deviation and slightly better Hypervolume Mean. The models obtained a p-value of 4.6300028968098476e-05. In Experiment 2, 3 and 4 GDE3 also presented better Hypervolume mean and standard deviation values. The respective p-values obtained from these Experiments were 0.0207, 7.112318931215443e-05 and 0.0001.

4.3.4 Comparing the approaches

The comparison discussed in this section does not aim to establish which approach obtained the best results but aim to present the similarities between approaches and expose the behavior of the conflicting objectives.

Despite using different approaches, we noted that the solutions reached in both cases for availability were very much close. This fact is justified by the budget constraint established in both problem definitions. It is essential to confirm that the comparison performed

Table 22 – Variability Hypervolume Summary

GDE3 | Variability		
Experiment	**Hypervolume Mean**	**Standard Deviation**
1	6330.786654069427	0.0
2	13233.670947325094	0.0025387774806024505
3	20136.522549472644	3.867272607749328e-06
4	27039.454562612726	2.3493078700874407e-05
NSGA-II | Variability		
Experiment	**Hypervolume Mean**	**Standard Deviation**
1	6330.777469081672	0.003472253838280767
2	13231.671057239777	2.001909688137509
3	20087.684765386974	33.9353652341439
4	26980.411055904493	49.97757984984292

in this section is based on decision criteria for maximizing availability and minimizing cost. Additionally, we divided this section in Heterogeneity and Variability Scenarios.

4.3.4.1 Heterogeneity Scenario

Analyzing Experiment 1 of the Heterogeneity scenario, we noted that DP and DE also reached one of the 5 results found in multi-objective approach. But the multi-objective approach achieved slightly higher availability in its maximum solution with both GDE3 and NSGA-II algorithms, with availability of 99.9963% and cost of $610. While mono-objective approach found (in DP and DE algorithms) availability of 99.9961% and cost of $603. The main difference is specified by downtime: 0,3351 for multi-objective approach and 0,3193 for mono-objective approach.

In Experiment 2 of Heterogeneity scenario, both approaches obtained the same maximum availability for all algorithms. However, in the multi-objective approach, it a total of 31 solutions were identified by GDE3 and NSGA-II, respectively.

In Experiment 3, the multi-objective approach found the best solution selected by the mono-objective approach (99.9974% and cost of $655). The same best availability reached in the mono-objective was also found among the multi-objective approach solutions with higher costs. Multi-objective approach found maximum availability solution with a slightly higher availability value (availability of 99.9977% and cost of $738).

Finally, in Experiment 4, the mono-objective approach delivered an availability value (availability of 99.9977% and cost of $757 found by DP) closer to that of the multi-

objective approach (availability of 99.9976% and cost of $800). In this case, the mono-objective obtained a lower cost.

In this scenario, it was expecting an increase in the availability across experiments due to the rising number of items associated with the budget increase per experiment. However, in both approaches, this increase is not linear, mainly in multi-objective approach, from experiment 3 to 4.

4.3.4.2 Variability Scenario

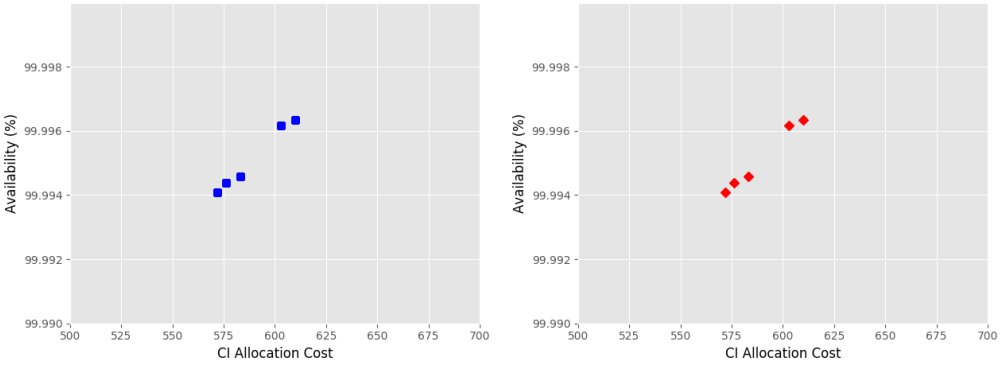
Analyzing Experiment 1 of Variability scenario, we see a similar behavior compared to the Heterogeneity scenario for the same experiment. Also in this case, the best result obtained by mono-objective approach was among the solutions given by the multi-objective approach. In mono-objective approach, DP found the best solution among the mono-objective algorithms: 99.9961% and cost of $603. This result is closer to results found in multi-objective approach: GDE3 found availability of 99.9966 % and cost of $629; and NSGA-II found availability of 99.9964% and cost of $627.

In Experiment 2 of Variability, the same maximum result found in Experiment 2 of Heterogeneity, for both approaches, is reached by its respective algorithms. It means that there is no variation in this experiment, even when scenarios are different. We believe that budget constraint associated to this experiment did not allow a greater variety of availability even when the variability of resource pool is better.

Multi-objective and mono-objective approaches presented a closer availability result in Experiment 3. For example, GDE3 found availability of 99.9976% and cost of $743 against availability of 99.9974% and cost of $655 found by DP. In this case, DP presented a lower cost due slightly lower availability. We also noted that the adopted algorithms within the multi-objective approach identified the same maximum result as the one for the mono-objective approach among their set of solutions and also gave results with the same maximum availability, but with lower cost.

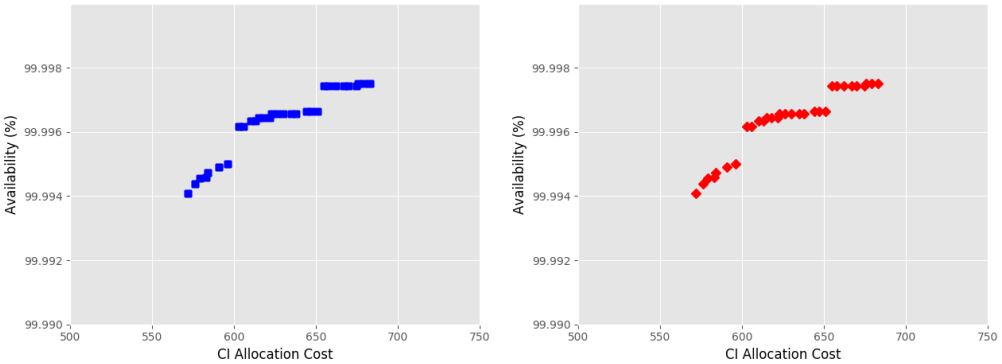
In Experiment 4, both approaches pointed to the same maximum availability value. In this case, the mono-objective approach achieved a cost of $774 with DP and the multi-objective achieve a cost of $799 for NSGA-II and $788 for GDE3. The cost suffered by the multi-objective approach also was similar to the one for the mono-objective approach when running PSO.

In this scenario, it was also expecting an increase in the availability across experiments due to the rising number of items associated with budget increase per experiment; and due to variability of the number sub-types utilized. However, in the multi-objective approach, for example, the availability suffers more variations across experiments than in the Heterogeneity scenario. In the Variability scenario, we also noted the availability values it was higher than the Heterogeneity one.



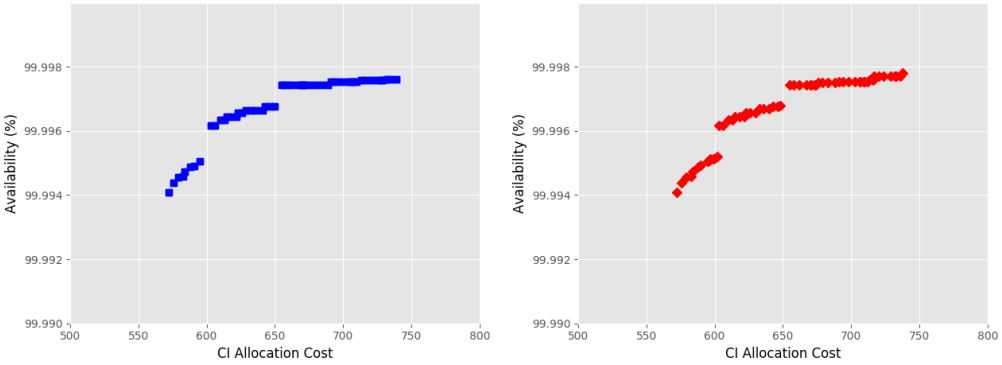
(a) NSGA-II - Experiment 1

(b) GDE3 - Experiment 1



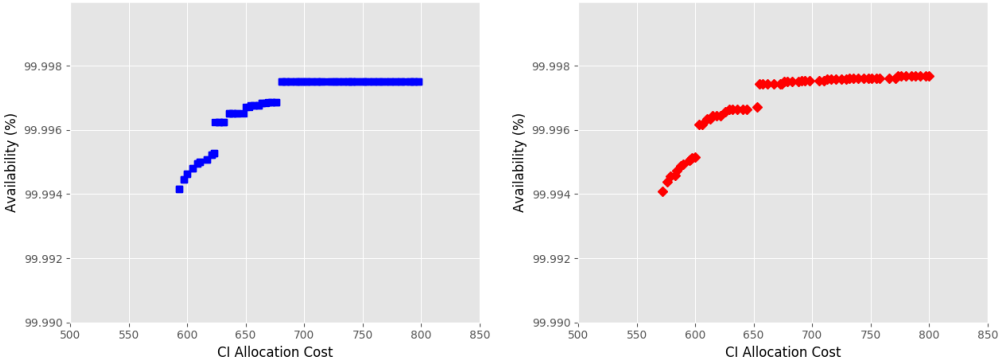
(c) NSGA-II - Experiment 2

(d) GDE3 - Experiment 2



(e) NSGA-II - Experiment 3

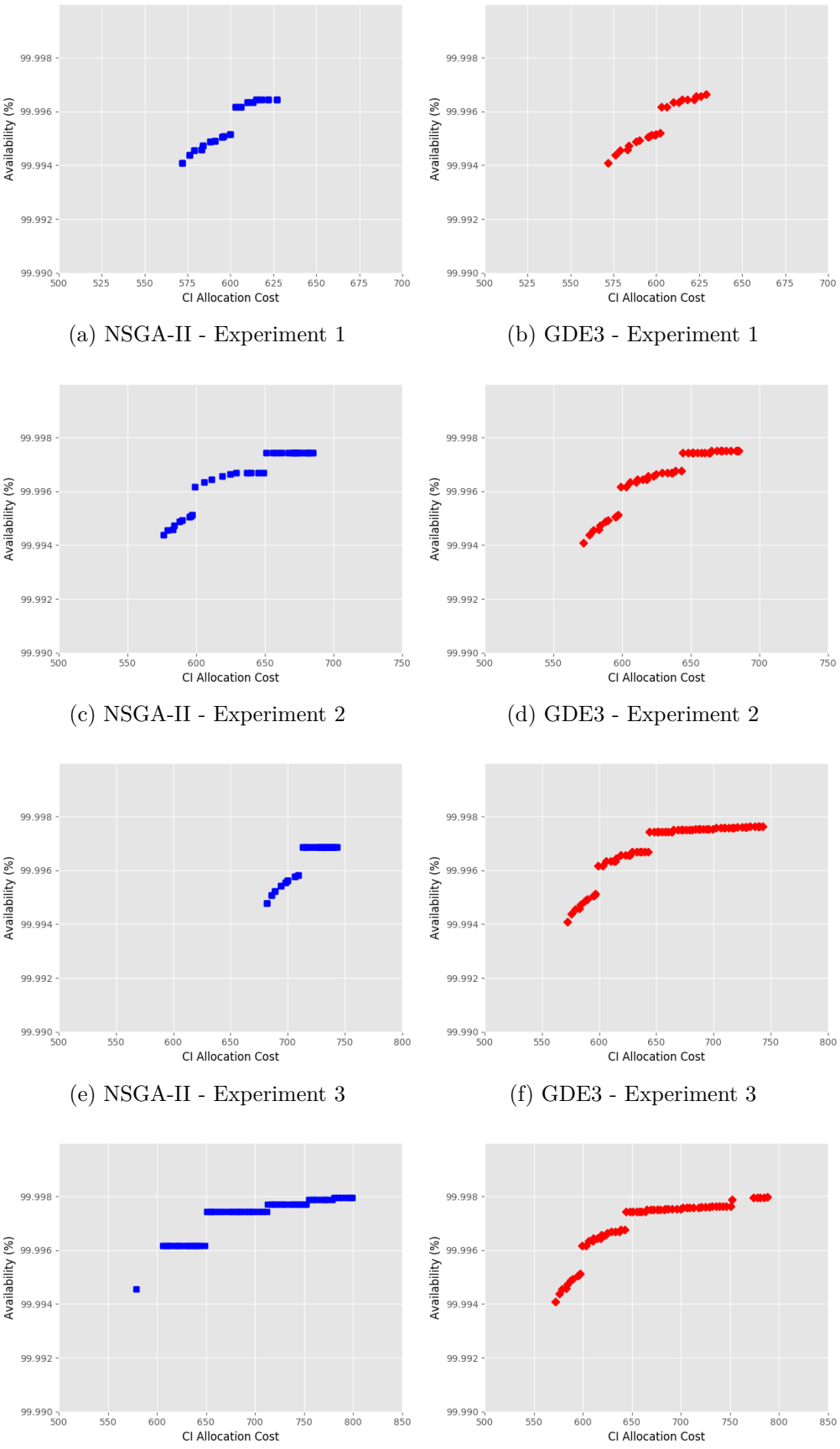
(f) GDE3 - Experiment 3



(g) NSGA-II - Experiment 4

(h) GDE3 - Experiment 4

Figure 9 – Results of the heterogeneity scenarios using NSGA-II and GDE3 algorithms of Experiment 1, Experiment 2, Experiment 3, and Experiment 4.



(a) NSGA-II - Experiment 1

(b) GDE3 - Experiment 1

(c) NSGA-II - Experiment 2

(d) GDE3 - Experiment 2

(e) NSGA-II - Experiment 3

(f) GDE3 - Experiment 3

(g) NSGA-II - Experiment 4

(h) GDE3 - Experiment 4

Figure 10 – Results of the variability scenarios using NSGA-II and GDE3 algorithms of the experiment 1, experiment 2, experiment 3, and experiment 4.

Table 23 – Solutions found by NSGA-II and GDE3 for Experiment 1 of the heterogeneity scenario. As there are 8 sub-types in this experiment, the maximum and minimum availability columns present the solution vector with the respective amount of hardware resources selected per sub-type.

Heterogeneity | Experiment 1							
Algorithm	**Component Type**	**Mininum Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**	**Maximum Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**
NSGA-II	Compute	[1 0]	292.000	370.00	[1 0]	292.000	370.00
	Memory	[1 0]	480.000	130.00	[1 0]	480.000	130.00
	Network	[1 0]	120.000	27.00	[1 1]	120.000	27.00
						140.168	31.00
	Storage	[1 0]	200.000	45.00	[1 0]	200.000	45.00
GDE3	Compute	[1 0]	292.000	370.00	[1 0]	292.000	370.00
	Memory	[1 0]	480.000	130.00	[1 0]	480.000	130.00
	Network	[1 0]	120.000	27.00	[1 1]	120.000	27.00
						140.168	31.00
	Storage	[1 0]	200.000	45.00	[1 0]	200.000	45.00

Table 24 – Solutions found by NSGA-II and GDE3 for Experiment 2 of the heterogeneity scenario. As there are 8 sub-types in this experiment, the maximum and minimum availability columns present the solution vector with the respective amount of hardware resources selected per sub-type.

Heterogeneity | Experiment 2							
Algorithm	**Component Type**	**Mininum Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**	**Maximum Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**
NSGA-II	Compute	[1 0 0 0]	292.000	370.00	[1 0 0 0]	292.000	370.00
	Memory	[1 0 0 0]	480.000	130.00	[0 1 0 0]	560.672	151.00
	Network	[1 0 0 0]	120.000	27.00	[0 1 1 0]	140.168	31.00
						153.790	34.00
	Storage	[1 0 0 0]	200.000	45.00	[1 1 0 0]	200.000	45.00
						233.613	52.00
GDE3	Compute	[1 0 0 0]	292.000	370.00	[1 0 0 0]	292.000	370.00
	Memory	[1 0 0 0]	480.000	130.00	[0 1 0 0]	560.672	151.00
	Network	[1 0 0 0]	120.000	27.00	[0 1 1 0]	140.168	31.00
						153.790	34.00
	Storage	[1 0 0 0]	200.000	45.00	[1 1 0 0]	200.000	45.00
						233.613	52.00

Table 25 – Solutions found by NSGA-II and GDE3 for Experiment 3 of the heterogeneity scenario. As there are 8 sub-types in this experiment, the maximum and minimum availability columns present the solution vector with the respective amount of hardware resources selected per sub-type.

Heterogeneity | Experiment 3							
Algorithm	**Component Type**	**Mininum Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**	**Maximum Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**
NSGA-II	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[1 0 0 0 0 0 0 0]	480.000	130.00	[0 0 0 0 1 0 0 0]	765.430	207.00
	Network	[0 1 0 0 0 0 0 0]	140.168	31.00	[0 1 1 0 0 0 0 0]	140.168	31.00
						153.790	34.00
	Storage	[1 0 0 0 0 0 0 0]	200.000	45.00	[1 1 0 0 0 0 0 0]	200.000	45.00
						233.613	52.00
GDE3	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00	[0 1 0 0 0 0 0 0]	341.075	432.00
	Memory	[1 0 0 0 0 0 0 0]	480.000	130.00	[0 1 0 0 0 0 0 0]	560.672	151.00
	Network	[1 0 0 0 0 0 0 0]	120.000	27.00	[1 1 0 0 0 0 0 0]	120.000	27.00
						140.168	31.00
	Storage	[1 0 0 0 0 0 0 0]	200.000	45.00	[1 1 0 0 0 0 0 0]	200.000	45.00
						233.613	52.00

Table 26 – Solutions found by NSGA-II and GDE3 for Experiment 4 of the heterogeneity scenario. As there are 8 sub-types in this experiment, the maximum and minimum availability columns present the solution vector with the respective amount of hardware resources selected per sub-type.

Heterogeneity | Experiment 4							
Algorithm	**Component Type**	**Mininum Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**	**Maximum Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**
NSGA-II	Compute	[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	292.000	370.00	[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	560.672	151.00	[0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	560.672	151.00
	Network	[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	120.000	27.00	[0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0]	140.168	31.00
						153.790	34.00
						173.914	39.00
	Storage	[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	200.000	45.00	[1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0]	200.000	45.00
						256.317	57.00
						318.929	71.00
GDE3	Compute	[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	292.000	370.00	[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	480.000	130.00	[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]	960.364	260.00
	Network	[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	120.000	27.00	[0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0]	153.790	34.00
						173.914	39.00
	Storage	[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	200.000	45.00	[1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	200.000	45.00
						233.613	52.00

Table 27 – Solutions found by NSGA-II and GDE3 for Experiment 1 of the variability scenario. As there are 8 sub-types in this experiment, the maximum and minimum availability columns present the solution vector with the respective amount of hardware resources selected per sub-type.

Variability | Experiment 1							
Algorithm	**Component Type**	**Mininum Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**	**Maximum Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**
NSGA-II	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[1 0 0 0 0 0 0 0]	480.000	130.00	[1 0 0 0 0 0 0 0]	480.000	130.00
	Network	[1 0 0 0 0 0 0 0]	120.000	27.00	[0 1 0 1 0 0 0 0]	140.168	31.00
						173.914	39.00
	Storage	[1 0 0 0 0 0 0 0]	200.000	45.00	[0 0 1 0 0 0 0 0]	256.317	57.00
GDE3	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[1 0 0 0 0 0 0 0]	480.000	130.00	[1 0 0 0 0 0 0 0]	480.000	130.00
	Network	[1 0 0 0 0 0 0 0]	120.000	27.00	[1 1 0 0 0 0 0 0]	120.000	27.00
						140.168	31.00
	Storage	[1 0 0 0 0 0 0 0]	200.000	45.00	[0 0 0 0 1 0 0 0]	318.929	71.00

Table 28 – Solutions found by NSGA-II and GDE3 for Experiment 2 of the variability scenario. As there are 8 sub-types in this experiment, the maximum and minimum availability columns present the solution vector with the respective amount of hardware resources selected per sub-type.

Variability | Experiment 2							
Algorithm	**Component Type**	**Mininum Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**	**Maximum Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**
NSGA-II	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[1 0 0 0 0 0 0 0]	480.000	130.00	[1 0 0 0 0 0 0 0]	480.000	130.00
	Network	[0 1 0 0 0 0 0 0]	140.168	31.00	[2 0 1 0 0 0 0 0]	120.000	27.00
						153.790	34.00
	Storage	[1 0 0 0 0 0 0 0]	200.000	45.00	[1 1 0 0 0 0 0 0]	200.000	45.00
						233.613	53.00
GDE3	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[1 0 0 0 0 0 0 0]	480.000	130.00	[0 1 0 0 0 0 0 0]	560.672	151.00
	Network	[1 0 0 0 0 0 0 0]	120.000	27.00	[0 1 0 0 1 0 0 0]	140.168	31.00
						191.357	43.00
	Storage	[1 0 0 0 0 0 0 0]	200.000	45.00	[2 0 0 0 0 0 0 0]	200.000	45.00

Table 29 – Solutions found by NSGA-II and GDE3 for Experiment 3 of the variability scenario. As there are 8 sub-types in this experiment, the maximum and minimum availability columns present the solution vector with the respective amount of hardware resources selected per sub-type.

Variability | Experiment 3							
Algorithm	**Component Type**	**Mininum Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**	**Maximum Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**
NSGA-II	Compute	[0 1 0 0 0 0 0 0]	341.075	432.00	[0 1 0 0 0 0 0 0]	341.075	432.00
	Memory	[0 0 1 0 0 0 0 0]	615.161	166.00	[0 0 1 0 0 0 0 0]	615.161	166.00
	Network	[1 0 0 0 0 0 0 0]	120.000	27.00	[0 0 0 0 1 1 0 0]	191.357	43.00
						202.104	45.00
	Storage	[0 0 1 0 0 0 0 0]	256.317	57.00	[0 0 1 0 0 0 0 0]	256.317	57.00
GDE3	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[1 0 0 0 0 0 0 0]	480.000	130.00	[0 0 0 0 0 1 0 0]	808.416	218.00
	Network	[1 0 0 0 0 0 0 0]	120.000	27.00	[0 1 1 0 0 0 0 0]	140.168	31.00
						153.790	34.00
	Storage	[1 0 0 0 0 0 0 0]	200.000	45.00	[2 0 0 0 0 0 0]	200.000	45.00

Table 30 – Solutions found by NSGA-II and GDE3 for Experiment 4 of the variability scenario. As there are 8 sub-types in this experiment, the maximum and minimum availability columns present the solution vector with the respective amount of hardware resources selected per sub-type.

Variability | Experiment 4							
Algorithm	**Component Type**	**Mininum Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**	**Maximum Solution**	**MTTF (In Hours)**	**Allocated Hardware Cost ($)**
NSGA-II	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00	[1 0 0 0 0 0 0 0]	292.000	370.00
	Memory	[1 0 0 0 0 0 0 0]	480.000	130.00	[2 0 0 0 0 0 0 0]	480.000	130.00
	Network	[0 0 1 0 0 0 0 0]	153.790	34.00	[0 0 1 0 0 1 0 0]	153.790	34.00
						202.104	45.00
	Storage	[1 0 0 0 0 0 0 0]	200.000	45.00	[2 0 0 0 0 0 0]	200.000	45.00
GDE3	Compute	[1 0 0 0 0 0 0 0]	292.000	370.00	[0 0 1 0 0 0 0 0]	374.223	474.00
	Memory	[1 0 0 0 0 0 0 0]	480.000	130.00	[0 0 1 0 0 0 0 0]	615.161	166.00
	Network	[1 0 0 0 0 0 0 0]	120.000	27.00	[1 1 0 0 0 0 0 0]	120.000	27.00
						140.168	31.00
	Storage	[1 0 0 0 0 0 0 0]	200.000	45.00	[2 0 0 0 0 0 0]	200.000	45.00

5 CONCLUSION

Over recent years, cloud computing consolidated itself as a powerful paradigm for delivering IT resources. Nevertheless, this paradigm faces technology-driven inefficiencies introduced mainly by resources underutilization and overprovisioning for redundancy. Consequently, cloud providers face new issues and challenges to continue offering highly available services, requiring changes in the data center infrastructure, management, and operation.

Composable data center addresses sources of inefficiencies in traditional cloud computing and also provides a solution for managing the differing infrastructure requirements of legacy applications and next generation cloud-native applications. Notwithstanding this potential and increasing industry support, composable infrastructure and so-called "infrastructure-as-code" is still at an early stage of adoption. As such, it presents a range of challenges that require industry hardened solutions not least resource discovery, allocation, provisioning, orchestration, optimization and remediation.

In this dissertation, we proposed an optimization problem to solve resource allocation in composable data center infrastructure. We used two approaches to solve this problem: mono-objective and multi-objective. In mono-objective approach, we used DP, DE and PSO algorithms and we used NSGA-II and GDE3 in multi-objective approach. These algorithms were utilized in their respective approaches to identify solutions that maximize application availability and minimize infrastructure cost while taking in to account minimum application infrastructure requirements and budget constraints.

In mono-objective approach, DE and PSO models demonstrated statistical equality in most of experiments. Analyzing an individual execution, the DP algorithm found better results, providing highest availability level with lowest cost in all scenarios proposed and within the parameters setting. In this approach, we also performed importance analysis of the computational components taking into account the availability and the costs of the components. The results showed that the network had a large impact on availability with a relatively little cost, while the CPU had less availability impact with a relatively high cost, with the difference of 20.18% in the impact each other.

In multi-objective approach, both NSGA-II and GDE3 were effective in finding solutions for all scenarios proposed and within the parameters setting. While both algorithms presented diversity of solutions, GDE3 presented the largest number of solutions in all scenarios. In this approach, we also performed convergence analysis of the algorithms for 10 executions of each experiment of each scenario. In this analysis, GDE3 presented slightly advantage over NSGA-II in Variability scenario. In Heterogeneity scenario, the models presented similarities.

Our results suggest that optimization algorithms are feasible solutions for identifying

a range of solutions for assembling solutions shared resource pools in composable infrastructure and thereby meeting the technical and commercial requirements of data center managers.

About some learn lessons obtained in this work, we can cite: (a) mono-objective and multi-objective approaches may provide closer results, varying according to a different selection of hardware resources, delivering satisfactory availability values with varying options of cost; (b) additionally, multi-objective algorithms may found mono-objective best solutions in Pareto-front; (c) cheaper resources may produce right resources arrangements, delivering grate results; and (d) multi-objective algorithms may provide a some good composable infrastructure options to the cloud's stakeholders.

Among the main contributions presenting in this work, we can cited:

- Optimization problem models to solve resource allocation in composable data center infrastructure and understand how different arrangements of components impact in the whole system.
- Optimization problem models to estimate composable data center infrastructure availability regarding budget constraints and requirements of application.
- Importance analysis of the computational components in order to understand the impact of them in availability of composable data center.
- To deliver feasible solutions for both cloud data center vendors, which looking for optimizing the utilization of their resources and cloud data center clients' for obtaining availability for their services attached to minimum cost.

5.1 DIFFICULTIES AND LIMITATIONS

This work faced some difficulties during execution. In mono-objective approach, all experiments were executed using R libraries. These experiments presented a long runtime, increasing from a scenario to another. Some mono-objective algorithms presented runtime even worst, such in PSO and Ant Lion Optimization cases. The latter could not be used in our experiments.

In multi-objective approach, we used a Python library called JMetal. Despite present a huge variability of algorithms, we performed experiments with several algorithms which did not presented satisfied results. In this approach, we also faced a long runtime in larger scenarios, as in Experiment 4 from Heterogeneity scenario. Thus, we could not to scale out ours scenarios and experiments.

Finally, in this dissertation, we performed a theoretical work about composable data center infrastructure. The MTTF and MTTR values related to composable data center were obtained from literature because such technology is still at an early stage.

5.2 PUBLICATIONS

In Table 31, we present the scientific papers related to this dissertation. The works 5 and 8 are directly connected to this dissertation. The other works are part of the research project about composable data center.

Table 31 – Scientific papers produced

#	Reference	Type	Status	Qualis
1	Rosendo, D., Gomes, D., Santos, G. L., Goncalves, G., Moreira, A., **da Silva, L. G. F.**, Endo, P. T., Kelner, J., Sadok, D., Mehta, A.& Wildeman, M. (2019). A methodology to assess the availability of next-generation data centers. The Journal of Supercomputing, 1-25.	Journal	Published	B1
2	Santos, G. L., Rosendo, D., Gomes, D., **da Silva, L. G. F.**, Moreira, A., Sadok, D., Kelner, J., Goncalves, G., Wildeman, M.& Endo, P. T. (2019, March). A Methodology for Automating the Cloud Data Center Availability Assessment. In International Conference on Advanced Information Networking and Applications (pp. 1011-1023). Springer, Cham.	Conference	Published	A2
3	Goncalves, G., Rosendo, D., **da Silva, L. G. F.**, Santos, G. L., Gomes, D., Moreira, A, Kelner, J., Sadok, D., Wildeman, M. & Endo, P. T. "A Standard to Rule Them All: Redfish". IEEE Communications Standards Magazine, vol. 3, no. 2, pp. 36-43, June 2019.	Journal	Published	-
4	Moreira, A., Rosendo, D., Gomes, D., Leoni Santos, G., **da Silva, L. G. F.**, Cani, C., Kelner, J., Sadok, D., Goncalves, G., Mehta, A., Wildeman, M. & Endo, P. T. DCAV: A software system to evaluate next-generation cloud data center availability through a friendly graphical interface. Software: Practice and Experience.	Journal	Published	A2
5	**da Silva, L. G. F.**, Goncalves, G., Rosendo, D., Santos, G. L., Moreira, A., Kelner, J., Sadok, D., Wildeman, M., Mehta, A. & Endo, P. T. Maximizing the Availability of Composable Systems of Next-Generation Data Centers. IEEE International Conference on Systems, Man, and Cybernetics (IEEE SMC 2019).	Conference	Published	A2
6	Rosendo, D., Gomes, D., Santos, G. L., **da Silva, L. G. F.**., Moreira, A., Kelner, J., Sadok, D., Goncalves, G., Mehta, A., Wildeman, M. & Endo, P. T. Availability Analysis of Design Configurations to Compose vPOD Systems in Next-Generation Cloud Data Center. Journal of Software: Practice and Experience (JSPE).	Journal	Published	A2
7	**da Silva, L. G. F.**, Endo, P. T., Rosendo, D., Santos, G. L., Santos, D., Moreira, A., Gonçalves, G., Mehta, A. & Wildeman, M. Standardization Efforts for Traditional Data Center Infrastructure Management: the Big Picture. IEEE Engineering Management Review.	Journal	Published	-
8	**da Silva, L. G. F.**, Rocha, E. S., Monteiro, K. H. C., Santos, G. L., Silva, F. A., Kelner, J., Sadok, D., Bastos Filho, C. J. A., Rosati, P., Lynn, T. & Endo, P. T. Optimizing resource availability in composable datacenter infrastructures. Wafers Latin America Symposium on Dependable Computing (LADC)	Conference	Published	B4

5.3 FUTURE WORKS

As future works, we intend to perform a parameter analysis of the algorithms in order to choose the best parameters to be utilized in experiments. We also plan to analyze the composable infrastructure allocation under cloud provider perspective, considering the cost to maintain the infrastructure to serve the clients.

As composable infrastructure supports hardware and software heterogeneity, we intent to explore this in future studies by expanding the number of resource sub-types and the granularity of resource characterization used in our models but also greater variation in application requirements. Furthermore, we intend to compare the performance of different algorithms for resource allocation problems in composable infrastructure.

REFERENCES

ADEKANMBI, O.; GREEN, P. Conceptual comparison of population based metaheuristics for engineering problems. *The Scientific World Journal*, Hindawi, v. 2015, 2015.

AJIBOLA, O. O.; EL-GORASHI, T. E.; ELMIRGHANI, J. M. On energy efficiency of networks for composable datacentre infrastructures. In: IEEE. *2018 20th International Conference on Transparent Optical Networks (ICTON)*. [S.l.], 2018. p. 1–5.

ARAUJO, J.; MACIEL, P.; TORQUATO, M.; CALLOU, G.; ANDRADE, E. Availability evaluation of digital library cloud services. In: IEEE. *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*. [S.l.], 2014. p. 666–671.

BARROSO, L. A.; HÖLZLE, U. The case for energy-proportional computing. 2007.

BELLMAN, R. *Dynamic programming*. [S.l.]: Courier Corporation, 2013.

BOUACHA, K.; TERRAB, A. Hard turning behavior improvement using nsga-ii and pso-nn hybrid model. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 86, n. 9-12, p. 3527–3546, 2016.

BOYER, V.; BAZ, D. E.; ELKIHEL, M. Solving knapsack problems on gpu. *Computers & Operations Research*, Elsevier, v. 39, n. 1, p. 42–47, 2012.

BROSCH, F.; KOZIOLEK, H.; BUHNOVA, B.; REUSSNER, R. Parameterized reliability prediction for component-based software architectures. In: SPRINGER. *International Conference on the Quality of Software Architectures*. [S.l.], 2010. p. 36–51.

CHENG, J.; GRINNEMO, K.-J. *Telco Distributed DC with Transport Protocol Enhancement for 5G Mobile Networks: A Survey*. [S.l.: s.n.], 2017.

CHUNG, I.; ABALI, B.; CRUMLEY, P. et al. Towards a composable computer system. In: ACM. *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*. [S.l.], 2018. p. 137–147.

CRAGO, S. P.; WALTERS, J. P. Heterogeneous cloud computing: The way forward. *Computer*, IEEE, v. 48, n. 1, p. 59–61, 2015.

CREMENE, M.; SUCIU, M.; PALLEZ, D.; DUMITRESCU, D. Comparative analysis of multi-objective evolutionary algorithms for qos-aware web service composition. *Applied Soft Computing*, Elsevier, v. 39, p. 124–139, 2016.

DAS, S.; MULLICK, S. S.; SUGANTHAN, P. N. Recent advances in differential evolution–an updated survey. *Swarm and Evolutionary Computation*, Elsevier, v. 27, p. 1–30, 2016.

DAVIS, W.; BAKER, L.; SPAJIĆ, S. *Time Notes: A Treasury of the Best Time Management Ideas*. FriesenPress, 2015. ISBN 9781460259160. Available at: <https://books.google.com.br/books?id=xppQCgAAQBAJ>.

DRAA, A.; BOUZOUBIA, S.; BOUKHALFA, I. A sinusoidal differential evolution algorithm for numerical optimisation. *Applied Soft Computing*, Elsevier, v. 27, p. 99–126, 2015.

DU, K.-L.; SWAMY, M. Particle swarm optimization. In: *Search and optimization by metaheuristics*. [S.l.]: Springer, 2016. p. 153–173.

DU, W.-B.; GAO, Y.; LIU, C.; ZHENG, Z.; WANG, Z. Adequate is better: particle swarm optimization with limited-information. *Applied Mathematics and Computation*, Elsevier, v. 268, p. 832–838, 2015.

DUMITRESCU, C.; PLESCA, A.; ADAM, M.; NITUCA, C.; DRAGOMIR, A. Methods for reducing energy consumption, optimization in operational data centers. In: IEEE. *2018 International Conference and Exposition on Electrical And Power Engineering (EPE)*. [S.l.], 2018. p. 0483–0486.

DURILLO, J. J.; NEBRO, A. J.; COELLO, C. A. C.; GARCÍA-NIETO, J.; LUNA, F.; ALBA, E. A study of multiobjective metaheuristics when solving parameter scalable problems. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 14, n. 4, p. 618–635, 2010.

EIBEN, A. E.; SMITH, J. E. et al. *Introduction to evolutionary computing*. [S.l.]: Springer, 2003.

ESFE, M. H.; HAJMOHAMMAD, H.; MORADI, R.; ARANI, A. A. A. Multi-objective optimization of cost and thermal performance of double walled carbon nanotubes/water nanofluids by nsga-ii using response surface method. *Applied Thermal Engineering*, Elsevier, v. 112, p. 1648–1657, 2017.

FARIAS, G.; BRASILEIRO, F.; LOPES, R.; CARVALHO, M.; MORAIS, F.; TURULL, D. On the efficiency gains of using disaggregated hardware to build warehouse-scale clusters. In: IEEE. *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. [S.l.], 2017. p. 239–246.

FAWZY, V. D.; SANGADJI, S.; AS'AD, S. Integer 1/0 knapsack problem dynamic programming approach in building maintenance optimization. In: *International Journal of Science and Applied Science: Conference Series*. [S.l.: s.n.], 2017. v. 2, n. 1, p. 429–439.

FAZLOLLAHTABAR, H.; NIAKI, S. T. A. Integration of fault tree analysis, reliability block diagram and hazard decision tree for industrial robot reliability evaluation. *Industrial Robot: An International Journal*, Emerald Publishing Limited, v. 44, n. 6, p. 754–764, 2017.

FICCO, M.; ESPOSITO, C.; PALMIERI, F.; CASTIGLIONE, A. A coral-reefs and game theory-based approach for optimizing elastic cloud resource allocation. *Future Generation Computer Systems*, Elsevier, v. 78, p. 343–352, 2018.

FIGUEIREDO, E. M.; LUDERMIR, T. B.; BASTOS-FILHO, C. J. Many objective particle swarm optimization. *Information Sciences*, Elsevier, v. 374, p. 115–134, 2016.

FIGUEIRÊDO, J.; MACIEL, P.; CALLOU, G.; TAVARES, E.; SOUSA, E.; SILVA, B. Estimating reliability importance and total cost of acquisition for data center power infrastructures. In: IEEE. *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*. [S.l.], 2011. p. 421–426.

GAI, K.; QIU, M.; ZHAO, H. Cost-aware multimedia data allocation for heterogeneous memory using genetic algorithm in cloud computing. *IEEE transactions on cloud computing*, IEEE, 2016.

GHAMISI, P.; BENEDIKTSSON, J. A. Feature selection based on hybridization of genetic algorithm and particle swarm optimization. *IEEE Geoscience and Remote Sensing Letters*, IEEE, v. 12, n. 2, p. 309–313, 2015.

GHAMISI, P.; COUCEIRO, M. S.; MARTINS, F. M.; BENEDIKTSSON, J. A. Multilevel image segmentation based on fractional-order darwinian particle swarm optimization. *IEEE Transactions on Geoscience and Remote sensing*, IEEE, v. 52, n. 5, p. 2382–2394, 2014.

GOLDBERG, D. E.; HOLLAND, J. H. Genetic algorithms and machine learning. *Machine learning*, Springer, v. 3, n. 2, p. 95–99, 1988.

GUIMARAES, A. P.; OLIVEIRA, H. M. N.; BARROS, R.; MACIEL, P. R. Availability analysis of redundant computer networks: A strategy based on reliability importance. In: IEEE. *ICCSN*. [S.l.], 2011. p. 328–332.

GUNAWI, H. S.; HAO, M.; SUMINTO, R. O.; LAKSONO, A.; SATRIA, A. D.; ADITYATAMA, J.; ELIAZAR, K. J. Why does the cloud stop computing?: Lessons from hundreds of service outages. In: ACM. *Proceedings of the Seventh ACM Symposium on Cloud Computing*. [S.l.], 2016. p. 1–16.

GURLER, S.; BAIRAMOV, I. Parallel and k-out-of-n: G systems with nonidentical components and their mean residual life functions. *Applied mathematical modelling*, Elsevier, v. 33, n. 2, p. 1116–1125, 2009.

IDC. *Infrastructure Usage and Overprovisioning Trends Survey*. [S.l.]: IDC, 2016. <https://comport.com/resources-article/5-key-benefits-of-composable-infrastructure/>.

IDG. *2018 Cloud Computing Survey*. [S.l.]: IDG, 2018. <https://resources.idg.com/download/executive-summary/cloud-computing-2018>.

JESUS, D. A. R. de; RIVERA, W. Application of evolutionary algorithms for load forecasting in smart grids. In: *Proceedings of the International Conference on Foundations of Computer Science (FCS)*. [S.l.: s.n.], 2018. p. 40–44.

KAMJOO, A.; MAHERI, A.; DIZQAH, A. M.; PUTRUS, G. A. Multi-objective design under uncertainties of hybrid renewable energy system using nsga-ii and chance constrained programming. *International Journal of Electrical Power & Energy Systems*, Elsevier, v. 74, p. 187–194, 2016.

KATRINIS, K.; SYRIVELIS, D.; PNEVMATIKATOS, D.; ZERVAS, G.; THEODOROPOULOS, D.; KOUTSOPOULOS, I.; HASHARONI, K.; RAHO, D.; PINTO, C.; ESPINA, F. et al. Rack-scale disaggregated cloud data centers: The dredbox project vision. In: EDA CONSORTIUM. *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*. [S.l.], 2016. p. 690–695.

KELLERER, H.; PFERSCHY, U.; PISINGER, D. *Knapsack Problems*. [S.l.]: Springer, Berlin, Germany, 2004.

KIM, W. Cloud computing: Today and tomorrow. *Journal of Object Technology*, Citeseer, v. 8, n. 1, p. 65–72, 2009.

KUKKONEN, S.; LAMPINEN, J. Gde3: The third evolution step of generalized differential evolution. In: IEEE. *2005 IEEE congress on evolutionary computation*. [S.l.], 2005. v. 1, p. 443–450.

KUMBHARE, N.; TUNC, C.; HARIRI, S.; DJORDJEVIC, I.; AKOGLU, A.; SIEGEL, H. J. Just in time architecture (jita) for dynamically composable data centers. In: IEEE. *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*. [S.l.], 2016. p. 1–8.

LAURENT, A.; STRAUSS, O.; BOUCHON-MEUNIER, B.; YAGER, R. *Information Processing and Management of Uncertainty: 15th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, IPMU 2014, Montpellier, France, July 15-19, 2014. Proceedings*. Springer International Publishing, 2014. (Communications in Computer and Information Science, pt. 2). ISBN 9783319088556. Available at: <https://books.google.com.br/books?id=_2IlBAAAQBAJ>.

LEW, A.; MAUCH, H. *Dynamic Programming: A Computational Tool*. Springer Berlin Heidelberg, 2006. (Studies in Computational Intelligence). ISBN 9783540370130. Available at: <https://books.google.com.br/books?id=H_m59Mp1kkEC>.

LI, B.; LI, J.; TANG, K.; YAO, X. Many-objective evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR)*, ACM, v. 48, n. 1, p. 13, 2015.

LI, C.-S.; FRANKE, H.; PARRIS, C.; ABALI, B.; KESAVAN, M.; CHANG, V. Composable architecture for rack scale big data computing. *Future Generation Computer Systems*, Elsevier, v. 67, p. 180–193, 2017.

LIN, A.-D.; LI, C.-S.; LIAO, W.; FRANKE, H. Capacity optimization for resource pooling in virtualized data centers with composable systems. *IEEE Transactions on Parallel and Distributed Systems*, IEEE, v. 29, n. 2, p. 324–337, 2018.

LIU, C.; DU, W.-B.; WANG, W.-X. Particle swarm optimization with scale-free interactions. *PloS one*, Public Library of Science, v. 9, n. 5, p. e97822, 2014.

LYNN, T. Addressing the complexity of hpc in the cloud: Emergence, self-organisation, self-management, and the separation of concerns. In: *Heterogeneity, High Performance Computing, Self-Organization and the Cloud*. [S.l.]: Palgrave Macmillan, Cham, 2018. p. 1–30.

MACIEL, P.; MATOS, R.; SILVA, B.; FIGUEIREDO, J.; OLIVEIRA, D.; FÉ, I.; MACIEL, R.; DANTAS, J. Mercury: Performance and dependability evaluation of systems with exponential, expolynomial, and general distributions. In: IEEE. *Dependable Computing (PRDC), 2017 IEEE 22nd Pacific Rim International Symposium on*. [S.l.], 2017. p. 50–57.

MACIEL, P.; TRIVEDI, K.; MATIAS, R.; KIM, D. Dependability modeling in: Performance and dependability in service computing: Concepts, techniques and research directions. *Hershey: IGI Global, Pennsylvania, USA*, v. 13, 2010.

MARTELLO, S.; TOTH, P. *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990. ISBN 0-471-92420-2.

NADKAMI, A. *Quantifying Datacenter Inefficiency: Making the Case for Composable Infrastructure*. [S.l.]: IDC, 2017. <http://www.integra1.net/uploads/files/IDC%20Making%20the%20Case%20for%20Composable%20Infrastructure.pdf>.

NADKARNI, A.; SHEPPARD, E.; STOLARKSI, K. *Worldwide Composable/Disaggregated Infrastructure Forecast, 2018–2023*. [S.l.]: IDC, 2018.

PAGÈS, A.; SERRANO, R.; PERELLÓ, J.; SPADARO, S. On the benefits of resource disaggregation for virtual data centre provisioning in optical data centres. *Computer Communications*, Elsevier, v. 107, p. 60–74, 2017.

PEDERSEN, M. E. H. Good parameters for differential evolution. *Magnus Erik Hvass Pedersen*, v. 49, 2010.

PEDERSEN, M. E. H. Good parameters for particle swarm optimization. *Hvass Lab., Copenhagen, Denmark, Tech. Rep. HL1001*, 2010.

POIAN, M.; POLES, S.; BERNASCONI, F.; LEROUX, E.; STEFFÉ, W.; ZOLESI, M. Multi-objective optimization for antenna design. In: IEEE. *2008 IEEE International Conference on Microwaves, Communications, Antennas and Electronic Systems*. [S.l.], 2008. p. 1–9.

RANKOTHGE, W.; LE, F.; RUSSO, A.; LOBO, J. Optimizing resource allocation for virtualized network functions in a cloud center using genetic algorithms. *IEEE Transactions on Network and Service Management*, IEEE, v. 14, n. 2, p. 343–356, 2017.

RONG, A.; FIGUEIRA, J. R. Dynamic programming algorithms for the bi-objective integer knapsack problem. *European Journal of Operational Research*, Elsevier, v. 236, n. 1, p. 85–99, 2014.

RONG, A.; FIGUEIRA, J. R.; KLAMROTH, K. Dynamic programming based algorithms for the discounted {0–1} knapsack problem. *Applied Mathematics and Computation*, Elsevier, v. 218, n. 12, p. 6921–6933, 2012.

ROOZBEH, A.; SOARES, J.; MAGUIRE, G. Q.; WUHIB, F.; PADALA, C.; MAHLOO, M.; TURULL, D.; YADHAV, V.; KOSTIĆ, D. Software-defined "hardware" infrastructures: A survey on enabling technologies and open research directions. *IEEE Communications Surveys & Tutorials*, IEEE, v. 20, n. 3, p. 2454–2485, 2018.

ROSENDO, D.; GOMES, D.; SANTOS, G. L.; GONCALVES, G.; MOREIRA, A.; FERREIRA, L.; ENDO, P. T.; KELNER, J.; SADOK, D.; MEHTA, A. et al. A methodology to assess the availability of next-generation data centers. *The Journal of Supercomputing*, Springer, p. 1–25, 2019.

SAFI, H. H.; UCAN, O. N.; BAYAT, O. Study the effect of high dimensional objective functions on multi-objective evolutionary algorithms. In: ACM. *Proceedings of the Fourth International Conference on Engineering & MIS 2018*. [S.l.], 2018. p. 65.

SCHWARTZ, C.; PRIES, R.; TRAN-GIA, P. A queuing analysis of an energy-saving mechanism in data centers. In: IEEE. *The International Conference on Information Network 2012*. [S.l.], 2012. p. 70–75.

SHAN, A. Heterogeneous processing: a strategy for augmenting moore's law. *Linux Journal*, Belltown Media, v. 2006, n. 142, p. 7, 2006.

SHARMA, Y.; JAVADI, B.; SI, W.; SUN, D. Reliability and energy efficiency in cloud computing systems: Survey and taxonomy. *Journal of Network and Computer Applications*, Elsevier, v. 74, p. 66–85, 2016.

SHENOY, G.; SRIVASTAVA, U.; SHARMA, S. *Operations Research for Management*. Wiley Eastern, 1986. ISBN 9780852269176. Available at: <https://books.google.com.br/books?id=UHrk32oEZ7sC>.

SINGH, H. K.; ISAACS, A.; RAY, T. A pareto corner search evolutionary algorithm and dimensionality reduction in many-objective optimization problems. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 15, n. 4, p. 539–556, 2011.

SINGH, U.; SINGH, S. N. Optimal feature selection via nsga-ii for power quality disturbances classification. *IEEE Transactions on Industrial Informatics*, IEEE, v. 14, n. 7, p. 2994–3002, 2017.

SMITH, W. E.; TRIVEDI, K. S.; TOMEK, L. A.; ACKARET, J. Availability analysis of blade server systems. *IBM Systems Journal*, IBM, v. 47, n. 4, p. 621–640, 2008.

SPECIFYING Data Center IT Pod Architectures. 2018. <http://download.schneider-electric.com/files?p_Doc_Ref=WTOL-AHAPRN_R0-EN>. Accessed: July, 2018.

STORN, R.; PRICE, K. Minimizing the real functions of the icec'96 contest by differential evolution. In: IEEE. *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*. [S.l.], 1996. p. 842–844.

STORN, R.; PRICE, K. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, Springer, v. 11, n. 4, p. 341–359, 1997.

TANG, L.; ZHAO, Y.; LIU, J. An improved differential evolution algorithm for practical dynamic scheduling in steelmaking-continuous casting production. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 18, n. 2, p. 209–225, 2014.

TSARMPOPOULOS, D. G.; PAPANIKOLAOU, A. N.; KOTSIANTIS, S.; GRAPSA, T. N.; ANDROULAKIS, G. S. Performance evaluation and comparison of multi-objective optimization algorithms. In: IEEE. *2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)*. [S.l.], 2019. p. 1–6.

UCHECHUKWU, A.; LI, K.; SHEN, Y. Energy consumption in cloud computing data centers. *International Journal of Cloud Computing and Services Science (IJ-CLOSER)*, v. 3, n. 3, p. 145–162, 2014.

VARGHA, A.; DELANEY, H. D. The kruskal-wallis test and stochastic homogeneity. *Journal of Educational and behavioral Statistics*, Sage Publications Sage CA: Los Angeles, CA, v. 23, n. 2, p. 170–192, 1998.

VERMA, A. K.; AJIT, S.; KARANKI, D. R. *Reliability and safety engineering*. [S.l.]: Springer, 2010.

VO-DUY, T.; DUONG-GIA, D.; HO-HUU, V.; VU-DO, H.; NGUYEN-THOI, T. Multi-objective optimization of laminated composite beam structures using nsga-ii algorithm. *Composite Structures*, Elsevier, v. 168, p. 498–509, 2017.

WAGNER, B.; SOOD, A. Economics of resilient cloud services. In: IEEE. *2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. [S.l.], 2016. p. 368–374.

WANG, D.; TAN, D.; LIU, L. Particle swarm optimization algorithm: an overview. *Soft Computing*, Springer, v. 22, n. 2, p. 387–408, 2018.

WU, X.; CHE, A. A memetic differential evolution algorithm for energy-efficient parallel machine scheduling. *Omega*, Elsevier, v. 82, p. 155–165, 2019.

YANG, F.; JIN, T.; LIU, T.-Y.; SUN, X.; ZHANG, J. Boosting dynamic programming with neural networks for solving np-hard problems. In: *Asian Conference on Machine Learning*. [S.l.: s.n.], 2018. p. 726–739.

YEO, S.; LEE, H.-H. Using mathematical modeling in provisioning a heterogeneous cloud computing environment. *Computer*, IEEE, v. 44, n. 8, p. 55–62, 2011.

ZERVAS, G.; JIANG, F.; CHEN, Q.; MISHRA, V.; YUAN, H.; KATRINIS, K.; SYRIVELIS, D.; REALE, A.; PNEVMATIKATOS, D.; ENRICO, M. et al. Disaggregated compute, memory and network systems: A new era for optical data centre architectures. In: IEEE. *2017 Optical Fiber Communications Conference and Exhibition (OFC)*. [S.l.], 2017. p. 1–3.

ZERVAS, G.; YUAN, H.; SALJOGHEI, A.; CHEN, Q.; MISHRA, V. Optically disaggregated data centers with minimal remote memory latency: technologies, architectures, and resource allocation. *Journal of Optical Communications and Networking*, Optical Society of America, v. 10, n. 2, p. A270–A285, 2018.

ZHANG, H.; JIANG, Y.; HUNG, W. N.; SONG, X.; GU, M.; SUN, J. Symbolic analysis of programmable logic controllers. *IEEE Transactions on Computers*, IEEE, v. 63, n. 10, p. 2563–2575, 2013.