Pós-Graduação em Ciência da Computação

Paulo Roberto Pereira da Silva

Edge-Fog-Cloud Continuum Applications: Analytical and Hierarchical Models for Availability and Performability Evaluation

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

Recife
2023

Paulo Roberto Pereira da Silva

Edge-Fog-Cloud Continuum Applications: Analytical and Hierarchical Models for Availability and Performability Evaluation

A Ph.D. Thesis presented to the Center for Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Philosophy Doctor in Computer Science.

Concentration Area: Performance and Dependability Evaluation
Advisor: Paulo Romero Martins Maciel
Co-Advisor: Jean Carlos Teixeira de Araujo

Recife
2023

FICHA

Paulo Roberto Pereira da Silva

“Edge-Fog-Cloud Continuum Applications: Analytical and Hierarchical Models for Availability and Performability Evaluation”

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação. Área de Concentração: Redes de Computadores e Sistemas Distribuídos

Aprovado em: 26/05/2023.

_____

Orientador: Prof. Dr. Paulo Romero Martins Maciel

BANCA EXAMINADORA

_____

Prof. Dr. Eduardo Antônio Guimarães Tavares
Centro de Informática / UFPE

_____

Prof. Dr. Jamilson Ramalho Dantas
Centro de Informática / UFPE

_____

Prof. Dr. Antônio Alfredo Ferreira Loureiro
Departamento de Ciência da Computação / UFMG

_____

Prof. Dr. Mario Antônio Ribeiro Dantas
Departamento de Ciências da Computação / UFJF

_____

Prof. Dr. Bruno Silva
Departamento de Ciência da Computação / UPM

I dedicate this work to my mother who always tried her best to make me succeed. She is responsible for who I am and for all I have achieved so far. I will never be able to compensate her sacrifices. I would also like to dedicate this work to my father for all the support he always gave me during the difficulties.

ACKNOWLEDGEMENTS

God is good all the time, and thanks to Him, I have reached this moment. He gave me opportunities that changed my life, health to work hard, and strength to overcome adversities. All I am and have, I owe God. I would also like to thank my mother for all the efforts and sacrifices that she has made her entire life to make me grow healthy and reach this moment. She is my biggest inspiration. **Carla Regina Pereira da Silva** is and will always be my biggest hero. She is the greatest warrior that I ever met and I am really proud of having her as my mother. Today, if I see further it is because I stand on the shoulders of a giant, and this giant is my mother. I would like to thank my father, Airton Mariano da Silva, as well, a great man that gave me all the psychological and financial support to accomplish it. I am so grateful and lucky for having these two people as parents.

My Fiancé, Luara Luiza de Oliveira, is a great woman, and I am really lucky to have her in my life. Her support during this phase of my life was crucial to help me achieve and finish this thesis. She always brings joy, lightness, and hope to my life when I need it the most. Hopefully, she will become my wife, and we will have kids of our own. I am really grateful to her and all support she always gives me. I am a better man because of her.

I would like to thank my lab friends (MoDCS group), which always tried to help me and responded to all my questions without hesitation or delay. They also built a really nice learning environment, where everyone is more than welcome.

I would also like to thank my Professor, co-advisor, great person, and friend, Jean Teixeira for all his support, all partnership, all psychological and financial help since my bachelor's degree.

I am really grateful to Paulo Maciel for opening the doors of the CIn-UFPE to me, and because of him, I had the opportunity to become part of one of the best tech centers in Brazil and the world. Paulo Maciel is also a hardworking Professor and such a great inspiration as a human being and professional for all of us.

To finish, I would like to thank the financial support from the funding agency FACEPE, which was essential for concluding this stage of my life.

“Mahalo ke Akua.”
(COPPOLA, 2013, p. 10)

ABSTRACT

Connectivity has introduced significant transformations in our society, requiring the continuous improvement of Quality of Service (QoS). The rise of various emerging technologies has created a demand for networks capable of low-latency communication to facilitate real-time data processing. As our reliance on these technologies grows, it becomes increasingly important to address the latency issue. While cloud computing environments offer high availability, reliability, and performance, they may not be suitable for applications that require low latency, such as disaster risk minimization, smart-traffic management, and crime prevention. For instance, numerous lives could be lost if a disaster risk minimization service delays in providing alerts about an earthquake. To overcome the challenges posed by latency and enhance computing capabilities between the cloud and edge devices (e.g., controllers, sensors, and smartphones), two complementary paradigms, namely edge and fog computing, have been proposed. However, evaluating the dependability and performance of distributed computing environments remains a concern due to the numerous challenges involved in supporting the required QoS of these systems. Therefore, this study aims to investigate the dependability of edge, fog, and cloud computing environments by assessing their availability and the resulting impact on performance. Additionally, we propose analytical and hierarchical models that facilitate scalability and capacity planning in these computing environments. The metrics considered in this study include availability, K out of N availability, capacity-oriented availability, as well as performance metrics such as utilization, response time, waiting time, and discard rate. Our proposed models serve as valuable tools for researchers, system designers, and practitioners in the field of edge-fog-cloud environments. By understanding the behavior and limitations of these systems, we can enhance their design, operation, and maintenance, ultimately leading to more reliable, efficient, and resilient infrastructures.

Keywords: Cloud Computing. Fog Computing. Edge Computing. Modeling. Availability. Performance.

RESUMO

A conectividade introduziu transformações significativas em nossa sociedade, exigindo o contínuo aprimoramento da Qualidade de Serviço (QoS, sigla em inglês). O surgimento de várias tecnologias emergentes criou uma demanda por redes capazes de comunicação com baixa latência para facilitar o processamento de dados em tempo real. À medida que nossa dependência dessas tecnologias cresce, torna-se cada vez mais importante abordar o problema de latência. Embora os ambientes de computação em nuvem ofereçam alta disponibilidade, confiabilidade e desempenho, eles podem não ser adequados para serviços que requerem baixa latência, como minimização de risco de desastres, gerenciamento de tráfego veicular inteligente e prevenção de crimes. Por exemplo, inúmeras vidas podem ser perdidas se um serviço de minimização de risco de desastres atrasar na entrega de alertas sobre um terremoto. Para superar os desafios impostos pela latência e aprimorar as capacidades de computação entre a nuvem e os dispositivos de borda (como controladores, sensores e smartphones), foram propostos dois paradigmas complementares, chamados de computação em borda (Edge) e neblina (Fog). No entanto, avaliar a confiabilidade e o desempenho de ambientes de computação distribuída continua sendo um grande desafio devido aos inúmeras questões envolvidas no suporte à QoS necessária desses sistemas. Portanto, este estudo tem como objetivo investigar a confiabilidade dos ambientes de computação em borda, névoa e nuvem, por meio da avaliação de sua disponibilidade e do impacto resultante no desempenho. Além disso, propomos modelos analíticos e hierárquicos que facilitam a escalabilidade e o planejamento de capacidade nesses ambientes de computação. As métricas consideradas neste estudo incluem disponibilidade, disponibilidade de K de N, disponibilidade orientada à capacidade, bem como métricas de desempenho, como utilização, tempo de resposta, tempo de espera e taxa de descarte. Nossos modelos propostos servem como ferramentas valiosas para pesquisadores, projetistas de sistemas e profissionais no campo de ambientes de borda-névoa-nuvem. Ao compreender o comportamento e as limitações desses sistemas, podemos aprimorar seu projeto, operação e manutenção, levando a infraestruturas mais confiáveis, eficientes e resilientes.

Palavras-chaves: Computação na Nuvem. Computação na Névoa. Computação na Borda. Modelagem. Disponibilidade. Desempenho.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

This chapter provides an overview of the motivation and objectives underlying this thesis. We outline the reasons and methodologies through which we contribute to the advancement of edge-fog-cloud continuum services within the state of the art. These services hold significant relevance in the contemporary era, characterized by the dominance of information, with data serving as its principal asset. Additionally, artificial intelligence, the Internet of Things (IoT), big data, cloud computing, and wireless networks act as the primary means for collecting and processing information (MELO et al., 2020; LECLERC; CALE, 2020). The emergence of these transformative technologies has brought about unforeseen changes in our society and businesses, facilitating the generation and analysis of vast volumes of dynamic data. Extracting novel societal patterns from this data has proven instrumental in understanding our evolving world.

1.1 MOTIVATION

Advancements in sensor technology, wireless networks, embedded systems, and actuators have paved the way for the development of affordable, energy-efficient, and compact devices that can connect to the Internet. This transformative innovation has given rise to the Internet of Things (IoT) (SAMIE; BAUER; HENKEL, 2016), where devices seamlessly interact and communicate within the network, offering extensive control over various services. The scope of the IoT paradigm encompasses a wide range of applications, including healthcare, smart homes, intelligent buildings, campus automation, urban infrastructure, industrial automation (known as Industry 4.0), and numerous others (SAMIE; BAUER; HENKEL, 2016).

The generation of extensive data by distributed sensors has necessitated the acquisition, integration, storage, processing, and utilization of this data as a vital means for companies to achieve their business objectives. In addition, researchers and engineers face the significant challenge of managing these vast and diverse data sets in highly distributed environments, particularly within cloud platforms (CAI et al., 2016). The proliferation of data generated through IoT sensors plays a pivotal role in the realm of big data, which can be characterized by three primary dimensions: volume, variety, and velocity (MARJANI et al., 2017).

Presently, data mining, statistical analysis, and machine learning methods are extensively employed for addressing specific problems and conducting general data analytics. Forecasts indicate a projected increase of 1 trillion sensors and actuators by the year 2030 (CHEN; LIN, 2014). This growth will significantly influence our understanding of big data and its applications. Currently, IoT services offer a multitude of resources and real-time

applications, facilitating effective communication among diverse deployed systems. Such services fulfill numerous societal requirements and have the potential to mitigate future challenges, including the management of food and water waste (RAMUNDO; OTCU; TERZI, 2020).

Harnessing valuable insights and knowledge from extensive datasets, in order to enhance the overall quality of our lives, is an enticing prospect; however, it is an endeavor that is far from straightforward. To surmount this intricate and demanding undertaking, the integration of innovative technologies, algorithms, and infrastructures becomes imperative (CHEN; LIN, 2014). The recent advancements in computing paradigms and sophisticated machine learning methodologies have opened up unprecedented opportunities for big data analytics, making it particularly suitable for applications within the Internet of Things (IoT) domain.

Within the context of the Internet of Things (IoT) ecosystem, cloud computing plays a pivotal role by offering significant processing power and storage capabilities, thereby alleviating the burden on local systems. Cloud computing facilitates convenient access to shared computing resources and services, including network infrastructure, storage, operating systems, and applications. These resources and functionalities can be dynamically allocated and released with minimal administrative effort while following a pay-as-you-go model based on resource utilization (NGUYEN; MIN; CHOI, 2020).

This distinctive attribute empowers administrators to concentrate solely on their business models, delegating the intricate details of the underlying infrastructure to the cloud service provider (MELL; GRANCE et al., 2011; BANKOLE; AJILA et al., 2013). The experience of procuring cloud services is often likened to the consumption of public utilities like electricity, where users pay exclusively for the resources they consume (BAUER; ADAMS et al., 2012).

Furthermore, cloud computing enables organizations and individuals to rely on external providers for data storage and processing. As a result, it has emerged as a highly successful computing paradigm, fostering an agile service-based computing market that offers seamless access to computing resources (ROY et al., 2011). By leveraging cloud computing, rapid deployment of new services with unrestricted connectivity to the Internet becomes feasible (DUTREILH et al., 2010).

The realm of cloud computing offers seemingly boundless resources that can be accessed at any given time and in any desired quantity. Despite its support for emerging technologies, there exists a substantial gap in terms of data processing and real-time decision-making for applications sensitive to latency, such as smart city services (NGUYEN; MIN; CHOI, 2020). Notably, the transmission of data to cloud servers often necessitates considerable network bandwidth, thereby leading to increased latency in the service (PEREIRA; ARAUJO; MACIEL, 2019). This drawback can have critical implications for certain applications, particularly those concerning security systems. In response to these challenges,

researchers and companies have proposed two paradigms: edge computing and fog computing (SUNYAEV, 2020). Fog computing extends communication, processing power, and storage capabilities toward the network edge, acting as an intermediary layer between the cloud and devices (NGUYEN; MIN; CHOI, 2020; PEREIRA et al., 2020). Conversely, edge computing pertains to the utilization of processing power, storage, and communication capabilities within the devices themselves (SUNYAEV, 2020).

Applications such as disaster risk minimization and crime prevention necessitate uninterrupted operation, as the potential loss of lives looms large (PEREIRA et al., 2021). Another notable instance is traffic congestion in metropolitan areas, which often gives rise to unpredictable accidents. However, the implementation of smart traffic management systems has the potential to prevent numerous collisions (PRAVEEN; RAJ, 2020). It is worth mentioning that both edge and fog paradigms are susceptible to failures and performance degradation. Therefore, it is of paramount importance to investigate and evaluate the integrated aspects of availability and performance in edge and fog computing environments. In essence, comprehensive studies should be conducted to assess and devise strategies for mitigating failures and performance degradation.

The study of dependability assumes paramount importance in the realm of diverse systems and services that have emerged in the wake of the Internet's advent (AVIZIENIS; LAPRIE; RANDELL, 2001; LAPRIE, 1992). These services are vulnerable to numerous threats and must adhere to specific requirements for service provision (PATEL; RANABAHU; SHETH, 2009). Within cloud, fog, and edge computing, the establishment of Service Level Agreements (SLAs) between providers and contractors aims to ensure the attainment of minimum acceptable levels for various dependability attributes (KAFHALI; SALAH, 2017).

It is worth noting that the precise specifications for dependability requirements in the context of edge and fog computing remain relatively uncharted within the existing literature, primarily due to the novelty of these paradigms. However, the most common objectives pertaining to dependability in edge and fog environments revolve around enhancing availability, reliability, performance, and Quality of Service (QoS). Numerous techniques employed to augment these attributes are frequently predicated on redundancy models. Nevertheless, redundancy comes at a cost, underscoring the need for judicious and intelligent application of such approaches (BAKHSHI; RODRIGUEZ-NAVAS, 2020).

Taking all of this into consideration, it becomes pertinent to inquire: How can the integration of edge, fog, and cloud computing environments effectively address latency concerns and enhance the dependability and performance of distributed systems, particularly when catering to applications that necessitate low latency, such as disaster risk minimization, smart-traffic management, and crime prevention? The response to the aforementioned research question serves as the guiding principle and compass for this thesis.

1.2 OBJECTIVES

When comparing edge and fog computing with the cloud, it becomes evident that their levels of availability are comparatively lower, thereby rendering them potentially critical for select applications, particularly those of an urgent nature, such as emergency systems. Such applications necessitate sustained operational functionality to minimize the risk of significant loss of life. An illustrative example lies within urban settings, where instances of robberies introduce unpredictable peril to the general population and pose potential threats to innocent individuals. Consequently, the implementation of intelligent facial recognition systems holds promise in reducing the occurrence of such crimes (MARIAPPAN; THONG; MUTHUKARUPPAN, 2020).

Under these circumstances, it becomes imperative to thoroughly evaluate pertinent metrics associated with service provisioning within the edge and fog computing environments. These metrics encompass crucial aspects such as availability, downtime, capacity-oriented availability, and performance (MACIEL et al., 2012; Fernandes et al., 2012). Furthermore, it is essential to conduct a comprehensive assessment of these metrics within the broader context of the edge-fog-cloud continuum, where these paradigms operate in synergy to deliver integrated solutions.

The primary goal of this thesis centers on the proposition of analytical and hierarchical models capable of assessing the availability and performance of the integrated edge-fog-cloud continuum environments. Through the application of these models, our research endeavors to examine the intricate relationship between availability and performance, commonly referred to as "performability," within the realms of edge, fog, and cloud computing environments. Additionally, these models will illuminate the critical components within the system that necessitate the intelligent adoption of redundancy measures. In particular, the inclusion of performability models will account for the dynamic behavior of a system's performance, duly considering instances of failures and subsequent repairs (MACIEL et al., 2012).

In addition to the aforementioned objective, our work also aims to propose a novel capacity-oriented availability model that leverages the average resource capacity as a means to quantify system availability. This model stands to provide valuable insights into the availability aspects of the examined system, shedding light on its robustness and capacity to effectively meet the demands imposed upon it.

To achieve these goals, our research endeavors encompass a set of distinct and noteworthy objectives that, in turn, serve as significant contributions to the academic and industrial field. These objectives are enumerated as follows:

- develop analytical models for availability and performability in edge and fog computing environments;

- construct hierarchical models to evaluate the edge-fog-cloud continuum and incorporate fault coverage probability;
- investigate the impact of availability on the performance of edge and fog computing;
- propose analytical models for capacity-oriented availability and compare cost effectiveness among edge, fog, and cloud computing paradigms;

1.3 THESIS OUTLINE

This work is outlined as follows: in Chapter 2, we provide an introduction to the fundamental concepts of edge, fog, and cloud computing, along with an elucidation of the essential notions pertaining to availability and performance evaluation. This foundational knowledge is crucial for a comprehensive understanding of the present study. Chapter 3 presents a thorough review of relevant literature, wherein a comparative analysis is conducted to juxtapose the existing works with our research. Moving forward, Chapter 4 outlines the methodology employed in developing the integrated solution for cloud infrastructure planning, accompanied by a detailed exposition of the methods employed. The proposed models, encompassing their distinctive features and a holistic overview, are expounded upon in Chapter 5. Subsequently, Chapter 6 details the experiments conducted to assess the efficacy of the proposed models and presents the achieved results. Lastly, Chapter 7 serves as a culmination of this study, presenting the concluding remarks and engaging in a comprehensive discussion on potential avenues for future research endeavors.

2 BACKGROUND

Internet of Things (IoT) applications are commonly established on intricate computational frameworks that encompass the integration of cloud, fog, and edge computing paradigms (ARAUJO et al., 2019). In order to mitigate resource wastage, it becomes imperative to devise models and conduct evaluations of these infrastructures. This chapter serves as an introduction to the fundamental concepts that underpin our strategic approach. Our research encompasses the capacity planning of edge, fog, and cloud paradigms, while simultaneously addressing the critical aspects of availability and performance evaluation.

Moreover, the following sections will depict how these computing paradigms contribute to the enhancement of the existing and future landscape of interconnected devices. Beginning with an examination of the cloud paradigm, we shall subsequently delve into a comprehensive discussion on the intricacies of fog infrastructure and edge environments. Lastly, we shall outline the key distinctions between the edge and fog paradigms, while also providing a concise overview of the different categories that near real-time systems encompass.

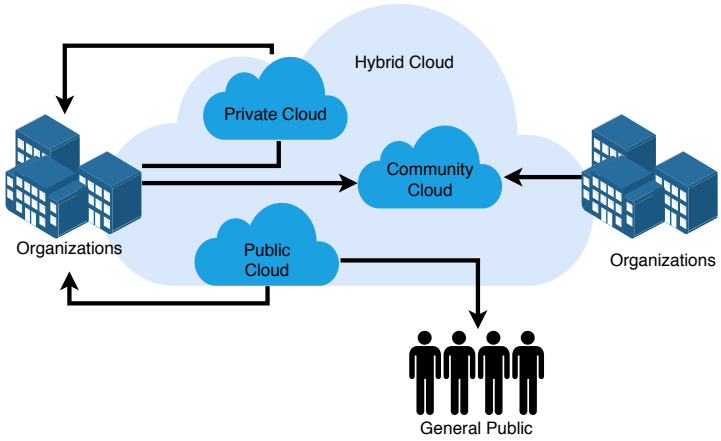
2.1 CLOUD COMPUTING

Cloud computing represents a comprehensive model that fosters pervasive and on-demand network access to shared computing resources (MELL; GRANCE et al., 2011). In recent years, this paradigm has gained considerable traction, primarily due to its ability to empower organizations with the option to entrust external providers with the storage and processing of their data. The advent of cloud computing has significantly contributed to the establishment of an agile and service-centric technology market, wherein computing resources can be readily accessed and utilized. As a result, the deployment of novel services has been greatly facilitated, enabling unrestricted connectivity to the vast expanse of the World Wide Web (PEREIRA; ARAUJO; MACIEL, 2019).

The cloud computing paradigm encompasses a diverse range of deployment models, classified into four main categories: public cloud, community cloud, private cloud, and hybrid cloud (MELL; GRANCE et al., 2011). Public clouds, offered and managed by service providers, have gained significant popularity owing to their ease of maintenance and cost-effectiveness for customers. In this deployment model, a robust Service Level Agreement (SLA) is established between customers and providers to ensure the maintenance of trust and adherence to agreed-upon service standards. Public clouds operate on a pay-as-you-go pricing model, enabling customers to pay for infrastructure usage only, a feature that has proven enticing for many businesses. However, it is important to note that public clouds may not provide complete customization options for network bandwidth, hardware,

middleware, and security settings, which may render them unsuitable for certain business requirements (MELL; GRANCE et al., 2011).

Figure 1 – Cloud deployment models.



Source: Elaborated by the author.

In cloud computing, there are different deployment models to offer solutions with specific requirements. Community clouds, for instance, serve as integrated platforms that support the needs of businesses, companies, communities, or industries, with a shared focus on common concerns such as missions, security requirements, policies, and compliance considerations (MELL; GRANCE et al., 2011). These deployment models are managed collectively by the organizations that comprise the community, thus avoiding dependence on a singular large cloud provider for the IT infrastructure.

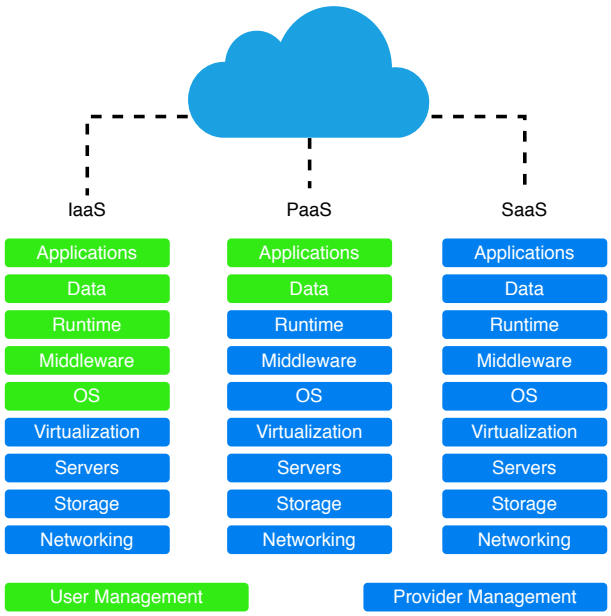
In contrast, private clouds are designed to cater exclusively to a single entity, ensuring heightened privacy and the ability to customize resources according to specific requirements. Similar to traditional company-owned data centers, private clouds offer companies the advantage of a dedicated infrastructure for their applications. However, it is important to note that private clouds do not benefit from the pay-as-you-go pricing model commonly found in public clouds (PEREIRA; ARAUJO; MACIEL, 2019).

Hybrid clouds, on the other hand, represent a fusion of the previously defined deployment models. This hybrid approach grants users greater control over virtualized infrastructure and harnesses the combined capabilities of different deployment models (SOTOMAYOR et al., 2009).

Within the cloud computing paradigm, services are broadly categorized into three types: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). IaaS enables customers to leverage hardware-related services using cloud computing principles. These services encompass storage solutions (e.g., database or disk storage), processing power, virtual servers, networking resources, and more (MELL; GRANCE et al., 2011). For instance, imagine an individual seeking to implement a facial

recognition-based security system in a building. In such a scenario, the individual would engage with cloud providers to acquire an infrastructure as a service to meet their requirements (YOUSEFPOUR et al., 2019b).

Figure 2 – Cloud service models.



Source: Elaborated by the author.

Platform as a Service (PaaS) is a cloud computing model that focuses on providing developers with an environment to create and manage software applications throughout their lifecycle. In this model, customers don't have to concern themselves with hardware configurations or operating systems as the entire development infrastructure is already established and taken care of (AHMED et al., 2012).

On the other hand, Software as a Service (SaaS) is a cloud computing model that offers a complete software package hosted in the cloud. With SaaS, customers can access and utilize software applications without the need for installation or worrying about database scalability, software errors, or socket management. It operates on a pay-per-use basis, allowing customers to only pay for the services they actually utilize (YOUSEFPOUR et al., 2019b).

2.2 FOG COMPUTING AT A GLANCE

Fog computing, a term introduced by researchers at Cisco Systems (BONOMI et al., 2012), pertains to the utilization of processing power in close proximity to the network edge. A related concept, known as cloudlets, is primarily associated with mobile networks. Fog computing finds extensive application in the context of IoT infrastructure (NAHA et al., 2018). This paradigm encompasses both virtualized and non-virtualized resources,

delivering storage, processing power, and networking services to end devices as well as cloud servers (DASTJERDI et al., 2016a).

The core objective of fog computing is to facilitate low-latency services (LI et al., 2017), while simultaneously enhancing the performance of non-latency services. By bringing computational power closer to end devices, fog computing minimizes networking bandwidth consumption and improves overall application performance. Fog environments often consist of a multitude of heterogeneous nodes, such as sensors and actuators, which generate substantial volumes of data requiring processing. In situations where fog infrastructures lack the necessary resources to handle this data influx, it is transmitted to the cloud for further processing (BONOMI et al., 2012). Moreover, fog computing supports various aspects including computing resources, cloud integration, communication protocols, mobility, interface heterogeneity, and distributed data analytics (DASTJERDI et al., 2016b).

It is crucial to note that any device equipped with processing power, networking capabilities, and storage capacity can function as a fog device (NAHA et al., 2018). These devices serve the dual roles of servers and gateways. Fog servers manage multiple fog devices, while fog gateways oversee and translate data from heterogeneous devices for transmission to fog servers (NAHA et al., 2018).

The main advantages associated with fog computing environments are:

- **reduction of network traffic** - fog environments drastically reduce the traffic sent to the cloud because most of the data is processed closer to the end devices;
- **suitable for IoT tasks and queries** - by using fog environments, IoT tasks may be processed closer to the physical location of the sensors/actuators. Consequently, the processing power is closer to the geographical context of the sensor or actuator;
- **low latency requirement** - the data processing is performed closer to the nodes; thus, it makes real-time response possible.

According to a report by Cisco Systems, it was projected five years ago that there would be over 50 billion connected devices worldwide (NAHA et al., 2018). Consequently, fog computing environments have emerged as a compelling solution for handling the immense volume of data generated. These environments serve as a filter, effectively reducing the burden on cloud servers by processing data at the network edge.

2.3 EDGE COMPUTING AT A GLANCE

Edge computing is a paradigm that offers processing, networking, and storage capabilities directly within the end devices themselves (NAHA et al., 2018). Unlike the traditional cloud paradigm, which involves transmitting all raw data from edge devices to the cloud server, edge computing processes the data locally at the edge nodes before selectively forwarding

the relevant insights or processed data to the cloud server (JAYASHREE; SELVAKUMAR, 2020).

Edge computing primarily focuses on the side of the end devices and is not inherently associated with any specific cloud-based services such as IaaS, PaaS, or SaaS. Its main objective is to process data as close as possible to the data sources. Any smart device with the ability to store and process data can serve as an edge node. For example, a smartphone that receives and processes data from a sensor can be considered an edge node for that particular application. Edge computing addresses several challenges, including privacy, latency, and connectivity. Due to its localized nature, edge computing typically exhibits lower latency compared to cloud computing. It also offers higher service availability, as connected devices do not have to rely on a highly centralized platform for service provision, and there are typically multiple physical nodes available (YOUSEFPOUR et al., 2019b).

In the edge computing paradigm, devices not only consume data but also produce it. They may request services from edge servers and perform various computing tasks, such as data storage, offloading, caching, and processing. Additionally, edge devices process incoming requests and deliver services to other end devices. As a result, it is crucial to design edge nodes with careful consideration for reliability, security, and privacy concerns (SHI et al., 2016; PEREIRA et al., 2020).

Compared to the traditional cloud computing paradigm, edge computing offers several benefits. For example, a facial recognition application demonstrated a significant reduction in response time, from 900 to 169 ms, when utilizing edge computing (YI et al., 2015). In another study, the adoption of edge computing for offloading computing tasks in wearable cognitive assistance resulted in an improved response time of 80 ms and a 40% reduction in energy consumption (HA et al., 2014).

Therefore, the edge computing paradigm serves as a means to retrieve, process, and analyze data in close proximity to the data source. This can be achieved through devices like IoT gateways, network switches, or even custom-built devices with sufficient computing power, such as Raspberry Pi. Edge computing empowers users to perform real-time, on-site data analysis gathered from various sensors and IoT devices.

2.4 EDGE VS FOG VS CLOUD COMPUTING

Edge computing and fog computing, although often associated with the same architecture, are distinct paradigms that require differentiation. This section aims to elucidate their differences and compare them to the cloud paradigm.

Edge computing and fog computing, as defined by the OpenFog Consortium, possess nuanced disparities that need to be recognized (CONSORTIUM et al., 2017). Fog computing is characterized as a hierarchical infrastructure that delivers computing, networking, storage, and control capabilities from the cloud to the end devices (YOUSEFPOUR et al., 2019b). In contrast, edge computing primarily focuses on limited processing power at the

network's edge. Fog computing aims to establish a unified infrastructure of computing services that spans from the cloud to the end devices, rather than treating network edges as separate computing architectures (CHIANG et al., 2017). In other words, fog computing brings intelligence down to the local area network (LAN) level, where data or services are processed in fog nodes or IoT gateways. On the other hand, edge computing embeds intelligence, processing, and communication capabilities directly into smart devices such as programmable automation controllers (PAC) (MAHMOOD; RAMACHANDRAN, 2018).

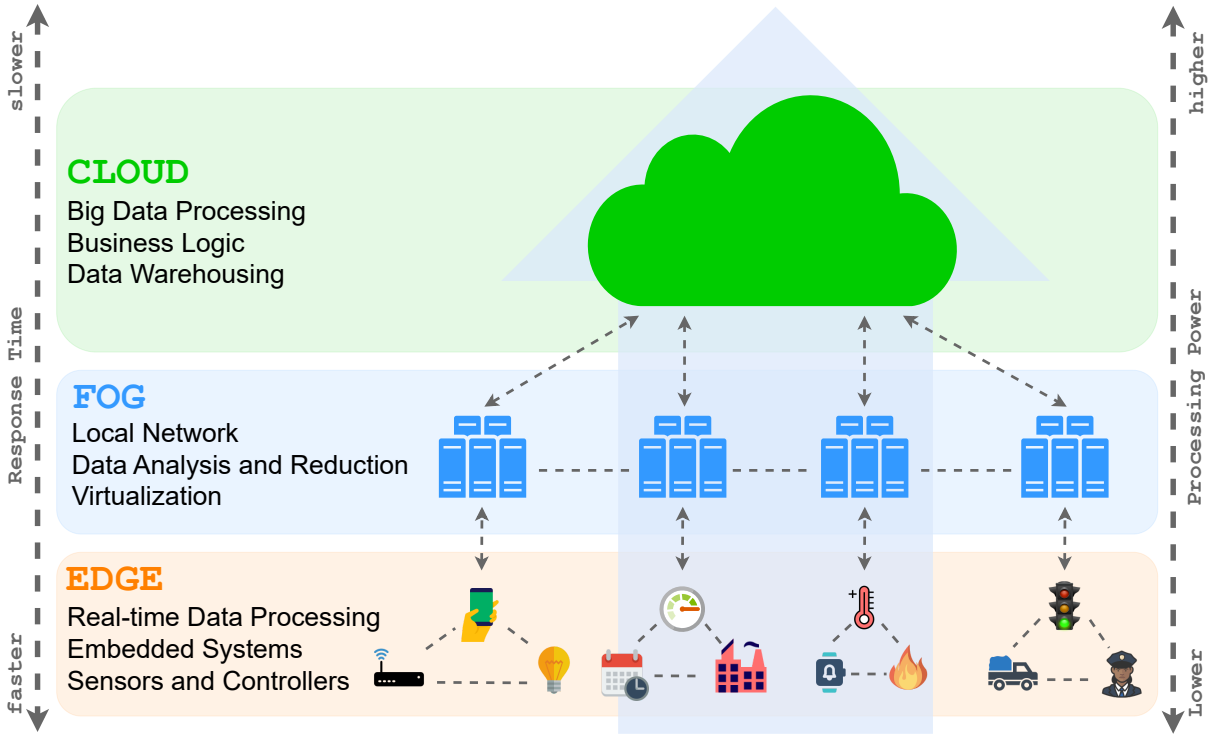
The fog layer exists between the edge and cloud layers and is typically geographically distant from the business premises (JAYASHREE; SELVAKUMAR, 2020). This paradigm typically involves a service provider's network infrastructure that is leased to companies. In the fog layer, fog nodes serve as the processing and communication components. Depending on the nature of the use case, the service provider determines which data is processed at the edge and uploaded to the fog nodes (JAYASHREE; SELVAKUMAR, 2020).

Fog environments encompass the cloud, while edge environments exclude it (MAHMOOD; RAMACHANDRAN, 2018). The fog computing paradigm exhibits a hierarchical and horizontal structure with multiple layers forming a network. Conversely, the edge paradigm typically distinguishes between nodes that are part of the network and those that are not. Fog environments feature extensive peer-to-peer interconnectivity between nodes, while edge environments isolate their nodes, necessitating data transportation back through the cloud for peer-to-peer interactions (MAHMOOD; RAMACHANDRAN, 2018; HARDESTY, 2017).

Both fog and edge computing are distributed computing paradigms that extend the capabilities of cloud computing to the network's edge. They complement the cloud solution, with a particular focus on the emerging IoT architecture (JAYASHREE; SELVAKUMAR, 2020). Both paradigms facilitate storage, networking, and computing services between end devices and cloud data centers. The fog computing architecture typically includes segments of an application running in both the cloud and fog nodes, such as smart gateways, routers, or dedicated fog devices (MAHMOOD; RAMACHANDRAN, 2018). Figure 3 provides a visualization of the edge-fog-cloud computing stack and illustrates the relationship between these paradigms.

To summarize, the utilization of fog and edge environments offers the advantage of reduced bandwidth consumption compared to traditional cloud solutions. Cloud-based systems typically require higher bandwidth, whereas fog and edge paradigms alleviate this demand. It is important to recognize that the Internet inherently exhibits unreliability, and wireless networks possess inherent limitations. Thus, the reduction in bandwidth usage becomes an attractive benefit (YOUSEFPOUR et al., 2019b). Moreover, these paradigms enable data transmission to circumvent the Internet and remain localized on smart devices whenever possible. It is crucial to emphasize that while a significant amount of data may still be transmitted to the cloud, sensitive data can be kept local, allowing more bandwidth

Figure 3 – Edge-fog-cloud stack.



Source: Elaborated by the author.

capacity for cloud users.

Fog and edge computing exhibit closer proximity to end-users and greater geographical distribution. In comparison to the cloud paradigm, fog and edge environments prioritize proximity to end devices and client objectives, local resource pooling, latency, and backbone bandwidth. Consequently, this approach leads to enhanced quality of service (QoS) and ultimately delivers a superior user experience (MAHMOOD; RAMACHANDRAN, 2018).

2.5 NEAR REAL-TIME SYSTEMS AT A GLANCE

Previously, the classification of real-time systems in the literature has predominantly focused on two main categories, namely soft real-time systems and hard real-time systems (LIU; NARAYANAN; BAI, 2000).

Soft real-time systems can be defined as systems in which meeting the specified timing constraints is important, but occasional deadline misses can be tolerated without catastrophic consequences. These systems prioritize timely execution of tasks, but some flexibility exists in meeting all the deadlines consistently. Examples of soft real-time systems include multimedia streaming, online video conferencing, and interactive applications where occasional delays can be accommodated without significant disruption.

On the other hand, hard real-time systems are characterized by strict timing con-

straints, where meeting the specified deadlines is of utmost importance. Failure to meet these deadlines can lead to severe consequences, such as system failure or compromise of safety-critical operations. Hard real-time systems require precise and deterministic timing guarantees, and even a single missed deadline is considered unacceptable. Examples of hard real-time systems include flight control systems, medical monitoring devices, and automotive safety systems.

It is worth noting that this traditional categorization has evolved over time, and contemporary research has explored additional dimensions and classifications to capture the diverse characteristics and requirements of real-time systems. Nowadays, we have the near real-time systems. Near real-time systems refer to a class of computational systems and applications that are designed to process and respond to data promptly, aiming to deliver timely outcomes or actions that closely align with real-time requirements (GOMES et al., 2021). While these systems may not achieve absolute real-time performance, their main focus is to minimize latency while providing fast processing and responsive functionalities (ZHANG; THORBURN, 2022).

The use of near real-time systems spans across various domains where the rapid processing of data is of utmost importance. Examples include financial trading platforms, online gaming, monitoring and control systems, emergency response systems, transportation management systems, and surveillance systems (WEBER et al., 2020). Overall, near real-time systems play a vital role in enabling time-sensitive applications and services by facilitating rapid data processing and decision-making in domains where timely actions are crucial (GOMES et al., 2021).

2.6 AVAILABILITY EVALUATION

Availability evaluation aims at measuring the ability of a system to be in a state where it is able to operate a function at a given instant of time or interval (AVIZIENIS; LAPRIE; RANDELL, 2001; EVER, 2019; MACIEL, 2023b; MACIEL, 2023a). If a system is not fault-tolerant, the system may stop working when a failure happens (AVIŽIENIS, 1967; TORQUATO; UMESH; MACIEL, 2018; MACIEL, 2023b; MACIEL, 2023a). To represent that, we may consider a random variable $X(t)$ as the system's state at the moment of t. $X(t) = 1$ denotes that the system is up, which means the system is working correctly. On the other hand, $X(t) = 0$ means that the system is down (i.e., not working). Formally, if we assume a random variable T expressing the necessary time to reach $X(t) = 0$, where the system begins in $X(0) = 1$, then T is the time that the system takes to fail ($T \geq 0$). $F_T(t)$ is its cumulative distribution function (MACIEL et al., 2012; MACIEL, 2023b; MACIEL, 2023a), where

- $0 \leq F_T(t) \leq 1$,
- $F_T(t)$ is non-decreasing, that is, for any $a < b$, $F_T(a) < F_T(b)$,

- $\lim_{t \to -\infty} F_T(t) = 0$, and $\lim_{t \to \infty} F_T(t) = 1$,
- $P(a < T < b) = F_T(b) - F_T(a)$. $F_T(t)$ is continuous over $t \geq 0$.

The derivative of cumulative distribution function, $F_T(t)$, of continuous random variable T is called the probability density function (*pdf*) of T, which is denoted by $f_T(t)$. More formally:

$$f_T(t) = \frac{dF_T(t)}{dt}, \tag{2.2}$$

where $f_T(t) \geq 0$ and $\int_0^\infty f_T(t)\, dt = 1$, assuming T is defined in $(0, \infty)$.

The *complementary cumulative distribution function* $F_T^c(t)$ is defined by

$$F_T^c(t) = 1 - F_T(t) = 1 - P(T < t) = P(T > t). \tag{2.4}$$

As T is the time to fail, $F_T^c(t) = R(t)$ is the system reliability. Thus,

$$R(t) = F_T^c(t) = 1 - F_T(t) = P(T > t) \tag{2.6}$$

and

$$R(0) = 1 \text{ and } \lim_{t \to \infty} R(t) = 0.$$

Equation 2.7 describes the steady-state availability of a system, where *up* is the system uptime, and *down* is the system downtime. It also can be represented as the association between the mean time to failure (MTTF) and the mean time to restore or repair (MTTR), Equation 2.8 and Equation 2.9, respectively (AVIZIENIS; LAPRIE; RANDELL, 2001; MACIEL et al., 2012; MACIEL, 2023b; MACIEL, 2023a). Assessing the availability using MTTF and MTTR, the result will always be a number between 0 and 1. For example, if the system's availability is 0.997, the system is working 99.70% and not working 0.30% of the time. The steady-state availability is expressed as

$$A = \frac{E(up)}{E(up) + E(down)}, \tag{2.7}$$

where $E(up)$ and $E(down)$ are the expected uptime and the expected downtime, respectively. The $MTTF$ may be expressed as

$$MTTF = \int_0^\infty R(t) dt. \tag{2.8}$$

The $MTTR$ required to achieve as steady-state availability may be estimated by

$$MTTR = MTTF \times \frac{1 - A}{A}. \tag{2.9}$$

As the $E(up) = MTTF$ (Mean Time to Failure) and $E(down) = MTTR$ (Mean Time to Repair), then

$$A = \frac{MTTF}{MTTF + MTTR}. \tag{2.10}$$

The availability can also be represented by the number of nines ($\#9s$), which is estimated by

$$\#9s = -\log(1 - A). \tag{2.12}$$

Table 1 depicts the number of nines in terms of yearly downtime (ÖHMANN; SIMSEK; FETTWEIS, 2014; MACIEL, 2023b; MACIEL, 2023a).

Table 1 – Service availability in number of nines.

# of 9's	**Avail. (%)**	**System Type**	**Downtime (year)**
1	90	unmanaged	5 weeks
2	99	managed	4 days
3	99.9	well-managed	9 hours
4	99.99	fault-tolerant	1 hour
5	99.999	high-availability	5 minutes
6	99.9999	very high-availability	30 seconds
7	99.99999	ultra availability	3 seconds

Source: Maciel et al. (2023).

In order to establish a relationship between the hardware resources and the overall system performance, relying solely on pure availability models is insufficient. To address this limitation, the capacity-oriented availability (COA) metric was introduced. The capacity-oriented availability metric evaluates how effectively the system delivers its services, taking into consideration both availability and unavailability states and their impact on the service provided (MELO et al., 2017; Sousa et al., 2015; MACIEL, 2023b; MACIEL, 2023a).

The capacity-oriented availability (COA) can be computed using Equation 2.13, where pc_i represents the number of available resources in a given state s_i. Additionally, π_i denotes the steady-state availability for the state s_i, while US represents a set containing all available states. Furthermore, MPC corresponds to the maximum processing capacity of the system (MATOS et al., 2017a; MACIEL, 2023b; MACIEL, 2023a). By employing the COA metric, we gain a comprehensive understanding of the system's performance in terms of resource allocation and its impact on the delivered services.

$$\text{COA} = \frac{\sum_{s_{i \in US}} pc_i \times \pi_i}{\text{MPC}} \tag{2.13}$$

2.7 PERFORMANCE EVALUATION

The primary objective of system administrators is to enhance system performance while minimizing costs. Performance evaluation plays a crucial role in assessing a system's behavior based on specific metrics. Prior to conducting a performance evaluation, engineers must determine the appropriate technique to employ. Three commonly used methods include analytical modeling, simulation, and measurement (JAIN, 1990; MACIEL, 2023b; MACIEL, 2023a).

Analytical models utilize closed-form equations to predict and analyze the behavior of a system (GOKHALE; TRIVEDI, 1998; MACIEL, 2023b; MACIEL, 2023a). These models provide an abstraction and representation of computer systems and are generally more cost-effective compared to other methods like measurements, as they do not require a system prototype to be implemented (JAIN, 1990). By employing analytical models, system administrators can identify potential bottlenecks within their infrastructure by varying input values and observing their impact on system performance (PEREIRA et al., 2020; MACIEL, 2023b; MACIEL, 2023a). However, it is important to note that analytical models may become overly complex and difficult to interpret as the number of system states increases. Consequently, resource exhaustion can occur when dealing with excessively intricate equations. Furthermore, there are instances where closed-form solutions are nearly impossible or highly costly to develop (JAIN, 1990; MACIEL, 2023b; MACIEL, 2023a).

The measurement method for performance evaluation is dependent on the existence of a system or prototype. This approach involves monitoring a system and drawing conclusions based on the obtained results. To effectively employ measurement-based methods, it is essential to create a realistic environment where the system can be monitored under heavy load. Additionally, careful analysis of the workload is necessary to ensure that the server is appropriately stressed, as the selection of measurement strategies and tools depends on the workload characteristics (LILJA et al., 2005; MACIEL, 2023b; MACIEL, 2023a).

Numerous tools are available to support performance evaluation; however, it is important to note that these tools can potentially alter the behavior of the system being evaluated. The degree of disturbance introduced by a measurement tool increases with the amount of information and functionality it provides. Consequently, the more comprehensive the tool, the greater the potential disturbance it may cause in the environment (LILJA et al., 2005; MACIEL, 2023b; MACIEL, 2023a). This disturbance introduced by measurement tools can compromise the reliability of the collected data. Therefore, measurement tools that operate based on events are considered more reliable, as they only initiate measurements when specific events occur. However, in situations where events occur frequently, these event-based tools may introduce more disturbance compared to other tools that rely on sampling (LILJA et al., 2005; MACIEL, 2023b; MACIEL, 2023a).

The primary limitation of the measurement method is its reliance on having at least a prototype system to perform measurements. Additionally, monitoring activities are sus-

ceptible to errors. If the system is still in the conceptual stage, the available techniques for performance evaluation are limited to analytical modeling and simulation (JAIN, 1990; MACIEL, 2023b; MACIEL, 2023a).

Simulations in performance evaluation can be resource-intensive as each run of the simulation represents a single sample point of the system's behavior. Consequently, engineers need to conduct multiple runs to obtain a sufficient sample space. A run refers to the execution of the system from start to finish. Since the simulation generates different outputs for each execution, it does not provide an exact result but rather yields a confidence interval representing the range of possible outcomes (JAIN, 1990).

Simulations are typically employed when the assumptions made by analytical models are not applicable or suitable for the system under evaluation (PEREIRA et al., 2020). Unlike measurement-based methods, simulation strategies rely on abstract models of the system, eliminating the need to construct an entire real environment for evaluation (MENASCE et al., 2004). Consequently, the models used in simulations are designed to capture the essential characteristics of the environment. The level of complexity and abstraction employed in these models may vary depending on the specific system being studied (LILJA et al., 2005).

On the other hand, analytical models are mathematical representations used to describe and predict system behavior (GOKHALE; TRIVEDI, 1998). These models enable the abstraction and characterization of the performance of computer systems, making them a cost-effective alternative to other methods. Analytical models calculate the behavior of a system over a defined time frame (GOKHALE; TRIVEDI, 1998). They are developed to understand system behavior, measure performance, and predict the behavior of specific system components. By introducing changes to various system components, analytical models can identify bottlenecks within the environment (CALIRI, 2000; MACIEL, 2023b; MACIEL, 2023a).

The performance metrics used in this thesis are:

- Utilization is resource usage, which is a term used to describe how much the resource is working. Resources can be monitored to see how much of their capacity is in use. Excessive usage for certain systems can be a cause for concern. The percentage of usage indicates how much of the resource capacity is currently in use;

- Response time is the time a service takes to respond to various types of requests. Response time is a function of load intensity, which can be measured in terms of arrival rates (such as requests per second) or the number of concurrent requests. QoS takes into account not only the average response time but also the percentile (95th percentile, for example) of the response time;

- Discard rate is specified as the fraction of discarded requests per unit of time since the beginning of the session;

- Waiting time is the time interval that one waits after placing a request for a service and before the service actually occurs.

These performance metrics play a crucial role in identifying bottlenecks within a system and have a direct impact on the user's perception of performance (CAMPOS et al., 2015; MACIEL, 2023b; MACIEL, 2023a). When applying queueing theory to model the performance of certain systems, it is possible to generalize their behavior. Specifically, the performance metrics for an $M/M/m/B$ queue with finite buffers can be represented using the following equations (JAIN, 1990; MACIEL, 2023b; MACIEL, 2023a).

$$\rho = \frac{\lambda}{m\mu} \tag{2.14}$$

$$\pi_n = \begin{cases} \dfrac{(m\rho)^n}{n!}\pi_0, & n = 1, 2, \ldots, m-1 \\ \dfrac{\rho^n m^m}{m!}\pi_0, & n = m, m+1, \ldots, B \end{cases} \tag{2.15}$$

$$\pi_0 = \left[1 + \frac{(1-\rho^{B-m+1})(m\rho)^m}{m!(1-\rho)} + \sum_{n=1}^{m-1}\frac{(m\rho)^n}{n!}\right]^{-1} \tag{2.16}$$

where π_n represents the probability of the system being in the state n. On the other hand, π_0 represents the probability of the system being in the state 0.

$$U = \rho(1 - \pi_B) \tag{2.17}$$

$$RT = \frac{\sum_{n=1}^{B} n\pi_n}{\lambda(1-\pi_B)} \tag{2.18}$$

$$WT = \frac{\sum_{n=m+1}^{B} (n-m)\pi_n}{\lambda(1-\pi_B)} \tag{2.19}$$

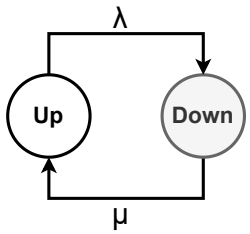
$$DR = \lambda\pi_B \tag{2.20}$$

2.8 MARKOV CHAIN

A Markov Chain is a state-based model, in which some future state only depends on the current states and not on the previous states (TRIVEDI; BOBBIO, 2017; MACIEL, 2023b; MACIEL, 2023a). Thus, with this formalism, it is possible to describe the functioning of a system through a set of states and transitions. Markov chains are useful mathematical models for the description of statistical analysis with time values in their parameters, also known as a stochastic process.

A stochastic process $X(t)$, $t \in T$ is a sequence of random variables defined over the same probability space, indexed by a time parameter ($t \in T$) and assuming values in the state space ($s_i \in S$) (CASSANDRAS; LAFORTUNE et al., 2009; MACIEL, 2023b; MACIEL, 2023a). Thus, if a set T is discrete, in other words, enumerable $X(t), t = 1, 2, 3, ...$, the process is a discrete process or discrete time. If T is a continuous set, we have a continuous parameter process or continuous time. The stochastic process is classified as a Markov process if for all $t_0 < t_1 < ... < t_n < t_{n+1}$ and for all $X(t_0), X(t_1), X(t_2), ..., X(t_n), X(t_{n+1})$ the conditional distribution of $X(t_{n+1})$ depends only on the last value $X(t_n)$, and not on previous values $X(t_0), X(t_1), X(t_2), ..., X(t_{n-1})$. On other words, for any real number $X_0, X_1, X_2, ..., X_n, X_{n+1}, P(X_{n+1} = s_{n+1}|X_n = s_n, X_{n-1} = s_{n-1}, ..., X_0 = s_0) = P(X_{n+1} = s_{n+1}|X_n = s_n)$, which is the Markov property (BOLCH et al., 2006; STEWART, 2009; MACIEL, 2023b; MACIEL, 2023a).

Figure 4 – Continuous-time Markov chain state diagram of a repairable system.



Source: Elaborated by the author.

In this work, we focus on Continuous-time Markov chain (CTMCs); hence, to find the probability of each state in a Markov chain, we group the transition rates into a matrix $\mathbf{Q}$, which is called the infinitesimal generator of a CTMC. For example, Figure 4 depicts a CTMC diagram of a repairable system, where we have a failure rate λ and a repair rate μ. In other words, the transition from Up to Down represents the failure with rate λ and the transition from Down to Up represents the repair with rate μ. Therefore, we have the following infinitesimal generator matrix of the CTMC in Figure 4:

$$\mathbf{Q} = \begin{bmatrix} -\lambda & \lambda \\ \mu & -\mu \end{bmatrix}.$$

Assuming we want to find the probability of the state Up and Down, π_u and π_d, respectively; therefore, we apply the Kolmogorov differential equations assuming the steady-state solution exists (KOLMOGOROFF, 1931; MACIEL, 2023b; MACIEL, 2023a):

$$\begin{bmatrix} \frac{d\pi_u}{dt} & \frac{d\pi_d}{dt} \end{bmatrix} = \begin{bmatrix} \pi_u & \pi_d \end{bmatrix} \begin{bmatrix} -\lambda & \lambda \\ \mu & -\mu \end{bmatrix}. \tag{2.21}$$

From the Equation 2.21, we can obtain the following scalar equations:

$$\begin{cases} \frac{d\pi_u}{dt} &= -\lambda\pi_u + \mu\pi_d, \\ \frac{d\pi_d}{dt} &= \lambda\pi_u - \mu\pi_d. \end{cases} \tag{2.22}$$

Solving the algebraic set of the equations, we can find the steady-state solution given by:

$$\begin{cases} \pi_u &= \frac{\mu}{\lambda+\mu}, \\ \pi_d &= \frac{\lambda}{\lambda+\mu}. \end{cases} \tag{2.23}$$

Therefore, π_u and π_d are the probability of the state Up and Down, respectively. For this specific example, π_u and π_d also mean the availability and unavailability of the system (MACIEL, 2023b; MACIEL, 2023a).

2.9 FAULT TREE DIAGRAM

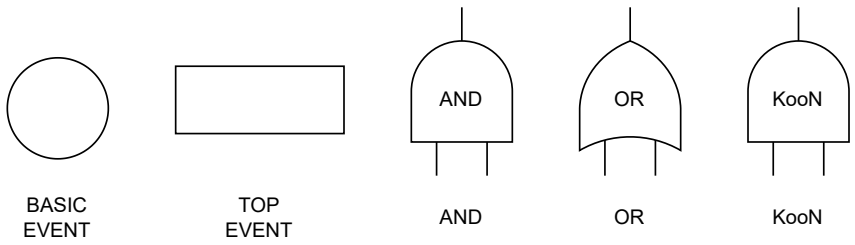
The Fault Tree (FT) is a combinatorial model for availability and reliability evaluation (MACIEL, 2023b; MACIEL, 2023a). If we assume that a random variable $y_i(t)$ indicates the state of the component i, thus:

$$y_i = \begin{cases} 1, \text{ if the component } i \text{ is faulty at time } t \\ 0, \text{ if the component } i \text{ is operational at time } t. \end{cases} \tag{2.24}$$

As we can see in Equation 2.24, Fault Tree (FT) models are failure oriented. In other words, we adopt Fault Tree models to build and evaluate the set of events depicting the system's failure (MACIEL, 2023a).

In Fault Tree (FT) models, we have two types of symbols: events and gates. Events can be characterized as failure or error in a component, and gates describe the relationship between input and output events (MACIEL, 2023b). Figure 5 shows the basic symbols of FT diagram.

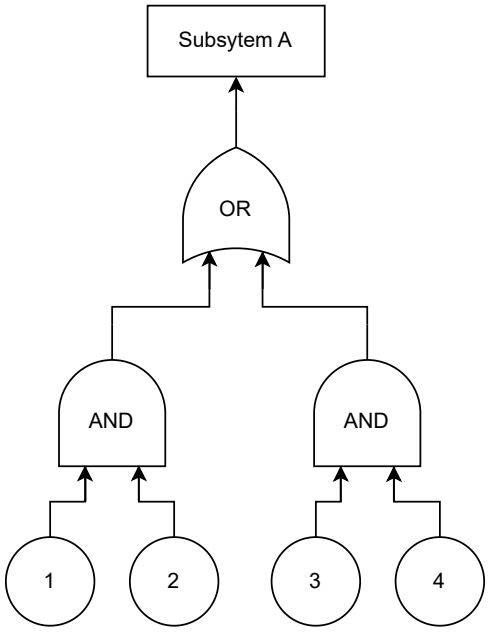
Figure 5 – Fault Tree basic symbols.



Source: Elaborated by the author.

Figure 6 depicts an example of a Fault Tree model, where if the Top event is 1, then it means the system is faulty. The circles (i.e. 1, 2, 3, and 4) are basic events, which can represent a failure in some component of the system.

Figure 6 – Fault Tree diagram example.



Source: Elaborated by the author.

In FT diagrams, we can combine gates and events to model complex system designs. However, we also can model subsystems and combine them to represent the whole system. From the graphical representation of the FT diagram, we derive the logical function, and from the logical function, we can obtain the structure function. The structure function allows us to calculate the reliability and availability of systems (TRIVEDI et al., 1996; MACIEL, 2023a).

2.10 RELIABILITY BLOCK DIAGRAM

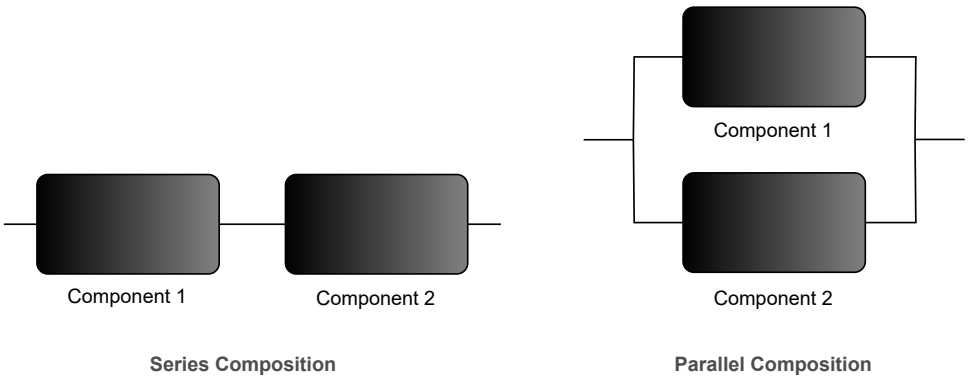
In the previous section, we presented a combinatorial failure-oriented model. On the other hand, in this section, we will present the Reliability Block Diagram (RBD), which is a combinatorial success-oriented model. If we assume that a random variable $y_i(t)$ indicates the state of the component i, thus:

$$y_i = \begin{cases} 1, \text{ if the component } i \text{ is operational at time } t \\ 0, \text{ if the component } i \text{ is faulty at time } t. \end{cases} \tag{2.25}$$

In an RBD model, each component of a system is a block and can be linked to others through arcs. The possible combinations in RBD models are series, and parallel. In the series composition, if one component fails, the whole system fails. In other words, the system is operational only if all components are operational, otherwise, the system is not functional. On the other hand, in the parallel composition, the system is working if at least one component is functioning. The parallel composition can be used to illustrate the

redundancy of components. It is worth pointing out that if we use parallel composition to depict the redundancy, we are representing the hot-standby redundancy (TRIVEDI et al., 1996; MACIEL, 2023b). Figure 7 depicts the two compositions.

Figure 7 – Reliability Block Diagram example.



Source: Elaborated by the author.

In RBD models, we can combine blocks to model complex system designs using different compositions. However, we also can model subsystems and combine them to represent the whole system. From the graphical representation of the RBD, we derive the logical function, and from the logical function, we can obtain the structure function. The structure function allows us to calculate the reliability and availability of systems (TRIVEDI et al., 1996; MACIEL, 2023a).

Within the domain of serial components, where the failure of a single element impacts the entire system, RBDs offer a means to depict the progression of failures throughout the system. By representing individual components as blocks in the diagram, interconnected appropriately, we can effectively examine the influence of component failures on the overall system's availability.

In our approach, we employ the RBD model to consolidate multiple components into a single entity. This consolidation involves combining the MTTF and MTTR values of the constituent components, resulting in an equivalent component within the RBD, as highlighted in previous research studies (MACIEL, 2023b; MACIEL, 2023a). This approach streamlines the model, rendering it more manageable and facilitating the analysis of system availability. However, it is crucial to acknowledge that this condensation assumes that failures among the individual components within the series are independent of one another.

2.11 PARAMETRIC SENSITIVITY ANALYSIS

Parametric sensitivity analysis is a valuable approach used to identify the factors in a model that exhibit the highest impact on the output measure with the smallest variation

(HAMBY et al., 1994). The primary objective of parametric sensitivity analysis is to detect performance, availability, and reliability bottlenecks within the system by assessing the effects of input variations on the corresponding outputs (HAMBY et al., 1994).

Various techniques can be employed to conduct parametric sensitivity analysis, including factorial experimental design (JAIN, 1990), correlation analysis, and regression analysis (CHATFIELD et al., 2010). One straightforward method involves systematically varying a single parameter while holding all others constant, allowing for the computation of changes in the model's output and the creation of a sensitivity ranking (HAMBY et al., 1994). Another useful approach is differential analysis, which involves calculating partial derivatives of the desired measures with respect to each input parameter. This method serves as the foundation for many parametric sensitivity analysis techniques (HAMBY et al., 1994).

The sensitivity of a particular measure Y to a specific parameter θ can be determined using Equation 2.26. Additionally, Equation 2.27 provides the scaled sensitivity, which aids in comparing the relative impact of different parameters on the measure of interest.

$$S_\theta(Y) = \frac{\partial Y}{\partial \theta} \tag{2.26}$$

$$SS_\theta(Y) = \frac{\lambda}{Y}\frac{\partial Y}{\partial \theta} \tag{2.27}$$

Partial derivatives are extensively used to perform sensitivity analysis. However, they may not be the best approach in some cases. For instance, when the parameters of a model under analysis are integer values, the partial derivatives method cannot be applied because it is designed for input values in a continuous domain (CHATFIELD et al., 2010; MATOS et al., 2016). Partial derivatives are not used in simulation either, because they might not be calculated due to the lack of closed-form equations or systems of equations to solve.

Another way to perform sensitivity analysis is by calculating the percentage difference when varying one input parameter from its minimum to its maximum value. This approach may be used as an alternative for partial derivatives. Equation 2.28 shows the expression for this approach, where $min\{Y(\theta)\}$ and $max\{Y(\theta)\}$ are the minimum and maximum output values respectively. These values are computed when varying the parameter θ over the range of its n possible values of interest.

$$S_\theta(Y) = \frac{max\{Y(\theta)\} - min\{Y(\theta)\}}{max\{Y(\theta)\}} \tag{2.28}$$

where

$$max\{Y(\theta)\} = max(Y(\theta_1), Y(\theta_2), ..., Y(\theta_n)) \tag{2.29}$$

and

$$min\{Y(\theta)\} = min(Y(\theta_1), Y(\theta_2), ..., Y(\theta_n)) \tag{2.30}$$

If Y consistently increases and never decreases or consistently decreases and never increases, so only the extreme values of θ may be used to compute $min\{Y(\theta)\}$, $max\{Y(\theta)\}$, and $S_\theta(Y)$ (MACIEL, 2023b; MACIEL, 2023a).

2.12 FINAL REMARKS

This chapter has presented the essential theoretical foundations to familiarize the readers with the fundamental concepts underlying this research. By discussing the background of edge, fog, and cloud computing paradigms, along with the concepts of availability and performance evaluation, we have provided a comprehensive understanding of the basis for the proposed work. Furthermore, the significance of sensitivity analysis has been elucidated, as it will be employed in subsequent case studies to enhance system availability.

3 RELATED WORKS

The significance of novel paradigms, such as edge, fog, and cloud computing, in facilitating Internet of Things (IoT) infrastructures has gained considerable attention from both academic and industrial sectors. In academia, the exploration of interoperability among edge, fog, and cloud environments for IoT infrastructures, aiming to enhance various Quality of Service (QoS) measures, such as availability and performance, has been the subject of recent studies. In this chapter, we aim to provide a comprehensive review of the current research landscape in terms of availability and performance evaluation, specifically focusing on the utilization of modeling techniques within edge, fog, and cloud computing environments. We divided the related works into three sections: Edge, Fog, and Cloud Computing. To present a cohesive overview, we have organized the relevant works into a tabular format, as depicted in Table 2, where the studies are classified based on the type of infrastructure, evaluation method employed, and the adoption of analytical modeling.

3.1 EDGE COMPUTING

Boukerche et al. (2020) introduced a data retrieval approach for computation offloading in Vehicular Edge Computing (VEC). Their strategy begins with the premise that when the vehicle finishes the computation, it will send the processed data to the sender Road-Side Units (RSU). The authors also state that depending on the speed and the processing time of the task, the vehicle's location may be different from when the task was received; therefore, they proposed a model of the computation offloading system, which analyzes the capacity to simulate the components that enhance its performance and availability in a vehicular edge computing environment (BOUKERCHE; SOTO, 2020).

Gamatié et al. (2019) introduced an original asymmetric multi-core infrastructure connected to a programming model and workload control to increase the availability and performance of edge environments. This infrastructure combines ultra-low power cores dedicated to parallel workloads for high throughput and a high-performance core. Even though their design resembles a CPU-GPU heterogeneous combination, their proposal is more flexible because the GPU is only used for regular parallel workloads (GAMATIÉ et al., 2019).

Gorbenko et al. (2019) introduced analytical models and practical guidance to developers of distributed fault-tolerant systems allowing them to predict the availability of IoT environments. Their work states that the proposed models will help developers analyze the trade-off between consistency, availability, and latency during system design and operation. Although the authors provided an availability model, they did not consider the performance aspects of such environments (GORBENKO; ROMANOVSKY; TARASYUK,

2019).

A data analytics framework to enhance the efficiency of decision-making regarding the availability of fog environments was introduced by Sanyal et al. (2018). The authors also showed an uncertainty model for IoT sensor data based on Shannon's entropy. The data aggregation and preprocessing scheme proposed may adequately eliminate IoT data uncertainties while maintaining data dynamics. They also stated that data restoration from the partial sets of raw data before the subspace tracking significantly benefits similar methods. Their approach does not demand any previous knowledge about the outliers or the missing values and may work on a fully and partially collected dataset (SANYAL; ZHANG, 2018).

Li et al. (2019) used gray-Markov chains as an analytical model and measurement to enhance the data availability of edge computing environment connected to a traditional cloud system. They had worked out with replicas managed by their proposed algorithm and could improve both performance and data availability by bringing the data closer to the user. The authors also presented some scenarios that could be applied, some as manufacturing, smart cities, and augmented reality (LI et al., 2019).

Aldaug et al. (2022) proposed an analytical model and the solution approach for QoS evaluation of fault-tolerant load balancer, also, QoS evaluation of web servers with mobility issues in fog computing. The authors focused on availability evaluation and how faults degrade SLAs (ALDAĞ; KIRSAL; ÜLKER, 2022).

An uplink transmission approach in IoT powered by energy harvesting was presented by Sun et al. (2017) to permit each node to transmit data through heterogeneous networks, where the uplink transmission approach's primary goal is to improve communication reliability and availability. The authors also proposed performance metrics to analyze outage probability and ergodic capacity and analytical models (SUN; LIU, 2017).

A simulation-based optimization approach was introduced by Huang et al. (2020) for REliability-Aware Service compositiON (REASON) for edge computing paradigm. The authors employ Stochastic Petri Net (SPN) to build the atomic service model for reliability-aware performance evaluation, which may model the dynamics of task arrivals, service procedures, failures, and recoveries. They also introduced a model combination scheme to express the complex service composition process, where scheduling between multiple layers and service collaboration inside or beyond layers can be dynamically modeled (HUANG; HUANG; LIN, 2020).

Miao et al. (2021) presented a quantitative performance analysis of vehicular edge computing (VEC) systems to cope with the requirements of vehicular applications. The authors proposed analytical models to evaluate the performance of vehicular edge computing systems (MIAO et al., 2021).

A theoretical approach was introduced by Li et al. (2017) to reliability-aware performance evaluation of IoT services. The authors proposed a generalized stochastic Petri net

(GSPN) to express the IoT environments' variability, including task scheduling, queueing, request arrivals, failures, repairs, and recoveries; the authors modeled atomic services and comprehensive systems, and they also presented corresponding quantitative analyses. Li et al. (2017) conducted empirical experiments based on real-world data to provide a predictive methodology of performance evaluation of IoT services, taking into account the reliability of these environments (LI; HUANG, 2017).

The Energy Efficient and Failure Predictive Edge Offloading (EFPO) framework were presented by Zilic et al. (2019) to determine the offloading decision policy's feasibility. They model the edge environment with Markov Decision Process (MDP). The results present performance and energy efficiency metrics, besides adopting failure rates to identify that Edge and computational server are less reliable than Edge regular and cloud (ZILIC; ARAL; BRANDIC, 2019).

3.2 FOG COMPUTING

In their study, Battula et al. (2020) conducted an assessment of the resource availability challenge in Fog computing infrastructure with the aim of enhancing the Quality of Service (QoS). Their research introduced a novel method employing a continuous-time Markov chain to effectively represent the Fog infrastructure and facilitate resource selection decisions. To evaluate their proposed model across various scenarios, the authors utilized simulation techniques. The primary objective of their approach was to minimize service provider-paid compensation by improving QoS in terms of availability and subsequently reducing SLA violations. However, it is worth noting that the authors' work did not encompass considerations for edge environments or capacity-oriented availability (BATTULA et al., 2020).

Security challenges in fog computing were addressed by Yakubu et al. (2019). The authors discussed several procedures utilized to solve security dilemmas within the fog-computing environment. In their study, the authors show that most of the security methods used by different researchers are not dynamic sufficient to overcome all the fog security issues, including availability and reliability problems. The main difference between our survey and their work is that this study is more general in the sense of showing several types of research regarding availability and reliability. This study also aggregates the edge and cloud computing paradigm, and not only the fog computing (YAKUBU et al., 2019).

An efficient and flexible architecture that uses SDN, fog computing, and a blockchain paradigm to capture and analyze IoT income data of end devices was presented by Sharma et al. (2017). Although their proposal focuses on performance evaluation, it also considerably increased the provided services' availability and reliability. Their distributed fog environment used SDN and blockchain paradigm to bring computing resources to the edge devices, so the traffic has a minimal end-to-end delay between IoT sensors and actuators and computing power (SHARMA; CHEN; PARK, 2017).

A Deterministic and Stochastic Petri Net (DSPN) approach was presented by Andrade et al. (2020) for evaluating Fog–Cloud IoT environments composed of hundreds of physical Things. The author's approach evaluates the trade-offs of many performability metrics such as utilization, response time, throughput, and availability and may help system designers choose the most suitable Fog–Cloud IoT environment. The results show that adopting a Fog node would improve availability and performance in some instances (ANDRADE et al., 2020).

Yousefpour et al. (2019) proposed the *deepFogGuard*, a deep neural network (DNN) augmentation scheme for making the distributed DNN inference task failure-resilient. The simulation considered mainly different reliability settings for cloud, fog, and edge environments. The results presented demonstrated that the *deepFogGuard* is more efficient than a DNN that is not trained for failure resiliency (YOUSEFPOUR et al., 2019a).

Barzegaran et al. (2021) introduced an evaluation of real-time applications and their availability. The authors used simulation to evaluate the availability of real-time applications on Fog computing environments (BARZEGARAN; POP, 2021).

3.3 CLOUD COMPUTING

In another work, Facchinetti et al. (2019) presented a lightly-pervasive well-being service called IPSOS Assistant for Mobile Cloud Computing (MCC); this service is to manage emergencies in indoor spaces. Their proposal combines indoor positioning, context monitoring, visualization, and social collaboration to assist users in indoor workplaces in case of personal and environmental emergencies. The authors also address the availability and reliability of their proposal (FACCHINETTI; PSAILA; SCANDURRA, 2019).

Already, Jia et al. (2020) introduced an edge computing environment that supports resource sharing according to several scenarios. The authors also proposed a trust property, where they subdivided the system based on the heterogeneity and dynamics for increasing the availability of the services (JIA; LIN; DENG, 2020).

A novel optimization model to investigate and minimize the total Cloud cost to serve requests to increase the service's availability and reliability was presented by Sharkh et al. (2018). In cases where IoT service providers give more importance to the cost, and the providers are limited by network capacity, they can use fog environments to serve all requests that arrive in the service. Their results show that regarding high-value requests, the minimal cost may be achieved by giving more importance to the Cloud nodes; in other words, the request must be sent to the cloud nodes. They also state that in cases where a high percentage of requests have high value, it is preferable to change some of the Fog Node work (SHARKH; KALIL, 2018).

Mobile-cloud computing was addressed in Angin et al. (2015), where the authors proposed a mobile-cloud methodology based on autonomous agent-based application modules. Their main goals are to introduce an operative and easy-to-adopt MCC model.

They also introduced a dynamic availability-performance estimation model that permits the tracking and relocation of offloaded computation; this dynamic model tries to achieve optimal availability and performance under varying mobile-cloud contexts (ANGIN; BHARGAVA; JIN, 2015).

A layered platform-centric model from the cloud's perspective was introduced by Guan et al. (2016) to provide a full picture of handling the offloading issue in the multi-user multi-core mobile cloud environment. A platform-centric offloading scheme was formulated and evaluated in this model, aiming to improve task execution efficiency and energy reduction during offloading and increase the service's availability (GUAN; GRANDE; BOUKERCHE, 2016).

An approach for availability analysis of IaaS cloud systems was introduced by Longo et al. (2011), which can generate a quick and scalable response and considers multiple classes of server pools. Their approach is based on modeling the system as a joined interacting Markov chain using sub-models. As a result, the authors elaborated closed-form equations for the sub-models; and regarding the dependencies among the sub-models, the authors solved it by using fixed-point iteration. Their proposal decreases the complexity and solution time for analyzing IaaS clouds (e.g., IaaS Cloud systems' availability composed of thousands of physical servers may be analyzed in seconds) (LONGO et al., 2011).

A stochastic model-driven mechanism to optimize the cost and capacity in an IaaS cloud infrastructure was proposed by Ghosh et al. (2013). Their approach is an optimization framework built throughout the performance and availability models of an IaaS Cloud because it helps the computation of different cost components. The authors also present simulated stochastic algorithms to solve the non-linearity and non-convexity of the overall optimization. Their proposal is quicker than a naive search algorithm while maintaining the solution's optimality (GHOSH et al., 2013).

In another work, Ghosh et al. (2014) proposed a stochastic modeling mechanism to evaluate large IaaS cloud systems' availability. Their work demonstrates a way to overcome scalability problems for a monolithic model through interacting sub-models or simulation. Using the sub-model interactions, the authors produced fast model solutions facilitating scalability without jeopardizing the accuracy. Through simulation, the authors were able to generate results that closely match the monolithic and interacting sub-models technique. The authors also derived closed-form solutions to solve large cloud models faster (GHOSH et al., 2014).

A component-based availability modeling framework called Candy was introduced by Machida et al. (2011). This framework composes a comprehensive availability technique to model cloud services from the perspective of system specifications described using SysML diagrams. The authors also state that their proposal semi-automatically converts the elements of SysML diagrams into model components. To validate their framework, the authors applied their framework in a web application system on an IaaS cloud, and the

generated model is used to evaluate the effectiveness of the automatic scale-up mechanism and failure-isolation zones (MACHIDA et al., 2011).

A sensitivity analysis of mobile cloud availability based on hierarchical analytical models was introduced by Matos et al. (2015). Their results improved the system availability considerably by focusing on a decreased set of components that create a considerable variation on steady-state availability. Their sensitivity analysis ranked the system's components. At the highest positions of sensitivity rankings, it is the battery discharge or the cloud infrastructure components named Infrastructure and Storage Managers; it means that these components must obtain the highest priority to produce improvements in system availability (MATOS et al., 2015).

In another work, Matos et al. (2017) introduced a sensitivity analysis of hierarchical heterogeneous models for Eucalyptus private cloud architectures, which may be used for guiding further improvements to the system availability. They also present closed-form equations that enable the solution of desired metrics and computation of sensitivity indices without the necessity for the RBD and MRM models' numerical solution (MATOS et al., 2017b).

Already, Melo et al. (2018) evaluated both availability and reliability issues regarding service provisioning over cloud computing environments. They used the Mercury tool's script language to represent and evaluate dynamic reliability block diagrams (DRBD) to obtain the related metrics and represent some dependencies between the system's components. Also, Melo et al. (2020) extended those DRBDs to evaluate the availability of hyper-converged cloud computing environments. It is relevant to mention that DRBDs is an extension of common RBDs. However, with the possibility of evaluating and establishing a dependency between the system's components, in both papers, the evaluated environments were managed by OpenStack cloud computing platform, and deployment costs related to service provisioning were considered (MELO et al., 2018).

Melo et al. (2019) evaluated the availability of cloud computing environments hosting a blockchain -as-a-service platform based on Hyperledger Fabric. The authors used RBDs and SPN models created on Mercury Tool to obtain the environment's annual downtime and detect the impact and the bottlenecks among the input parameters over the system's availability by applying difference sensitivity analysis (MELO et al., 2019a).

Liu et al. (2018) proposed monolithic availability models based on Stochastic Reward Net (SRN) formalism in order to evaluate the availability of a cloud computing infrastructure-as-a-service (IaaS) provisioning. The authors also had performed a parametric sensitivity analysis evaluation to identify the components that impact the most on the service provisioning availability considering a cloud data center infrastructure. Two repair routines were also considered on the sensitivity analysis evaluation and pointed out that shared maintenance can have lower costs and keep a similar availability value for individual maintenance applied to each server or site in a cloud data center (LIU et al.,

2018).

A characterization by an availability perspective of the container-based implementation of an IMS system (cIMS) was presented by Di et al. (2019). The authors are interested in recognizing the optimal settings of a cIMS deployment that can provide five-nine availability requirements, meaning that a maximum downtime of 5 minutes and 26 seconds per year is allowed (MAURO et al., 2019).

The work developed by Dantas et al. (2012) proposes hierarchical modeling to represent redundant cloud infrastructure, comparing their availability. The authors consider a high-level model based on RBD representing the Eucalyptus platform subsystems and a low-level model based on Continous Time Markov Chain representing the respective subsystems employing a warm-standby replication mechanism. Dantas et al. (2015) still considers as an extension of the paper, where the authors include the acquisition cost and compare then to the public cloud (DANTAS et al., 2012).

In another work presented by Dantas et al. (2020), the authors estimate the availability and capacity-oriented availability through closed-form equations. The authors compare the approach to using models such as Continuous Time Markov Chains and SPN simulation model, considering execution time and values of metrics obtained with both approaches (DANTAS et al., 2020).

A systematic study of correlations amongst availability, reliability, performance, and energy consumption metrics was presented by Qiu et al. (2017). Their models incorporate Markov models, Laplace-Stieltjes transform, a Bayesian approach, and a recursive method. In their work, the significant R-P-E relationships are examined using their models for estimating expected service time and energy consumption of a cloud service; their models estimate the resource usage under two typical recovery approaches, retrying and checkpointing (QIU et al., 2017).

A modeling approach to evaluate the performance and power consumption of an IaaS cloud while taking into account the availability was introduced by Ataie et al. (2017). They presented a series of SRN models for computing the availability, performance, and power consumption of cloud environments and they also presented two estimated models that are flexible enough to evaluate the metrics of interest in cloud systems. The principal benefit of the fixed-point estimated model proposed by them is its capacity in modeling large cloud environments without state space explosion (ATAIE et al., 2017).

A visual model-based framework for cloud-based PHM using block definition diagram (BDD) and internal block diagram (IBD) was proposed by Mao et al. (2017). The authors used structure diagrams and behavior diagrams to describe the static structure and dynamic states of PHM. They introduced a framework that permits several algorithm implementations of function modules. The authors performed a performance and availability evaluation and proposed a measurement method for their PHM framework (MAO et al., 2017).

An SRN model to understand the method of VM deployment and migration was designed by Liu et al. (2019); the authors proposed a stochastic Petri net model, where they use different guard functions of specific transitions to represent several strategies. They assumed that all inter-event times are exponentially distributed; they also designed three metrics and computation techniques to analyze their algorithms, including energy usage, QoS, and Cost of Migration (LIU et al., 2019).

A utility-based optimization approach was presented by Addis et al. (2013) to considerably increase the availability and performance of cloud servers running virtual machines (VMs). The authors provided a theoretical framework to investigate the circumstances in which a hierarchical method may be applied (ADDIS et al., 2013).

A high-availability component called CHASE was proposed by Jammal et al. (2015) for cloud environments. Using CHASE, the high availability of services is achieved while considering performance and delay demands and redundancies; the authors also consider different failure ranges. An analysis is conducted to provide higher scheduling priorities for critical components than standard ones. The HA-aware scheduler estimates the availability of components using its mean time to failure (MTTF), mean time to repair (MTTR), and recovery time (JAMMAL; KANSO; SHAMI, 2015).

Stochastic availability models were proposed by Santos et al. (2018) to understand how failures in edge devices, fog devices, and cloud infrastructure impacts e-health monitoring system availability. The models are also used to perform sensitivity analysis to understand which components significantly impact e-health monitoring systems' availability. The authors also proposed stochastic performance models integrated with availability models and investigate the impact on performance metrics, such as service time and throughput (SANTOS et al., 2018).

A new reliability problem and, adopting combinatorial optimization techniques, developed a randomized algorithm and a heuristic to evaluate the proposed solution was formulated by Weifa et al. (2020). The experimental results showed that the proposed solution is promising, and its empirical results are superior to the analytical ones (LIANG et al., 2020).

Lee et al. (2019) proposed a simulator to optimize the speed and reliability of a mobile cloud computing environment running over a smartphone cluster. The Mobile MapReduce Task Allocation (MTA) simulator had improved both associated metrics significantly by performing a better allocation process (LEE et al., 2019).

Ever et al. (2019) introduced a stochastic performance and an energy availability evaluation model for WSNs, where they consider node and link failures. Their proposal contemplates an integrated performance and availability approach. In their study, several duty cycling schemes and the medium-access control of the WSNs were also examined in order to include the consequence of sleeping and idle states. Their results reveal the consequences of failures, several queue capacities, and system scalability (EVER, 2019).

Tian et al. (2020) used a data set from Google Cloud to identify, predict, and mitigate the impact of failures over a public cloud environment reliability and efficiency. They had analyzed the operational data and diagnosed the tasks that can impact the most on service provisioning. A predictive mathematical model and a set of simulations showed to be enough to mitigate issues and improve the system's reliability and efficiency (TIAN; TIAN; LI, 2020).

Dehnavi et al. (2019) evaluated the reliability of both hybrid cloud computing, edge and fog environments using the reliability expression, as well as a set of simulation experiments over eight different setups. It is significant to mention that this research focused on the reliability requirements of real-time applications to smart factory environments. Once again, fog and edge's use proved to be a sound alternative to improving cloud-based applications (DEHNAVI et al., 2019).

Huang et al. (2020) evaluated cloud and edge environments, aiming at service provisioning to IoT devices. They had modeled with a heightened graph, a network that connects all components from the IoT devices up to what they had called edge servers nodes and cloud servers nodes. The authors had considered a multi-state distributed network (MDN) and, through the graphs, calculates the probability of data being successfully processed by the nodes (HUANG; LIANG; ALI, 2020).

Our study is relevant because it comprehends several features of the related works, whereas there is an improvement in approaches proposed by them. This work approaches some of the weaknesses of related works to improve the evaluations and capacity planning for edge, fog, and cloud computing environments. This research makes use of the three paradigms and uses analytical and hierarchical models to improve the availability and performance of the infrastructures. Our models aim at improving the service quality. Table 2 summarizes the differences among the related works and our study, where we cover more features of the edge-fog-cloud continuum environments.

Table 2 – Comparison among related Works.

Authors	Analytical Modeling	Avail. and Perf.	Edge	Fog
(SANTOS et al., 2018)		✓	✓	✓
(MELO et al., 2018)	✓	✓		
(SHARKH; KALIL, 2018)		✓		
(YAKUBU et al., 2019)		✓		✓
(LI et al., 2019)	✓	✓	✓	
(DEHNAVI et al., 2019)		✓	✓	✓
(LEE et al., 2019)		✓	✓	
(EVER et al., 2019)		✓		
(LIU et al., 2019)		✓		
(SANYAL; ZHANG, 2018)			✓	
(QIU et al., 2017)	✓	✓		
(FACCHINETTI; PSAILA; SCANDURRA, 2019)		✓		
(MELO et al., 2019b)	✓	✓		
(MAURO et al., 2019)		✓		
(GAMATIÉ et al., 2019)		✓	✓	
(GORBENKO; ROMANOVSKY; TARASYUK, 2019)	✓	✓	✓	
(YOUSEFPOUR et al., 2019a)	✓		✓	✓
(ZILIC; ARAL; BRANDIC, 2019)	✓	✓	✓	
(DANTAS et al., 2020)	✓	✓		
(TIAN; TIAN; LI, 2020)		✓		
(ANDRADE et al., 2020)		✓		✓
(HUANG; LIANG; ALI, 2020)		✓	✓	
(JIA; LIN; DENG, 2020)		✓		
(BOUKERCHE; SOTO, 2020)		✓	✓	
(MELO et al., 2020)	✓	✓		
(LIANG et al., 2020)	✓	✓		
(BATTULA et al., 2020)		✓		✓
(BARZEGARAN; POP, 2021)		✓		✓
(MIAO et al., 2021)	✓	✓	✓	
(ALDAĞ; KIRSAL; ÜLKER, 2022)	✓	✓	✓	
This Thesis	✓	✓	✓	✓

Source: Elaborated by the author.

3.4 FINAL REMARKS

In this chapter, we presented relevant related works for this thesis. Although there are many studies in the literature about the modeling and evaluation mechanism of availability

and performance, most of them cover only one or two paradigms. It may not represent real-world services, which use the whole stack of paradigms. Another problem is that by evaluating only one paradigm, the availability and performance of other components are neglected, which may increase the probability of QoS violations. The impact of improving capacity planning is huge for companies because it reduces the probability of financial loss.

4 METHODOLOGY

In this chapter, we present the methodology employed in our research to develop the proposed models. Our approach draws inspiration from prior works, namely (BRILHANTE et al., 2014; PEREIRA et al., 2018; SOUZA et al., 2013). To ensure clarity and organization, we have structured this chapter into distinct sections: an Overview, Evaluation, Validation, Model Adoption, and Final Remarks.

4.1 OVERVIEW

The methodology is structured into three sections encompassing various activities. These activities include studying the system, monitoring the environment, building models, validating the models, and adopting models to plan new infrastructures. Each activity is represented by a box, and their sequential execution is illustrated in Figure 8.

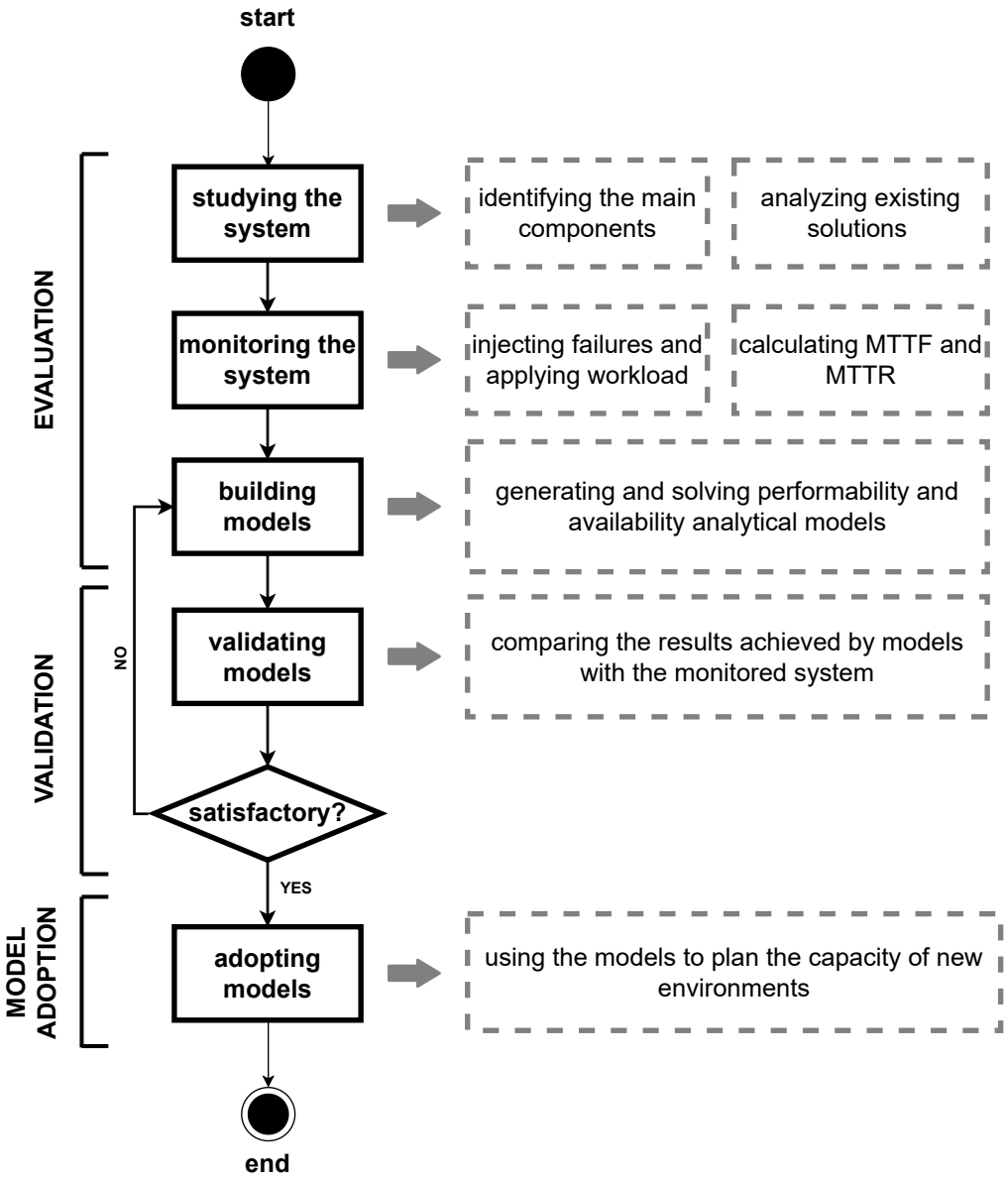
Progression to the subsequent activity is contingent upon the completion of the current one. The rhombus symbolizes an activity that branches into two distinct paths. If the results are deemed satisfactory, the models are considered validated. Conversely, if the results are unsatisfactory, we return to the building models stage, where enhancements are implemented until the models exhibit behavior similar to that of the real system. The dashed rectangles indicate tasks necessary for accomplishing the corresponding activities.

The initial activity involves a thorough examination of the system to gain a comprehensive understanding of its operations. Our focus lies in assessing the availability of the edge and fog environments. To accomplish this, we conducted an extensive review of existing literature and solutions. Utilizing this understanding of the system, we proceeded to construct an experimental environment where we deliberately introduced failures to expedite the identification of faults. By monitoring the actual system, we were able to collect the relevant input values required for the proposed models.

Subsequently, in the monitoring system phase, our attention shifted towards observing the system to capture essential data such as time to failure (TTF), time to repair (TTR), and performance metrics. This allowed us to calculate the system's availability and performance. As part of the model-building activity, we developed analytical models that accurately represent our environment. We then proceeded to test numerous scenarios to evaluate which configurations would enhance both availability and performance.

The validation activity aims to compare the output values of our analytical models with those obtained from monitoring the actual environment. Essentially, we statistically verify whether our models accurately represent the real system. In our case studies, we provide a detailed account of the validation process, including a step-by-step description and an overview of the experimental environment. Once our models yield satisfactory

Figure 8 – Methodology to support our evaluation.



Source: Elaborated by the author.

results, we can employ them to assess the availability and performability of our system under various settings and scenarios, thereby enabling us to plan for new environments.

Satisfactory results in this context refers to the statistical alignment of the model results with the metrics observed in the actual environment. The results encompass the proposed models, the evaluation of availability based on these models, capacity-oriented availability evaluation, performability evaluation, sensitivity analysis, and cost evaluation.

4.2 EVALUATION

The evaluation phase comprises three key components: system analysis, system monitoring, and model development. The primary objective during this stage is to closely observe a real environment, extract the Mean Time to Failure (MTTF) and Mean Time to Repair (MTTR) of the system, and utilize these parameters as inputs for the proposed models.

4.2.1 Studying the system

To effectively plan edge-fog-cloud infrastructures, it is imperative to acquire a comprehensive understanding of their operations, identify key components, and evaluate existing solutions. Thorough analysis of such systems necessitates meticulous attention from the evaluator to avoid errors that could compromise subsequent phases. This preliminary step holds immense significance as it facilitates the acquisition of knowledge regarding applicable techniques that can be adopted and customized.

- Precondition: Familiarity with the concepts of edge, fog, and cloud computing.
- Input: Extensive literature review encompassing scientific papers, books, and websites to comprehend edge-fog-cloud infrastructures and system modeling.
- Action: Development of a foundational environment for subsequent monitoring.
- Post-condition: Attainment of a profound comprehension of edge-fog-cloud infrastructures, establishment of a baseline environment, and proficiency in system modeling techniques.

4.2.2 Monitoring the system

The subsequent phase involves the monitoring of the previously constructed baseline environment. In this stage, a strategy is formulated to effectively monitor the system's availability and performance. Ensuring the monitoring of system availability necessitates the acceleration of the fault-prone processes, as faults are typically unpredictable or occur infrequently over extended periods of time (SOUZA et al., 2013; BRILHANTE et al., 2014; JAMMAL et al., 2018; PEREIRA et al., 2021). When a fault transpires, it induces alterations within the system, resulting in an erroneous state of the environment. Moreover, faults have the potential to propagate throughout the system, subsequently affecting the availability of the provided services.

To facilitate the monitoring of system availability, a fault injector was developed to actively control and monitor our environment during fault events (AGARWAL et al., 2020). The fault injector simulates failures by subjecting the system to stress. To generate random values for Time to Failure (TTF) and Time to Repair (TTR), exponential distribution is

utilized, considering the accelerated Mean Time to Failure (MTTF) and Mean Time to Repair (MTTR) derived from scientific papers and technical reports.

- Precondition: A fully developed baseline environment.
- Input: MTTF and MTTR values for the edge, fog, and cloud components.
- Action: Development of a fault injector and utilization of the fault injector to monitor the baseline system.
- Post-condition: Collection of comprehensive data pertaining to the baseline system.

4.2.3 Building models

Upon conducting the experiment and gathering data from the baseline environment, we proceed to propose analytical models. These models hold significant importance as they enable the evaluation of system concepts that involve a substantial number of components, which would otherwise incur exorbitant expenses if implemented physically. Consequently, analytical models facilitate the assessment of various scenarios by simply modifying the input parameters.

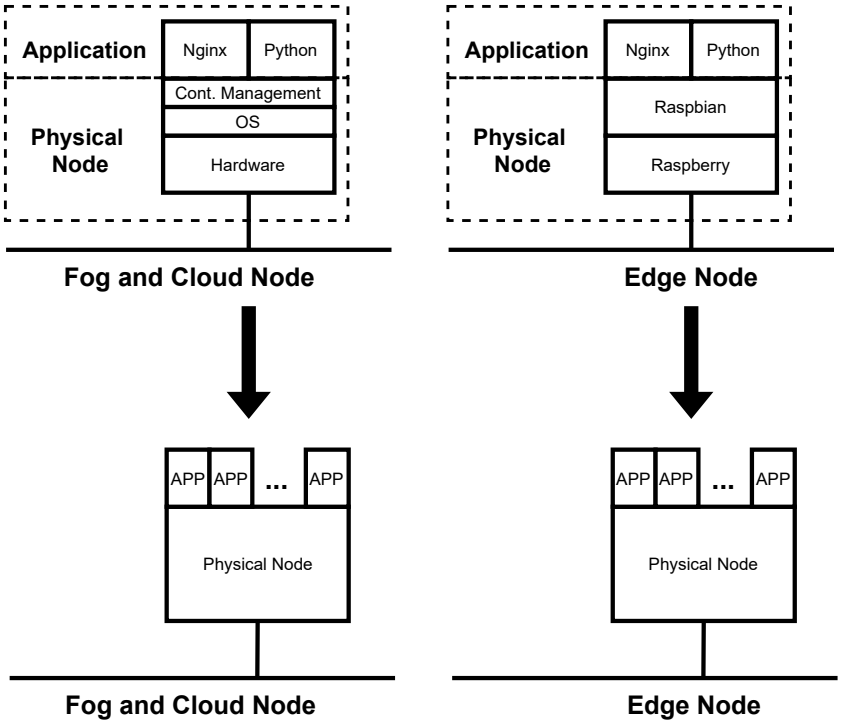
The scalability of analytical models is a key attribute of their significance. They allow for the adjustment of component quantities within a given scenario by manipulating the input parameters alone. This capability enables us to observe the system's behavior under diverse conditions and configurations.

It is important to note that analytical models typically abstract hardware and application-specific details in order to derive closed-form equations (SIVASUBRAMANIAM; RAMACHANDRAN; VENKATESWARAN, 1994). In essence, for feasibility purposes, certain components within the system need to be abstracted within the analytical models (JAIN, 1990). Figure 9 provides a visual representation of the abstractions implemented in constructing the models.

The abstraction process was carried out using the Reliability Block Diagram (RBD) model, which effectively condenses multiple components into a single representative entity. This approach facilitates the simplification of system representation and enables the application of analytical techniques based on Continuous-Time Markov Chain (CTMC) analysis.

- Precondition: Comprehensive understanding of the system under evaluation, availability of data obtained from the baseline environment.
- Input: Profound comprehension of the system's components.
- Action: Development of analytical models based on the system's components.
- Post-condition: Creation of analytical models.

Figure 9 – Basic component abstraction.



Source: Elaborated by the author.

4.3 VALIDATION

This section encompasses the validation of the models. The primary objective is to ascertain whether the proposed models from the previous section yield results that are equivalent to those generated by the existing system.

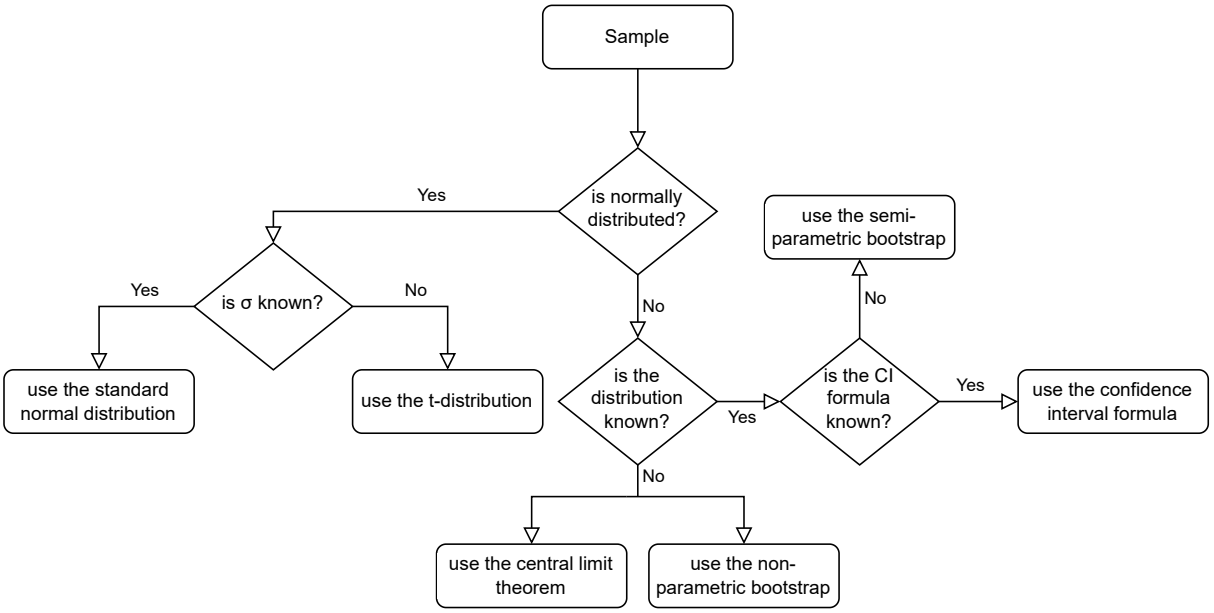
4.3.1 Validating models

In this stage, we undertake the validation of our proposed models. The purpose of validation is to ascertain whether our models accurately represent an existing system. In essence, we analyze whether the results generated by our models align statistically with those observed in the baseline environment. We subject both our models and the baseline system to an evaluation under identical conditions, and if the outcomes are comparable, we consider our models to be validated.

During the validation of our models, we place particular emphasis on comparing the means of our models with those of the existing system. To facilitate this process, Figure 10 (adapted from Maciel et al., 2023) outlines the sequential steps we follow to estimate the mean of the existing system (MACIEL, 2023b; MACIEL, 2023a)

As our data represents merely a sample from a larger population, it becomes imperative to establish confidence intervals that illustrate the probable range of mean values for the population. Therefore, we examine our data and construct the confidence interval for the

Figure 10 – Statistical inference flowchart.



Source: Elaborated by the author.

mean.

Initially, we assess whether our sample conforms to a normal distribution. If it indeed follows a normal distribution and we possess knowledge of the population standard deviation (σ), we can utilize the analytical formula for the standard normal distribution. If we do not know the population standard deviation, we use the student's t-distribution with the appropriate degrees of freedom. However, if the sample does not adhere to a normal distribution, we inquire as to whether we possess information regarding the appropriate distribution. If affirmative, we further inquire if we have access to the analytical formula for estimating the confidence interval. If so, we employ said formula to derive the confidence interval; otherwise, we resort to utilizing the semi-parametric bootstrap approach. In instances where the suitable distribution is unknown, options include utilizing the non-parametric bootstrap method or relying on the central limit theorem.

Upon obtaining the confidence interval, we proceed to evaluate whether the results obtained from our model fall within the plausible range of mean values for the population. If this evaluation yields positive results, we consider our model to be validated. Conversely, if the model's results fall outside the plausible range, adjustments must be made accordingly.

- Precondition: Completion of the analytical model construction.
- Input: Analytical models.
- Action: Conducting rigorous statistical analysis.

- Post-condition: Validated models.

4.4 MODEL ADOPTION

Ultimately, the utilization of our validated models enables us to strategically plan and develop edge-fog-cloud infrastructures. By leveraging these models, we gain the capability to explore and evaluate the behavior of various infrastructural configurations, thus enabling us to propose the most appropriate and well-suited options.

4.4.1 Adopting models

Once our proposed models have been validated, we can proceed with their adoption to assess and evaluate various compositions of the systems. This allows us to investigate and predict the behavior of a system under specific parameter configurations for its components. Additionally, we can assess the required number of redundant components to maintain a desired level of availability, among other considerations.

- Precondition: Validated models.
- Input: Parameters associated with the system's components.
- Action: Configuring the models by setting the respective component parameters and solving analytical equations.
- Post-condition: Attainment of relevant metrics pertaining to the system.

4.5 FINAL REMARKS

This chapter outlines the employed methodology for the development of proposed models and the evaluation of existing systems. By following a structured set of steps, which encompass comprehending the fundamental operation of the system, utilizing proposed models for enhanced infrastructure planning, and providing replicable details, this methodology offers valuable insights to be utilized by fellow researchers.

5 PROPOSED AVAILABILITY AND PERFORMABILITY MODELS

This chapter is organized into four distinct sections, each addressing a specific aspect of the proposed models. The first section focuses on the proposed availability models employing Markov chains. In the subsequent section, we delve into the description of the proposed performability models, which serve to evaluate the impact of availability on infrastructure performance. The third section encompasses the hierarchical model, providing a comprehensive representation of the edge-fog-cloud continuum environments. Lastly, the final section delves into the hierarchical model with fault coverage probability, enhancing the analysis with considerations of fault coverage probabilities.

5.1 AVAILABILITY MODELS

The proposed analytical availability models offer distinct advantages over measurement-based evaluation methods, as they are not only more cost-effective but can also be employed during the system's design phase (GOKHALE; TRIVEDI, 1998). By utilizing analytical models, we gain the ability to predict and plan how a system will behave prior to its deployment, facilitating the development of superior systems.

In this section, we introduce the analytical models that capture the availability metrics specific to edge and fog environments. We begin by presenting a Markov chain, which serves as a generalized representation of our environment. Subsequently, we leverage the Markov chain to derive closed-form solutions that allow for the calculation of availability metrics pertaining to the edge and fog environments.

In Figure 9, we provide an illustration of the component stacks and the abstractions made to facilitate the derivation of closed-form solutions. The abstraction process was carried out using the Reliability Block Diagram (RBD) model, which effectively condenses multiple components into a single representative entity. This approach facilitates the simplification of system representation and enables the application of analytical techniques based on Continuous-Time Markov Chain (CTMC) analysis.

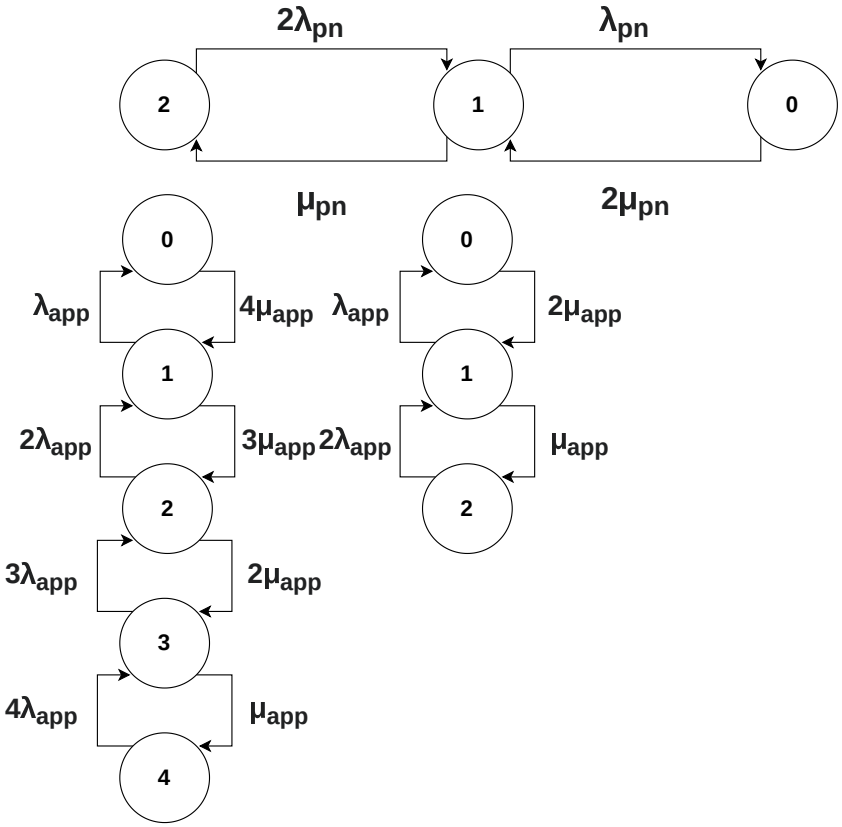
Through the combination of Mean Time to Failure (MTTF) and Mean Time to Repair (MTTR) values of serial components, we create an aggregated equivalent component with a unified representation of availability characteristics. This condensation procedure significantly reduces the complexity associated with modeling individual components, allowing for a more streamlined analysis of the system as a whole, considering the combined availability behavior of the condensed component.

The use of this abstraction technique becomes particularly advantageous when employing CTMC analysis, as it allows for the system to be represented as a set of interconnected states and transitions. By consolidating multiple components into a single entity, the num-

ber of states within the CTMC model is reduced, resulting in a more manageable analysis process. Consequently, analytical solutions, such as steady-state probabilities and performance measures, can be derived, offering valuable insights into the system's availability.

While there exist various modeling techniques to evaluate system availability, many of these methods rely on numerical approximations. However, by employing Markov chains, we can derive cost-effective closed-form equations (JAIN, 1990). In our modeling approach, we utilize a two-level continuous-time Markov chain (CTMC) to represent our environments. The first level represents the physical nodes, such as servers like Dell PowerEdge R240, while the second level pertains to the applications or services. The number of running applications depends on the operational physical nodes. These services can take the form of container instances, virtual machines, or processes, depending on the architectural design. It is important to emphasize that the construction of both levels involves the utilization of multiple independent Continuous-Time Markov Chains (CTMCs), which are subsequently integrated within the closed-form equations.

Figure 11 – Two-physical-node model.



Source: Elaborated by the author.

The system is considered unavailable if it is in state 0 within the first-level CTMC, signifying the absence of any operational physical nodes. Similarly, if the system is in state 0 within any second-level CTMC, it is also regarded as unavailable, regardless of the presence of multiple working physical nodes. As an example, if we intend to model a

system consisting of two similar physical nodes capable of running two applications each, and with sufficient independent repair teams, the corresponding CTMC is depicted in Figure 11.

The model presented in Figure 11 serves as a generalization of the continuous-time Markov chain (CTMC) depicted in Figure 12. Within this Markov chain, M denotes the number of physical nodes, while L represents the number of concurrently running applications. The parameters λ_{pn} and μ_{pn} signify the failure and repair rates of the physical nodes, respectively, whereas λ_{app} and μ_{app} denote the failure and repair rates of the applications. It is important to note that we are considering a homogeneous environment with an adequate number of independent repair teams. Thus, our environment assumes the presence of M similar machines, each with its own independent repair capabilities.

From the Markov chain model depicted in Figure 12, we can calculate various metrics, including the availability (A), K-out-of-N availability (A_{KooN}), unavailability (UA), uptime (UT), downtime (DT), average number of available applications ($ANAPPA$), and capacity-oriented availability (COA).

Please be aware that Figure 12 presents multiple distinct Markov Chains, which have been modeled individually. Each Markov Chain is accompanied by its corresponding matrix $\mathbf{Q}$.

To calculate the system's availability, we sum the probabilities of states in which both the physical nodes and applications are operational. By expressing this calculation in terms of the failure and repair rates (λs and μs), we establish an analytical availability model represented by Equation 5.1.

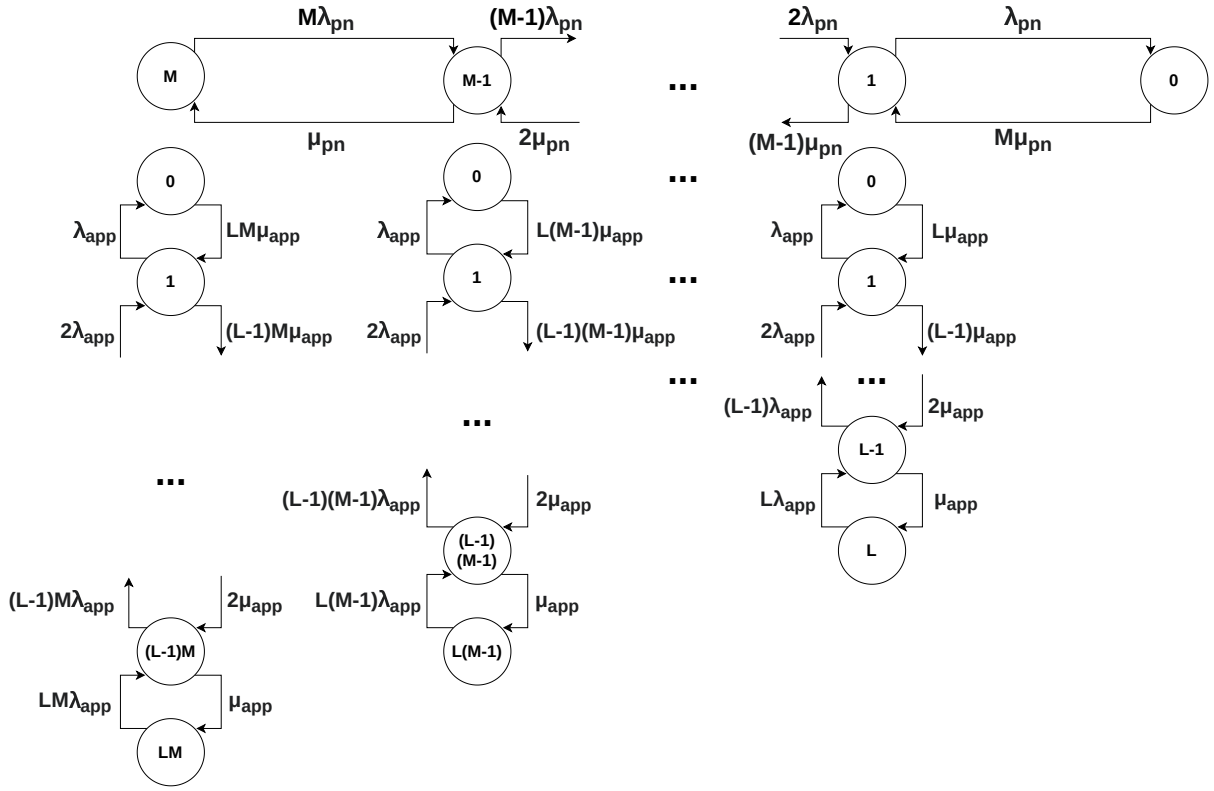
Availability:

$$A = \sum_{i=1}^{M} \left(\left(1 - \frac{\lambda_{app}^{iL}}{(\lambda_{app} + \mu_{app})^{iL}} \right) \times \frac{M!}{i!\ (M-i)!} \times \frac{\lambda_{pn}^{M-i}\ \mu_{pn}^{i}}{(\lambda_{pn} + \mu_{pn})^{M}} \right). \tag{5.1}$$

Availability K out of N:

$$A_{KooN} = \sum_{i=1}^{M} \left(\left(\sum_{j=K}^{iL} \frac{(iL)!}{j!\ (iL-j)!} \times \frac{\lambda_{app}^{iL-j}\ \mu_{app}^{j}}{(\lambda_{app} + \mu_{app})^{iL}} \right) \times \left(\frac{M!}{i!\ (M-i)!} \times \frac{\lambda_{pn}^{M-i}\ \mu_{pn}^{i}}{(\lambda_{pn} + \mu_{pn})^{M}} \right) \right). \tag{5.2}$$

Figure 12 – Availability model for M physical nodes and L applications.



Source: Elaborated by the author.

Average number of applications available:

$$ANAPPA = \sum_{i=1}^{M} \sum_{j=1}^{i \times L} \left(\frac{j(iL)!\ \mu_{app}^{j}\ \lambda_{app}^{iL-j}\ (\lambda_{app} + \mu_{app})^{-iL}}{j!\ (iL-j)!} \times \frac{M!\ \mu_{pn}^{i}\ \lambda_{pn}^{M-i}\ (\lambda_{pn} + \mu_{pn})^{-M}}{i!\ (M-i)!} \right). \tag{5.3}$$

Equation 5.1 represents the availability when at least one out of N applications is operational, where N represents the total number of applications that the system can deliver (i.e., $M \times L$). This implies that the model considers the system available as long as at least one application is running, which is a specific case of Equation 5.2. Conversely, Equation 5.2 represents the availability when exactly K out of N applications need to be operational to consider the system available. In other words, the system is deemed available only if K applications are functioning. Once we have established the availability

models, deriving the unavailability, uptime, and downtime models becomes straightforward, as demonstrated by Equations 5.4, 5.5, and 5.6, respectively. Additionally, from the Markov chain model, we can derive the closed-form solution for the average number of applications available in the environment, as represented by Equation 5.3. Moreover, this information allows us to model the capacity-oriented availability, as depicted in Equation 5.7.

$$UA = 1 - A \tag{5.4}$$

$$UP = A \times T \tag{5.5}$$

$$DT = UA \times T \tag{5.6}$$

$$COA = \frac{ANAPPA}{M \times L} \tag{5.7}$$

Table 3 – The description of the parameters.

Parameter	**Description**
M	Number of physical nodes (i.e., servers)
L	Number of applications running in each server
λ_{pn}	Failure rate of the physical node
λ_{app}	Failure rate of the application
μ_{pn}	Repair rate of the physical node
μ_{app}	Repair rate of the application

Source: Elaborated by the author.

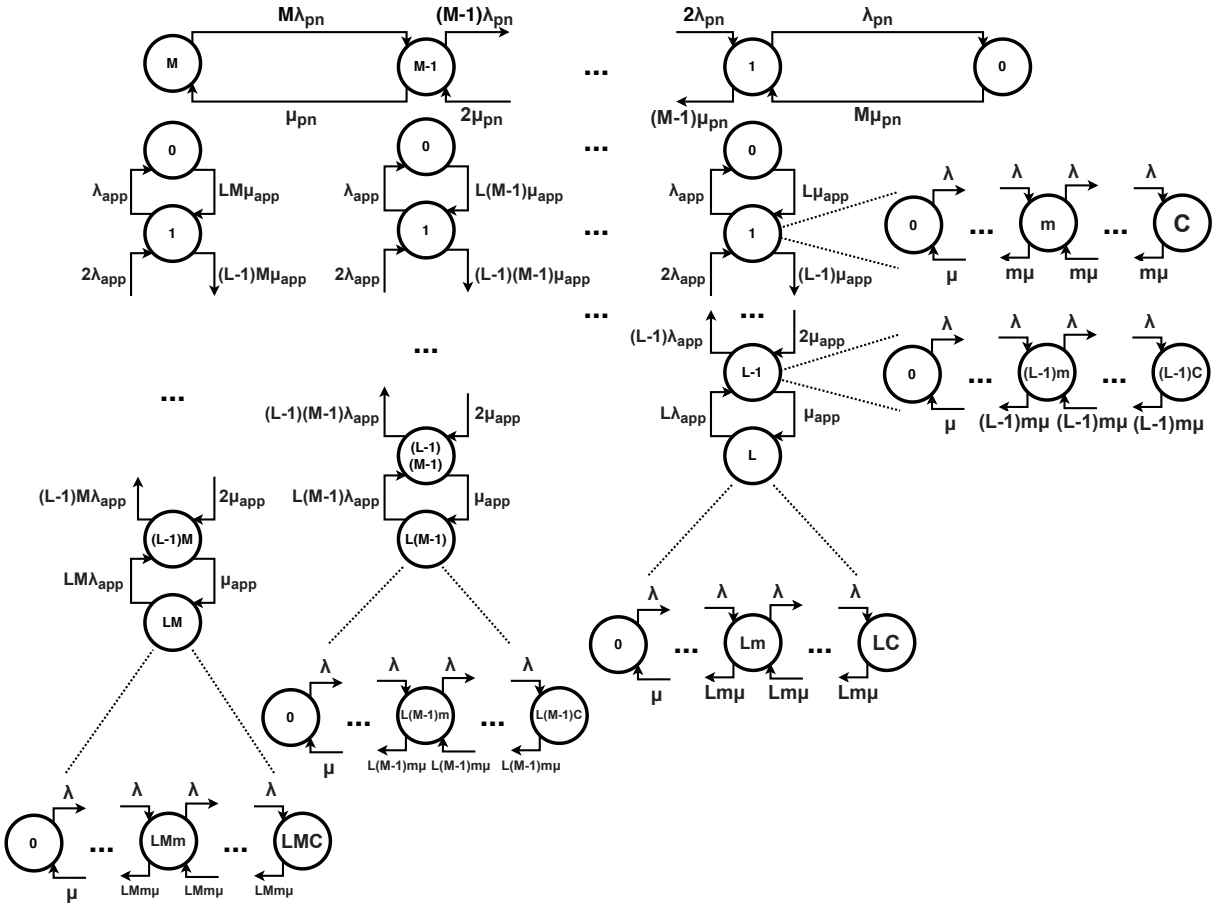
Table 3 provides a description of each parameter present in the closed-form solution. Our decision to employ a continuous-time Markov chain as the modeling technique was deliberate, as it allows for the development of analytical models (CALIRI, 2000). In comparison, other modeling techniques such as Reliability Diagram Models (RBD) and Stochastic Petri Nets (SPN) rely on numerical approximations, which can limit scalability in comparison to analytical models. The advantage of analytical models lies in their ability to readily accommodate multiple scenarios by simply adjusting the input parameters. This flexibility facilitates the observation and analysis of various configurations without significant computational overhead.

5.2 PERFORMABILITY MODELS

By employing analytical performability models, we can forecast how system availability will impact performance, thereby enabling the development of more efficient systems and

preventing unnecessary financial resource allocation. In this section, we introduce the proposed analytical models that capture the performability metrics specific to edge and fog environments. These models serve as an extension of the availability models discussed in the previous section. Initially, we present a hierarchical Markov chain, which provides an abstraction of our environment. Subsequently, we employ the Markov chain to derive closed-form equations that facilitate the calculation of performability metrics associated with the infrastructure.

Figure 13 – Performability model.



Source: Elaborated by the author.

The evaluation of performance and availability in edge and fog environments is a challenging task, primarily due to the diverse nature of IoT applications at the edge and the varying Quality of Service (QoS) requirements in fog environments. Additionally, the edge paradigm is subject to numerous constraints and restrictions, making it more vulnerable to failures. These failures can significantly degrade both the availability and performance, particularly when considering data-intensive applications.

Pure performance models often overestimate a system's capacity to perform or provide services, while pure availability models tend to underestimate this capacity (EVER, 2019).

Therefore, we propose the utilization of a three-level hierarchical Markov chain to conduct a composite evaluation, leveraging both performance and availability (i.e., performability), enabling realistic analysis and optimization of edge and fog environments.

In Section 4, we presented the component stacks and abstractions made to facilitate the derivation of closed-form solutions, as depicted in Figure 9. Our modeling approach incorporates a hierarchical three-level continuous-time Markov chain, where the first and second levels represent the availability of physical nodes and applications, respectively. The third level captures the performance of each application. Consequently, the number of running applications depends on the operational physical nodes, while the capacity is determined by the size of the Markov chain at the third level. The system is deemed unavailable if it is in state 0 within the first-level CTMC, indicating the absence of any operational physical nodes. Similarly, if the system is in state 0 within any second-level CTMC, it is considered unavailable, regardless of the presence of multiple operational physical nodes.

Table 4 – The description of the parameters.

Parameter	**Description**
M	Number of physical nodes (i.e., servers)
L	Number of applications in each server
λ_{pn}	Failure rate of the physical node
λ_{app}	Failure rate of the application
μ_{pn}	Repair rate of the physical node
μ_{app}	Repair rate of the application
C	Maximum number of jobs in the system
m	Number of threads in the system
$\pi_{i\times j\times C}$	Probability of the state $i \times j \times C$

Source: Elaborated by the author.

Table 4 provides a comprehensive summary of the parameters present in the closed-form solution. Within this Markov chain model, M represents the number of physical nodes, while L denotes the number of running applications. The parameters λ_{pn} and μ_{pn} signify the failure and repair rates of the physical nodes, respectively. Similarly, λ_{app} and μ_{app} express the failure and repair rates of the applications. We are also assuming a homogeneous environment with M physical nodes that possess identical characteristics and independent repair facilities.

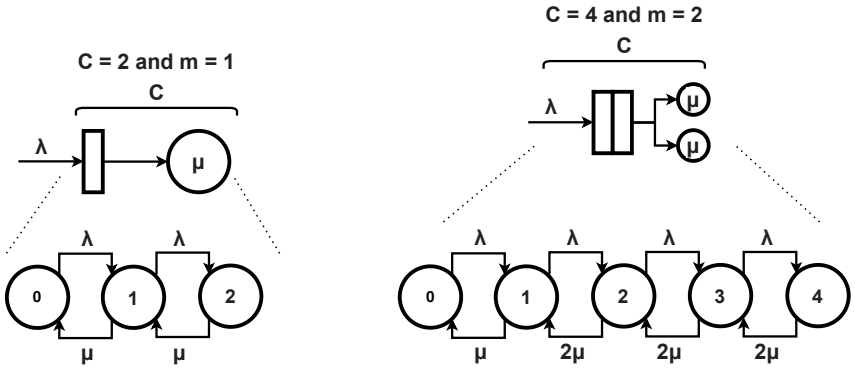
The third level of the hierarchy captures the performance of the applications within the infrastructure. Here, m represents the number of threads, and C represents the system's total capacity to handle incoming service requests, taking into account both processing and queuing capacity. Hence, C is the sum of the number of threads (m) and the buffer size

for the applications. As the number of physical nodes and application instances increases, the overall capacity of the system also expands.

Figure 14 illustrates two examples of application behavior. In the first example, the application has a total capacity of two requests, with one request being processed in a thread while another request waits in the buffer (Figure 15(a)). In the second example, the capacity doubles, representing two applications running concurrently (Figure 15(b)).

A queuing process can be characterized by the arrival of requests for service, their potential wait in a queue if necessary, and their eventual departure from the system upon being served (CONSTANTIN, 2011). Analyzing this queuing process allows us to construct a continuous-time Markov chain, where the parameter λ represents the arrival rate of requests, and μ denotes the rate at which requests are processed. It is important to note that the buffer in the system is finite, implying that if the incoming workload exceeds the processing rate, any excess requests will be discarded.

Figure 14 – Queueing theory represented by Markov chains.



(a) Capacity of 2 requests.

(b) Capacity of 4 requests.

Source: Elaborated by the author.

The Markov chain model depicted in Figure 13 facilitates the calculation of various performance metrics, taking into account the failure and repair times of the infrastructure. These metrics encompass utilization, response time, waiting time, and discard rate. To estimate the availability of the system, we sum the probabilities of states in which both the physical nodes and applications are operational. Moreover, to calculate performance-related metrics, we employ the probabilities derived from the third level of the model. By obtaining closed-form solutions from the three-level Markov chain, we establish analytical performability models represented by Equations 5.8, 5.9, 5.10, and 5.11.

The effective arrival rate, denoting the rate at which jobs enter the infrastructure, is expressed as $\lambda(1 - \pi_{i \times j \times C})$. In this equation, i and j respectively represent the number of operational physical machines and applications at a given moment. Additionally, C signifies the maximum number of jobs that can be accommodated by the application,

accounting for the overall capacity considering the buffer size and the number of threads allocated for request processing. Thus, $\pi_{i\times j\times C}$ represents the probability of the state in which there are i operational physical machines, j running applications, and the maximum number of requests within the infrastructure.

Performability Utilization:

$$PU = \sum_{i=1}^{M} \left(\frac{M!}{i!\ (M-i)!} \ \times \ \frac{\lambda_{pn}^{M-i}\ \mu_{pn}^{i}}{(\lambda_{pn}\ +\ \mu_{pn})^{M}} \right) \ \times \ \left((\sum_{j=1}^{iL} \frac{(iL)!}{j!\ (iL-j)!} \times \frac{\lambda_{app}^{iL-j}\ \mu_{app}^{j}}{(\lambda_{app}\ +\ \mu_{app})^{iL}} \ \times \ \frac{\lambda(1-\pi_{i\times j\times C})}{i\times j\times m\times \mu} \right). \tag{5.8}$$

Performability Response Time:

$$PRT = \sum_{i=1}^{M} \left(\frac{M!}{i!\ (M-i)!} \ \times \ \frac{\lambda_{pn}^{M-i}\ \mu_{pn}^{i}}{(\lambda_{pn}\ +\ \mu_{pn})^{M}} \right) \ \times \ \left(\sum_{j=1}^{iL} \frac{(iL)!}{j!\ (iL-j)!} \times \frac{\lambda_{app}^{iL-j}\ \mu_{app}^{j}}{(\lambda_{app}\ +\ \mu_{app})^{iL}} \ \times \ \frac{\sum_{n=1}^{ijC} n\pi_n}{\lambda(1-\pi_{i\times j\times C})} \right). \tag{5.9}$$

Performability Waiting Time:

$$PWT = \sum_{i=1}^{M} \left(\frac{M!}{i!\ (M-i)!} \ \times \ \frac{\lambda_{pn}^{M-i}\ \mu_{pn}^{i}}{(\lambda_{pn}\ +\ \mu_{pn})^{M}} \right) \ \times \ \left(\sum_{j=1}^{iL} \frac{(iL)!}{j!\ (iL-j)!} \times \frac{\lambda_{app}^{iL-j}\ \mu_{app}^{j}}{(\lambda_{app}\ +\ \mu_{app})^{iL}} \ \times \ \frac{\sum_{n=ijm}^{ijC} (n-ijm)\pi_n}{\lambda(1-\pi_{i\times j\times C})} \right). \tag{5.10}$$

Performability Discard Rate:

$$PDR = \left(\sum_{i=1}^{M} \left(\frac{M!}{i!\ (M-i)!} \ \times \ \frac{\lambda_{pn}^{M-i}\ \mu_{pn}^{i}}{(\lambda_{pn}\ +\ \mu_{pn})^{M}} \right) \ \times \ \left(\left(\sum_{j=1}^{iL} \frac{(iL)!}{j!\ (iL-j)!} \right. \right. \right.$$
$$\left. \left. \left. \times \ \frac{\lambda_{app}^{iL-j}\ \mu_{app}^{j}}{(\lambda_{app}\ +\ \mu_{app})^{iL}} \ \times \ \lambda\pi_{i\times j\times C} \right) + \ \frac{\lambda_{app}^{iL}}{(\lambda_{app}\ +\ \mu_{app})^{iL}} \right) \right) + \frac{\lambda_{pn}^{M}}{(\lambda_{pn}\ +\ \mu_{pn})^{M}}. \tag{5.11}$$

Equation 5.8 provides a representation of the infrastructure's utilization, accounting for the total number of applications that the system is capable of delivering (i.e., $M \times L$). This equation calculates the utilization by considering both the occurrence of failures and the subsequent repair routines. Equation 5.9 quantifies the response time of the infrastructure, which is a crucial performance metric in the context of edge and fog computing environments. Similarly, Equation 5.10 computes the waiting time experienced by a request within the system, an important performance metric directly linked to the user's experience. Furthermore, Equation 5.11 represents the number of lost requests due to system unavailability or reaching its maximum capacity. In other words, any request received when the system is either down or operating at full capacity will be lost.

Our decision to employ a continuous-time Markov chain for these models, as opposed to other modeling techniques such as Reliability Diagram Models (RBD) or Stochastic Petri Nets (SPN), stems from the advantages it offers in developing analytical models to evaluate system availability (CALIRI, 2000). RBD and SPN models rely on numerical approximations, which can limit scalability compared to analytical models. By utilizing analytical models, we gain the flexibility to explore multiple scenarios with different input parameters easily. Additionally, our proposed models possess the ability to represent various systems by adjusting the relevant parameters accordingly.

5.3 HIERARCHICAL AVAILABILITY MODEL FOR EDGE-FOG-CLOUD CONTINUUM

The utilization of hierarchical models allows us to capture a more precise representation of the environment and facilitate the design of the system. Furthermore, these models enable us to predict the behavior of the service infrastructure. Within this section, we introduce a hierarchical model that specifically focuses on representing the availability metric in an edge-fog-cloud environment. Initially, we present a Markov chain as a means of generalizing the components within the infrastructure. Subsequently, we leverage this Markov chain to derive closed-form solutions that effectively calculate the availability metrics of the individual components.

Figure 9 illustrates the configuration of components and the abstraction employed to derive closed-form solutions for the nodes. Our node modeling utilizes a two-level continuous-time Markov chain, where the first and second levels respectively represent the

physical nodes and applications. Consequently, the number of applications in operation is contingent upon the availability of functional physical nodes. Conversely, the remaining components are modeled using a one-level continuous-time Markov chain.

Within the first-level continuous-time Markov chain (CTMC), a system is deemed unavailable if it resides in the state 0, indicating the absence of operational physical machines. Similarly, in any second-level CTMC, a node is regarded as unavailable if it exists in the state 0, indicating the absence of running applications, regardless of the number of functional physical machines. To exemplify, suppose we aim to model a system comprising two identical physical machines capable of accommodating two applications each, along with two independent repair teams. In this case, the corresponding CTMC is depicted in Figure 11.

The Markov chain model depicted in Figure 11 can be generalized by the model presented in Figure 12. Within this Markov chain, M represents the total number of physical nodes, while L represents the number of applications currently running. The parameters λ_{pn} and μ_{pn} respectively denote the failure rate and repair rate of the physical node, while λ_{app} and μ_{app} signify the failure rate and repair rate of the application. It is important to emphasize that we assume a homogeneous environment, where each of the M machines is identical, and there are sufficient independent repair teams to cater to the system's needs.

Availability:

$$A = \sum_{i=1}^{M} \left(1 - \frac{\lambda_{app}^{iL}}{(\lambda_{app} \ + \ \mu_{app})^{iL}}\right) \ \times \ \left(\frac{M!}{i! \ (M-i)!} \ \times \ \frac{\lambda_{pn}^{M-i} \ \mu_{pn}^{i}}{(\lambda_{pn} \ + \ \mu_{pn})^{M}}\right). \tag{5.12}$$

Availability K out of N:

$$\begin{aligned} A_{KooN} = \sum_{i=1}^{M} \left(\sum_{j=K}^{iL} \frac{(iL)!}{j! \ (iL-j)!} \ \times \ \frac{\lambda_{app}^{iL-j} \ \mu_{app}^{j}}{(\lambda_{app} \ + \ \mu_{app})^{iL}} \right) \\ \times \left(\frac{M!}{i! \ (M-i)!} \ \times \ \frac{\lambda_{pn}^{M-i} \ \mu_{pn}^{i}}{(\lambda_{pn} \ + \ \mu_{pn})^{M}}\right). \end{aligned} \tag{5.13}$$

By utilizing the Markov chain model illustrated in Figure 12, we can derive several performance metrics, including the availability (A) and (A_{KooN}), unavailability (UA), uptime (UP), downtime (DT), average number of applications available ($ANAPPA$), and

capacity-oriented availability (COA). The system's availability is calculated by summing the probabilities of states where both the physical nodes and applications are operational. Expressing it in terms of the failure rate (λ) and repair rate (μ), we obtain the analytical availability model as represented by Equation 5.12.

$$UA = 1 - A \tag{5.14}$$

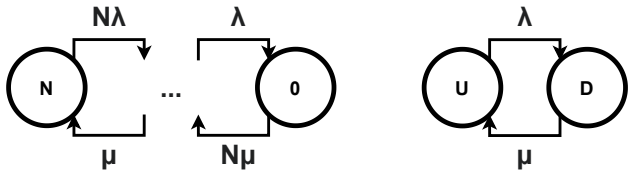
$$UP = A \times T \tag{5.15}$$

$$DT = UA \times T \tag{5.16}$$

Equation 5.12 provides an expression for the availability of at least one out of N applications functioning within the system, where N represents the total number of applications that can be delivered by the system (i.e., $M \times L$). This equation considers the system to be available if at least one application is operational, thus representing a specific case of Equation 5.13. Conversely, Equation 5.13 represents the availability of K out of N applications, where K denotes the minimum number of functioning applications required for the system to be considered available. In other words, the system is deemed available only if a minimum of K applications are operational.

Additionally, by utilizing these availability models, we can easily determine the unavailability, uptime, and downtime of the system, as expressed by Equations 5.14, 5.15, and 5.16, respectively. It is important to note that the servers are modeled using a two-level Continuous-Time Markov Chain (CTMC), while other components such as gateways, storage, and devices are represented by a one-level CTMC. In the case of these components, the software element is not considered, allowing us to abstract them using a one-level CTMC.

Figure 15 – Other components' CTMC.



(a) Device Model. (b) Storage and Gateway Model.

Source: Elaborated by the author.

Figure 15 illustrates the additional models employed to represent the gateway, storage, and devices. These models are essential components in the system hierarchy. The closed-form solutions of these models are presented in Equations 5.17, 5.18, and 5.19, which

correspond to the mathematical representations depicted in Figure 15. These closed-form equations were utilized to calculate the input values required for higher-level models in the hierarchy.

Availability of Figure 16(a):

$$A = 1 - \frac{\lambda^N}{(\lambda \ + \ \mu)^N}. \tag{5.17}$$

Availability of Figure 16(b):

$$A = \frac{\lambda}{\lambda \ + \ \mu}. \tag{5.18}$$

Availability K out of N of Figure 16(a):

$$A_{KooN} = \sum_{i=K}^{N} \frac{N!}{i! \ (N-i)!} \ \times \ \frac{\lambda^{N-i} \ \mu^i}{(\lambda \ + \ \mu)^N}. \tag{5.19}$$

Through the implementation of a hierarchical model, we can effectively represent environments by utilizing multiple sub-models at distinct levels. This approach allows for the modeling of heterogeneous attributes inherent in the overall system architecture. Additionally, it enables the capturing of detailed operational behaviors exhibited by subsystems and components at lower levels. The rationale behind employing hierarchical modeling lies in the amalgamation of the simplicity offered by combinatorial models with the capability of state-based models to encapsulate intricate operational behaviors via states and transitions.

The fault tree modeling formalism offers a means to represent the state of a system using Boolean functions that yield a value of True when the system experiences a failure. This failure-oriented approach differs from Reliability Block Diagrams, where the system state is described by a structure-function that characterizes system failures. In scenarios where the system exhibits multiple unwanted states, it is necessary to define Boolean functions or structure functions to represent each failure mode. Our model assumes the independence of component failures to facilitate a straightforward solution.

Figure 16 illustrates our proposed model for representing the edge-fog-cloud continuum, utilizing both continuous-time Markov chains and fault tree modeling techniques.

Within the depicted model, we consider the end-to-end application as a composition of multiple components, denoted as $C = \{c_i | 1 \leq i \leq n\}$. In this context, we define a random variable $y_i(t)$ to indicate the state of component i, where:

$$y_i = \begin{cases} 1, \text{ if the component } i \text{ is faulty at time } t \\ 0, \text{ if the component } i \text{ is operational at time } t. \end{cases} \tag{5.20}$$

In our hierarchical model, we have seven components, so our FT structure function is expressed as:

$$\begin{aligned} \Phi(y_i) = [1 - (1 - y_0) \times (1 - y_1) \times (1 - y_2) \\ \times (1 - y_3) \times (1 - y_4) \\ \times (1 - y_5) \times (1 - y_6)] \end{aligned} \tag{5.21}$$

From the structure function, we are able to derive the availability of the system, thus:

$$E(y_i) = A_i \tag{5.22}$$

$$\begin{aligned} A_s = E(\Phi(y_i)) = E[1 - (1 - y_0) \times (1 - y_1) \times (1 - y_2) \\ \times (1 - y_3) \times (1 - y_4) \\ \times (1 - y_5) \times (1 - y_6)] \end{aligned}$$

then

$$\begin{aligned} A_s = E[1] - E[(1 - y_0)] \times E[(1 - y_1)] \times E[(1 - y_2)] \\ \times E[(1 - y_3)] \times E[(1 - y_4)] \\ \times E[(1 - y_5)] \times E[(1 - y_6)] \end{aligned}$$

as y_i are independent, then

$$\begin{aligned} A_s = 1 - (1 - E[y_0]) \times (1 - E[y_1]) \times (1 - E[y_2]) \\ \times (1 - E[y_3]) \times (1 - E[y_4]) \\ \times (1 - E[y_5]) \times (1 - E[y_6]) \end{aligned}$$

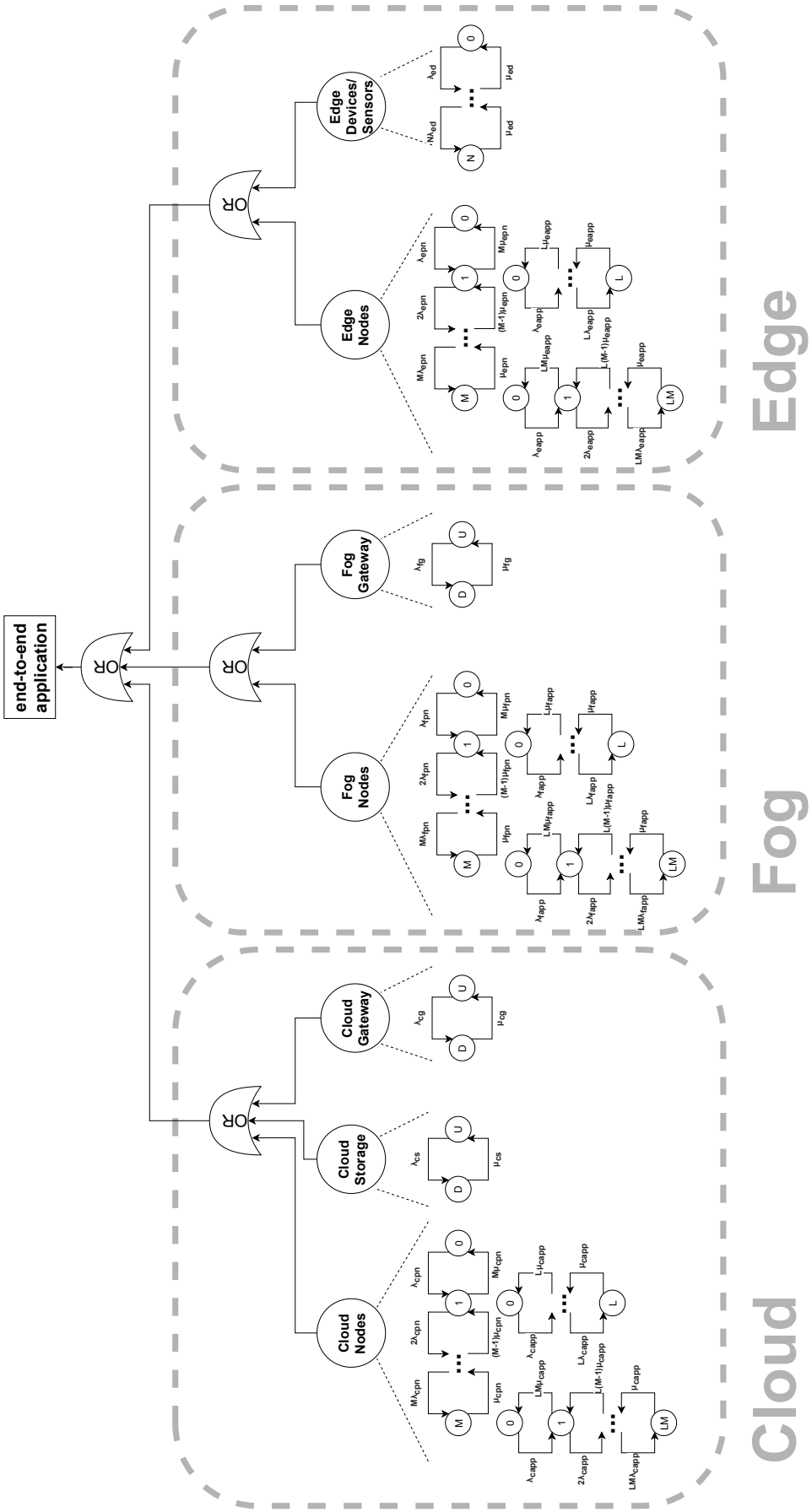
finally

$$\begin{aligned} A_s = 1 - (1 - A_0) \times (1 - A_1) \times (1 - A_2) \\ \times (1 - A_3) \times (1 - A_4) \\ \times (1 - A_5) \times (1 - A_6) \end{aligned} \tag{5.22}$$

Note that in our model, the operational status of all components is essential for the application to function correctly. Any failure or unavailability of components would render the application faulty.

Furthermore, our model assumes that the cloud environment comprises cloud nodes, a cloud storage, and a cloud gateway. In contrast, the fog environment consists of fog nodes and a fog gateway. Lastly, the edge environment is composed of edge nodes, as well as edge devices and sensors.

Figure 16 – Hierarchical model.



Source: Elaborated by the author.

5.4 HIERARCHICAL AVAILABILITY MODELS WITH FAULT COVERAGE PROBABILITY

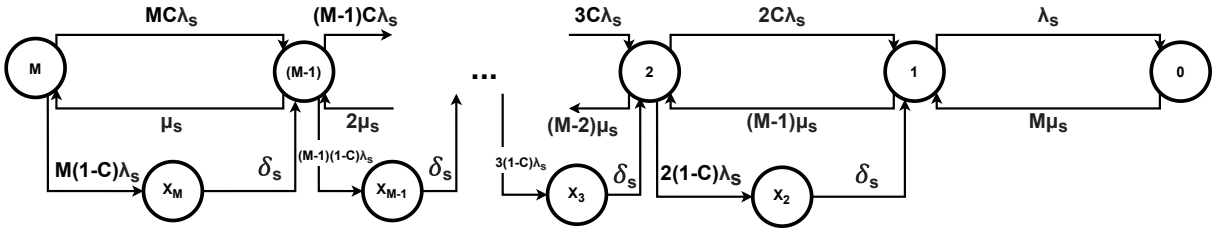
In this section, we present our hierarchical availability model incorporating fault coverage probability. The model consists of two levels of hierarchy: the first level comprises a Markov chain, and the second level utilizes a Reliability Block Diagram (RBD) model. The availability values obtained from the closed-form equations at the first level serve as input for the RBD model, enabling the calculation of the overall service availability.

Similar to our previous models, certain abstractions are necessary to derive closed-form equations, as depicted in Figure 9. We employ two types of continuous-time Markov chains (CTMCs) to represent the nodes: a two-level hierarchy model and a model with fault coverage probabilities. In the two-level model, the first level represents the physical nodes, while the second level represents the applications. The sensor is modeled using a one-level CTMC with fault coverage probabilities.

In the two-level hierarchical CTMC, a state of 0 at the first level indicates the absence of operational physical machines, resulting in unavailability of the service. Additionally, even if multiple physical nodes are functional, a state of 0 at any second level signifies the absence of running applications, rendering the system inoperative. Figure 12 illustrates the generalization for the edge and fog nodes. In our proposed model, M represents the total number of physical nodes, and L represents the total number of supported applications in our infrastructure. The parameters λ_{pn} and μ_{pn} correspond to the failure and repair rates of the physical node, while λ_{app} and μ_{app} denote the failure and repair rates of the application. Moreover, we assume the existence of M autonomous repair teams in this scenario.

On the contrary, Figure 17 illustrates the representation of multiple sensors, taking into account the fault coverage probabilities. In this Markov chain model, M denotes the number of sensors. The sensor's fault is covered with a probability of C and not covered with a probability of $1 - C$. The parameter δ_s represents the time required to detect the failure in the sensor. Additionally, λ_s and μ_s correspond to the failure and repair rates of the sensor, respectively.

Figure 17 – Markov chain model of sensors.



Source: Elaborated by the author.

Utilizing the models depicted in Figure 12 and Figure 17, we estimate various met-

rics, including availability (A), unavailability (UA), uptime (UP), downtime (DT), average number of applications available ($ANAPPA$), average number of sensors available ($ANSA$), and capacity-oriented availability (COA). By employing the failure and repair rates represented by λs and μs, we derive Equation 5.24 for the availability of edge and fog nodes and Equation 5.25 for the availability of sensors.

Availability of Edge and Fog Nodes:

$$A_{nodes} = \sum_{i=1}^{M} \left(1 - \frac{\lambda_{app}^{iL}}{(\lambda_{app} \ + \ \mu_{app})^{iL}}\right) \ \times \ \frac{M!}{i! \ (M-i)!} \ \times \ \frac{\lambda_{pn}^{M-i} \ \mu_{pn}^{i}}{(\lambda_{pn} \ + \ \mu_{pn})^{M}}. \tag{5.24}$$

Availability of Sensors:

$$A_{sensors} = \frac{M(C-1)\lambda_s\mu_s^M}{P} + \sum_{i=1}^{M-2}\left(\frac{M!}{i!(M-1-i)!}\right)\frac{C-1)\lambda_s^{M-i}\mu_s^{i+1}}{Q} - \sum_{i=0}^{M}\left(\frac{M!}{i!(M-i)!}\right)\frac{\delta_s\lambda_s^{M-i}\mu_s^{i}}{Q},$$

where,

$$P = \delta_s(\lambda_s+\mu_s)^M - M(C-1)\lambda_s\mu_s^2\left(\sum_{i=1}^{M-1}\frac{(M-1)!}{i!(M-1-i)!}\lambda_s^{M-1-i}\mu_s^{i-1}\right),$$
$$Q = -\delta_s(\lambda_s+\mu_s)^M + M(C-1)\lambda_s\mu_s^2\left(\sum_{i=1}^{M-1}\frac{(M-1)!}{i!(M-1-i)!}\lambda_s^{M-1-i}\mu_s^{i-1}\right). \tag{5.25}$$

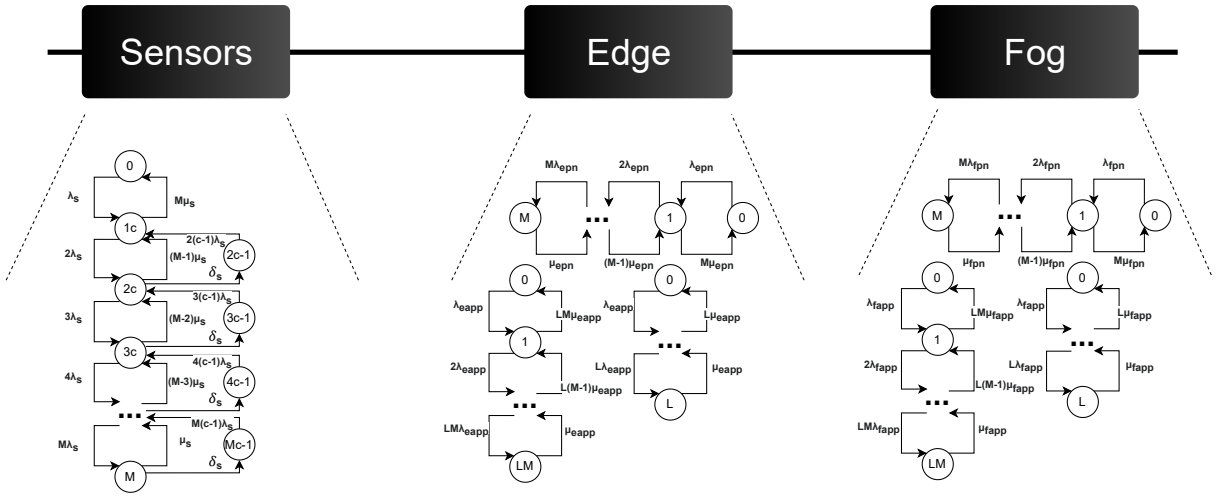
Average number of sensors available:

$$ANSA = \sum_{i=1}^{M} \frac{M! \ \mu_s^i \ \lambda_s^{M-i} \ (\lambda_s+\mu_s)^{-M}}{i! \ (M-i)!}. \tag{5.26}$$

$$COA_{sensors} = \frac{ANSA}{M} \tag{5.27}$$

The primary objective of employing hierarchical modeling in our study is to capitalize on the advantages offered by combinatorial models, known for their simplicity and ease of analysis. The Reliability Block Diagram formalism is utilized to represent the system state as a Boolean function, yielding a value of True when the system is deemed available. To facilitate a more streamlined solution, we assume that the failures of individual components are independent within our model.

Figure 18 – Hierarchical model with fault coverage probability.



Source: Elaborated by the author.

Figure 18 illustrates our proposed model for representing a smart building monitoring service using continuous-time Markov chain and Reliability Block Diagram techniques. It is assumed that all components must be operational for the service to be considered functional, and any deviation from this state indicates a faulty condition.

The utilization of Markov chain modeling in this study is driven by its ability to derive closed-form equations for calculating availability (CALIRI, 2000). In contrast, numerical models lack the same level of scalability as analytical models, making them less suitable for our purposes.

In the depicted model shown in Figure 18, our end-to-end application consists of a collection of components denoted as $C = \{c_i | 1 \leq i \leq n\}$. Consequently, we make the assumption that a random variable $y_i(t)$ represents the state of component i, yielding:

$$y_i = \begin{cases} 1, \text{ if the component } i \text{ is operational at time } t \\ 0, \text{ if the component } i \text{ is faulty at time } t. \end{cases} \tag{5.28}$$

In our hierarchical model, we have three components arranged in a pure series structure, resulting in a Reliability Block Diagram (RBD) structure function expressed as:

$$\Phi(y_i) = y_0 \times y_1 \times y_2 \tag{5.29}$$

From the structure function, we are able to derive the availability of the system, thus:

$$E(y_i) = A_i \tag{5.30}$$

$$A_s = E(\Phi(y_i)) = E[y_0 \times y_1 \times y_2]$$

as y_i are independent, then

$$A_s = E[y_0] \times E[y_1] \times E[y_2]$$

finally

$$A_s = A_0 \times A_1 \times A_2 \tag{5.30}$$

Therefore, we can calculate the steady-state availability, which provides an estimate of the system's availability over a long period of time.

5.5 FINAL REMARKS

This chapter has presented a comprehensive exploration of availability and performance proposed models for systems in the edge-fog-cloud continuum. By employing analytical techniques, such as continuous-time Markov chains and Reliability Block Diagrams, we have been able to derive closed-form equations that provide valuable insights into the behavior of these systems.

The selection of continuous-time Markov chains (CTMCs) as the preferred modeling approach for edge-fog-cloud applications was driven by several factors. CTMCs offer a robust mathematical framework that facilitates the analysis and prediction of the intricate behaviors exhibited by complex systems characterized by stochastic dynamics and state transitions. Moreover, the continuous and time-dependent nature of these systems further reinforces the suitability of CTMCs as a modeling formalism. The inherent versatility and analytical capabilities of CTMCs make them an ideal choice for accurately representing and studying the complex interplay of components within edge-fog-cloud applications.

Overall, the availability and performance proposed models presented in this chapter serve as valuable tools for researchers, system designers, and practitioners in the field of edge-fog-cloud environments. By understanding the behavior and limitations of these systems, we can enhance their design, operation, and maintenance, ultimately leading to more reliable, efficient, and resilient infrastructures.

6 CASE STUDIES

The development of Internet of Things (IoT) applications entails the integration of complex computational environments that encompass cloud, fog, and edge computing paradigms (ARAUJO et al., 2019). In this chapter, we conducted a series of case studies, which we categorized into four distinct sections: Availability Models, Performability Models, Hierarchical Models, and Hierarchical Models With Coverage Probability. Each section serves to validate our models by demonstrating their ability to accurately represent the baseline edge, fog, and cloud computing environments. The case studies primarily focused on performing steady-state availability (SSA) analyses of the underlying infrastructures. Within these analyses, key measures of interest included downtime hours, mean time to failure, mean time to repair, and availability metrics expressed in terms of nines. Furthermore, additional case studies were conducted to evaluate capacity-oriented availability, allowing for an assessment of the impact of resource scalability on service delivery. Cost evaluations were also conducted, enabling a comparison of the actual expenses associated with each environment.

6.1 AVAILABILITY MODELS

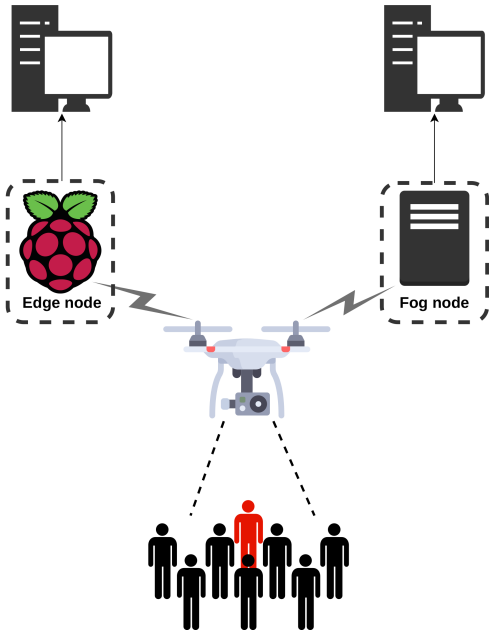
In this section, we have undertaken the development of a face recognition system as a means to validate our models. Face recognition systems hold significant relevance in various contexts, such as mobile banking applications, where they are employed for transaction authentication purposes. Additionally, face recognition plays a crucial role in security and safety systems deployed at locations like airports, stadiums, and other service establishments. Recognizing the criticality of these systems in ensuring safety and security, we have conducted thorough evaluations of availability models specifically tailored to address face recognition services.

6.1.1 Case Study I - Availability model validation

In this case study, we present the experiments conducted to assess the validity of our availability models in a real-world scenario. The experimental environment comprised a set of components, including a drone, three machines (consisting of a fog node and two clients), and a Raspberry Pi device serving as an edge node. The drone employed for the experiment was the Dji Mavic Pro Platinum model, tasked with capturing video footage and transmitting it to the fog and edge infrastructure. The captured video footage was in MP4 format, with a frame rate of 30 frames per second and a resolution of 3840×2160. The fog node utilized the XenServer 7.2.0 hypervisor platform, featuring 8 GB of RAM, 1 TB of storage, and an Intel i5 CPU running at 3.20 GHz quad-core with eight threads.

The two client machines, used for testing purposes, were equipped with 8 GB of RAM, 1 TB of storage, and an Intel i7-4510U CPU running at 2.00 GHz. The Raspberry Pi, functioning as the edge node, possessed a 1 GHz single-core processor with two threads and 512 MB of RAM. Both the fog node and edge node executed a Python script responsible for face recognition tasks, in addition to an RTMP server (such as Nginx[1]) employed for receiving the video stream transmitted by the drone. Figure 9 and Figure 19 provide visual representations of the baseline environment concept.

Figure 19 – Testbed environment.



Source: Elaborated by the author.

The VGGFace2 model, developed by researchers at the Visual Geometry Group at Oxford (CAO et al., 2018), was employed for face recognition in our experiments. Utilizing pre-trained models was facilitated through the adoption of the Keras framework. Following the establishment of the experimental environment, we conducted multiple case studies by introducing faults using Python scripts.

Our baseline infrastructure consisted of two physical nodes, namely the edge and fog nodes, with each node running a single application. The failure and repair rates utilized in our models were obtained from previous studies (LISBOA et al., 2018; DANTAS et al., 2016; PEREIRA et al., 2021). It was assumed that the time to failure and repair followed an exponential distribution, with the failure and repair rates representing the reciprocals of the Mean Time to Failure (MTTF) and Mean Time to Repair (MTTR), respectively. The values of MTTF and MTTR in hours, as adopted from the literature, are presented in Table 5.

[1] https://www.nginx.com/

When assessing system availability, the fault injection technique is commonly employed due to the unpredictable nature or lengthy occurrence of faults (SOUZA et al., 2013; BRILHANTE et al., 2014; JAMMAL et al., 2018; PEREIRA et al., 2021). The fault injection mechanism allows for the acceleration, control, and monitoring of experiments during fault events (AGARWAL et al., 2020). Upon the occurrence of a fault, it induces modifications within the system, leading to an erroneous state. Additionally, faults may propagate throughout the system, resulting in disruptions to the availability of the provided services.

Table 5 – MTTF and MTTR from literature.

Node	Components	MTTF (h)	MTTR (h)
Edge	Raspberry	4767.8	3.48
	OS	2880	1
	Python App	217.8	0.46
	Nginx App	217.8	0.46
Fog	Hardware	8760	1.67
	OS	2880	1
	Cont. Management	2880	1
	Python App	217.8	0.46
	Nginx App	217.8	0.46

Source: Elaborated by the author based on Dantas et al. (2016).

In our baseline environment, we aimed to minimize the number of components required to provide the service. Hence, we employed only the essential components for each node. The minimal components for the edge node encompassed the Raspberry Pi, an operating system, a Python application for face recognition, and an Nginx RTMP server responsible for receiving the video stream from the drone. The fog computing node shared the same minimal components as the edge node, differing only in the hardware utilized, where a personal computer was employed.

To validate our models, it was necessary to induce stress on the system to accelerate the Mean Time to Failure (MTTF) of the components. For this purpose, we developed a fault injector capable of introducing faults into the nodes. Following the assumption that the time to failure and repair of each component followed an exponential distribution, the failure and repair rates were obtained as the inverses of the MTTF and Mean Time to Repair (MTTR) values, respectively, obtained from the literature (LISBOA et al., 2018; DANTAS et al., 2016; PEREIRA et al., 2021). By accelerating the MTTF, we increased the speed by a factor of 876, effectively compressing one year into 10 hours. The specific values set in the fault injector are presented in Table 6 in hours. As a result of this accelerated MTTF, the system's availability decreased accordingly.

By employing the abstraction approach utilized in our Markov chain model, we are able to identify two distinct components within the edge and fog computing environment,

Table 6 – MTTF and MTTR in the fault injector.

Node	Components	MTTF (h)	MTTR (h)
Edge	Raspberry	5.4	3.48
	OS	3.28	1
	Python App	0.24	0.46
	Nginx App	0.24	0.46
Fog	Hardware	10	1.67
	OS	3.28	1
	Cont. Management	3.28	1
	Python App	0.24	0.46
	Nginx App	0.24	0.46

Source: Elaborated by the author.

namely the application and the physical node. The abstraction process was conducted using the Reliability Block Diagram (RBD) model, leveraging the Mercury tool (SILVA et al., 2015). This approach effectively consolidates multiple components into a single representative entity, simplifying the system representation and enabling the application of analytical techniques based on Continuous-Time Markov Chain (CTMC) analysis. To this end, the values provided in Table 6 were employed to derive the corresponding values for the abstraction components, which can then be utilized as inputs in the model. These mapped values for the abstraction components are presented in Table 7.

By combining the Mean Time to Failure (MTTF) and Mean Time to Repair (MTTR) values of serial components, we create an aggregated equivalent component that encapsulates the unified representation of availability characteristics. This condensation procedure significantly simplifies the modeling of individual components, enabling a more streamlined analysis of the entire system. By considering the combined availability behavior of the condensed component, a comprehensive understanding of the system's overall availability can be achieved.

This abstraction technique proves particularly advantageous when employing CTMC analysis, as it allows for the system to be represented as a set of interconnected states and transitions. The consolidation of multiple components into a single entity reduces the number of states within the CTMC model, thereby facilitating a more manageable analysis process. As a result, analytical solutions, such as steady-state probabilities and performance measures, can be derived, providing valuable insights into the system's availability.

Figure 20 illustrates the deployment of our fault-injection environment. To generate the workload, we utilized a drone for capturing images and transmitting them to our nodes, as described previously. The fault injector was executed on a personal computer equipped with 8 GB of RAM, 1 TB of storage, and an Intel i7-4510U CPU running at 2.00 GHz.

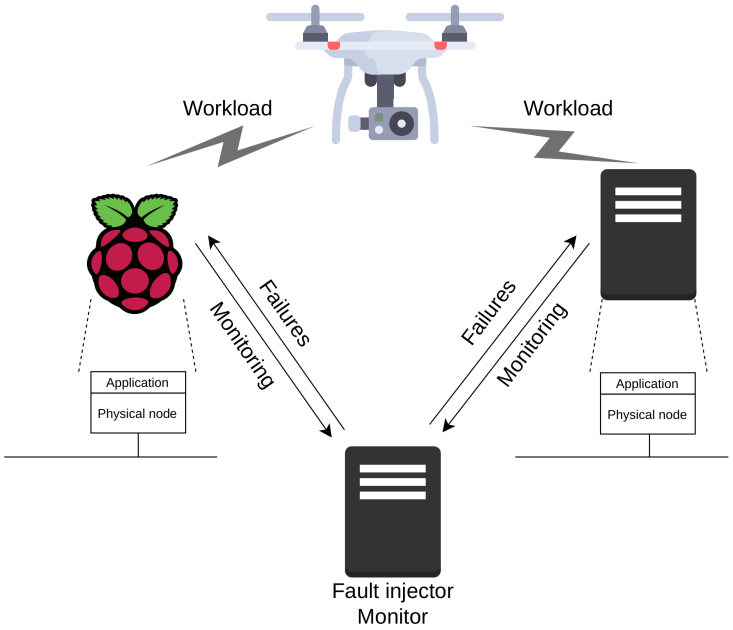
Table 7 – MTTF and MTTR for model.

Node	Components	MTTF (h)	MTTR (h)
Edge	Physical Node	2.04	2.34
	Application	0.34	0.44
Fog	Physical Node	2.48	1.29
	Application	0.34	0.44

Source: Elaborated by the author.

Implemented in Python, the fault injector enumerates all components to be monitored and subsequently fail. For each component, the fault injector spawns a dedicated thread responsible for injecting faults. The time until failure for each component is assumed to follow an exponential distribution based on the values specified in Table 6. Consequently, the thread generates a random number adhering to this distribution and waits for the specified duration before injecting a fault. Additionally, a separate thread continuously monitors the availability of the system. The monitoring results, denoted as either U for system uptime or D for system downtime, are stored in a text file. The interval between samples is set to every 10 seconds.

Figure 20 – Fault-injection environment.



Source: Elaborated by the author.

Utilizing the environment depicted in Figure 20, we conducted two experiments, one for the fog node and another for the edge node, each spanning a duration of 96 hours. These experiments generated two text files, one for each node, containing a series of occurrences denoted by U (uptime) and D (downtime). By analyzing the number of Us

and Ds, we computed the corresponding time to failure (TTF) and time to repair (TTR). For example, if we observed 50 consecutive Us followed by a D, the duration of uptime in that interval would be 50×10 seconds (the sampling interval), resulting in a TTF of 500 seconds. We employed this approach to calculate the TTFs and TTRs, resulting in 73 sample points for the edge node and 61 sample points for the fog node.

In accordance with the validation procedure outlined in Chapter 4, we performed a data analysis and determined that no theoretical distribution adequately represented our sample distribution. Consequently, we applied the Bootstrap technique to estimate the confidence intervals for the availability of the fog and edge nodes. The Bootstrap technique involves resampling a dataset with replacement to generate statistics that pertain to the population (DEVORE, 2008).

Subsequently, we employed a statistical software package to generate 1000 bootstrap samples for the availability of the edge and fog nodes. Each sample represented the mean availability for a specific node, allowing us to calculate the confidence interval. The 25th smallest and 25th largest values among these 1000 bootstrap samples served as the lower and upper bounds, respectively, of the 95% confidence interval. It is important to note that the correct interpretation of a 95% confidence interval is based on repeated experiments. Over the long run, if the experiment were to be repeated multiple times, the availability of the edge and fog nodes would be expected to fall within these calculated confidence intervals approximately 95% of the time.

By employing the analytical models described in Section 5.1 and utilizing the component values provided in Table 7, we obtained the availability values for the edge and fog nodes. The point estimations and their corresponding confidence intervals are summarized in Table 8, along with the results obtained from the models.

Table 8 – Validation result.

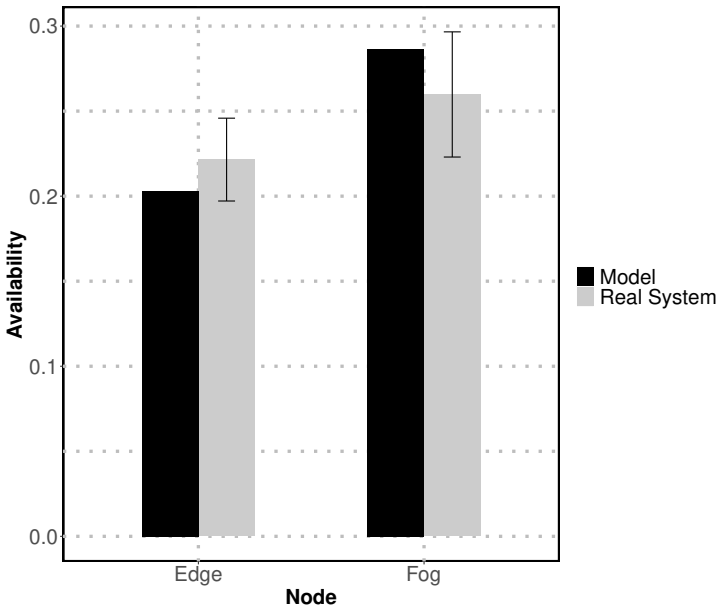
Node	Model Result	System Measurement	CI 95% of the system
Edge	0.2030	0.2215	$0.1972 < \theta < 0.2459$
Fog	0.2867	0.2598	$0.2230 < \theta < 0.2966$

Source: Elaborated by the author.

Upon analysis of the confidence intervals, it is evident that the models' estimations for the nodes fall within these intervals. Thus, we cannot refute the notion that our models accurately represent the actual environment. As a result, we can assert that our models have been validated. The graphical representation of the result values from Table 8 is depicted in Figure 21.

Henceforth, we can employ the models to compute the availability of the edge and fog nodes. Furthermore, we can develop strategies to enhance availability levels in various scenarios and conduct a comprehensive cost analysis.

Figure 21 – Fault injection environment.



Source: Elaborated by the author.

6.1.2 Case Study II - Availability evaluation

In this case study, we employ the proposed analytical model to estimate the availability of our baseline environment. This case study exemplifies the utility of our models in investigating availability concerns within a system. Subsequent case studies involve the analysis of hypothetical scenarios that surpass the scale of our baseline infrastructure.

Analytical models distinguish themselves from other modeling techniques, such as Reliability Block Diagrams and Stochastic Petri Nets, due to their scalability. Through the use of analytical models, we can assess multiple scenarios with reduced computational costs (JAIN, 1990). It is important to note that, for the ensuing evaluations, we utilize the literature values of Mean Time to Failure (MTTF) and Mean Time to Repair (MTTR) for the edge and fog nodes, as depicted in Table 5. These values are employed in the Mercury tool (SILVA et al., 2015; PINHEIRO et al., 2021) to calculate the corresponding abstraction values as shown in Table 9.

Table 9 – The abstraction MTTF and MTTR.

Node	Components	MTTF (h)	MTTR (h)
Edge	Physical Node	1795.45	1.93
	Application	108.89	0.46
Fog	Physical Node	2167.42	1.16
	Application	108.89	0.46

Source: Elaborated by the author.

Assuming that our single-edge physical node has the capability to run two applications, while the single fog physical node can accommodate eight applications, we set the values of L to two for the edge environment and eight for the fog environment in our model. In this initial evaluation, redundancy is not considered, resulting in M being equal to one for both the edge and fog environments.

Through the evaluation process, we obtained a downtime of 9.56 hours for the edge node, indicating that within a year, the system would experience approximately 10 hours of unavailability. Consequently, the uptime would amount to 8750.44 hours. By adopting the edge solution, the calculated number of nines is 2.9622. Conversely, for the fog node, we obtained a downtime of 4.84 hours, corresponding to approximately 5 hours of unavailability, which is half of the edge node. The resulting uptime for the fog node is 8755.16 hours. The fog node achieves a number of nines of 3.2576, indicating effective management. The summary of the achieved results for both nodes can be found in Table 10.

Table 10 – Results of the baseline environment.

Node	**Metric**	**Values**
Edge	Availability	0.9989
	Unavailability	0.0011
	# of 9's	2.9622
	Downtime (h)	9.56
	Uptime (h)	8750.44
Fog	Availability	0.9994
	Unavailability	0.0006
	# of 9's	3.2717
	Downtime (h)	4.68
	Uptime (h)	8755.32

Source: Elaborated by the author.

The enhancement of availability can be achieved through the implementation of redundancy, which is a well-established mechanism. Redundancy can be classified into three types: cold, warm, and hot standby, each suitable for different scenarios based on the criticality of the process and the consequences of equipment failures (MELO et al., 2018). Cold-standby redundancy is typically employed for non-critical services where human intervention is deemed acceptable, and time is not of utmost importance. Warm-standby redundancy, on the other hand, is utilized when the response to failure and time are significant but not critical, allowing for temporary outages. Lastly, hot-standby redundancy offers instantaneous process correction in the event of a failure, prioritizing time and security with no tolerance for system interruptions. The selection of the appropriate

redundancy type depends on the specific requirements and constraints of the system at hand.

Table 11 – Results of the hot-standby redundancy.

Node	Metric	Values
Edge	Availability	0.9999988
	Unavailability	0.0000012
	# of 9's	5.9381
	Downtime (h)	0.0101
	Uptime (h)	8759.9899
Fog	Availability	0.9999997
	Unavailability	0.0000003
	# of 9's	6.5434
	Downtime (h)	0.0025
	Uptime (h)	8759.9975

Source: Elaborated by the author.

Given the critical nature of our surveillance environment, where system downtime is unacceptable, we have opted for hot-standby redundancy. By implementing this mechanism, we have significantly reduced the downtime of the edge environment to 0.0101 hours, equivalent to a mere 36.36 seconds in a year. Similarly, the downtime of the fog environment has been minimized to 0.0025 hours, representing just 9.02 seconds per year. These remarkable improvements in availability have resulted in an impressive number of nines for both the edge and fog nodes. The edge node now achieves a level of availability represented by 5.9381 nines, transforming it into a highly available system. Likewise, the fog node demonstrates exceptional availability with 6.5434 nines. The addition of a new physical node has elevated the system from a well-managed state to one with exceptional availability. Table 11 provides a comprehensive summary of the obtained results.

6.1.3 Case Study III - Capacity-oriented availability (COA)

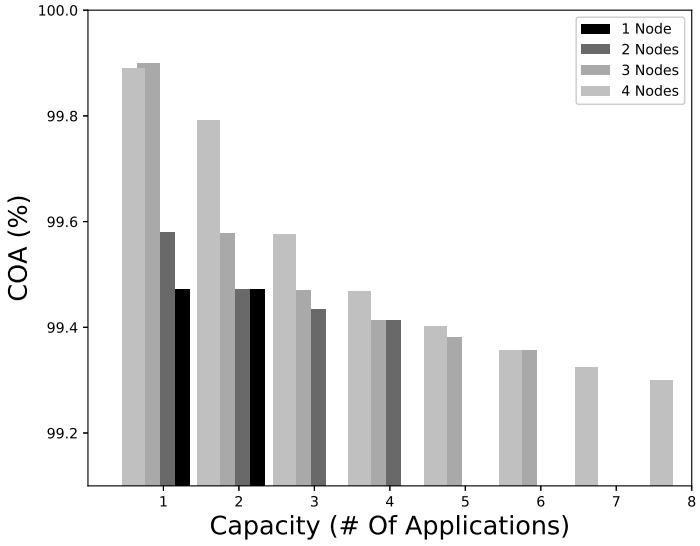
In this case study, our focus lies in the estimation of the capacity-oriented availability (COA) of our environment. COA takes into account the availability or unavailability states and their impact on the service provided (MELO et al., 2017; Sousa et al., 2015; PEREIRA et al., 2021). The primary objective of this study is to determine the actual capacity, specifically the number of running applications, considering the infrastructure's availability. To achieve this, we have employed a Markov Chain model, as depicted in Figure 12, to represent the system's states. By computing the probabilities associated with each state and multiplying them by the corresponding capacity, we can determine the weighted average of the available applications. Finally, we divide this sum by the

maximum system capacity to obtain the COA. In the context of edge and fog computing scenarios, the COA is directly linked to the number of applications that the physical nodes can support.

To summarize, the COA of our environments can be calculated by dividing the average number of available applications by the maximum number of applications that our environment can accommodate. Utilizing our analytical model, represented by Equation 5.7, we can determine the COA for various configurations of the edge and fog infrastructure, as long as they adhere to the abstraction framework presented in Section 4.

In this particular case study, we have assumed that a single physical node can handle one application per CPU thread. For instance, a physical node equipped with a quad-core CPU featuring eight threads can support up to eight applications. It is important to note that each application is responsible for processing a camera stream. Accordingly, our physical edge node, with two threads, can accommodate two applications. Conversely, the fog node, with eight threads, can support eight applications. For the purpose of this case study, we have examined the COA for four physical nodes, employing the input values from Table 9. It is worth emphasizing that Equation 5.7 enables us to assess the COA for any desired combination of physical nodes and applications.

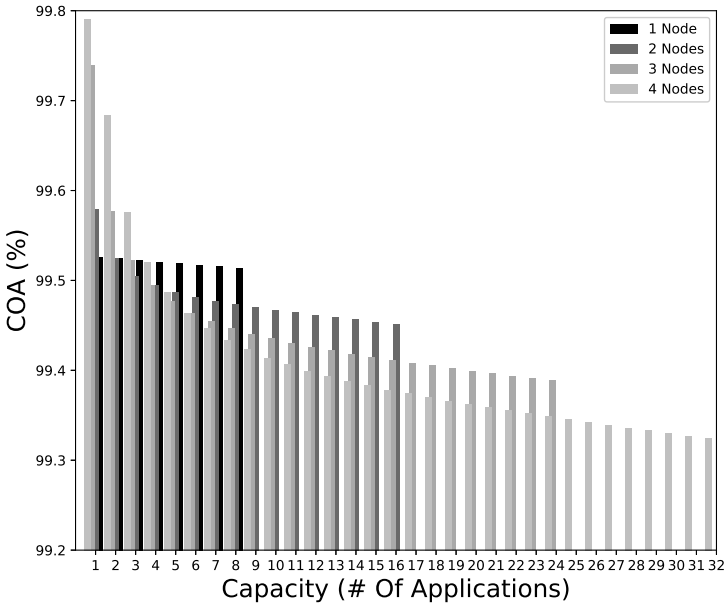
Figure 22 – COA x Edge system's capacity.



Source: Elaborated by the author.

The capacity axis depicted in Figure 22 and Figure 23 corresponds to the number of applications operational within the environment. Upon examining these figures, we observe an inverse relationship between resource availability and capacity-oriented availability. Specifically, as the number of available resources increases, the capacity-oriented availability decreases. This phenomenon arises due to the heightened likelihood of resource failures as the quantity of resources increases.

Figure 23 – COA x Fog system's capacity.



Source: Elaborated by the author.

Figure 22 and Figure 23 provide evidence that the capacity-oriented availability of the fog environment surpasses that of the edge environment. This outcome aligns with our expectations, given the lower failure rate of the fog node compared to the edge node. However, it is important to note that the suitability of an edge computing environment can still be sufficient depending on the specific needs of users or companies. Despite its comparatively lower availability when compared to fog computing, as demonstrated in the subsequent case study, an edge computing environment may fulfill the requirements of certain scenarios.

6.1.4 Case Study IV - Cost analysis

There are numerous advantages associated with the utilization of edge and fog computing, encompassing techniques for efficient resource utilization and the reduction of operational costs by stabilizing infrastructure. While these benefits are evident, the precise financial advantages that companies may derive from adopting edge and fog computing remain uncertain. To address this issue, a comprehensive cost analysis is conducted in this case study.

It is crucial to acknowledge the following assumptions made in this particular study:
The cost analysis in this case study is conducted under the following assumptions:

- The costs related to software, training, licensing, and maintenance are assumed to be identical for both the edge and fog infrastructures. This assumption is made based on the premise that the same software environment is employed in both environments.

- Monitoring costs are considered to be the same for both the edge and fog infrastructures. This assumption is based on the notion that monitoring software may have its own infrastructure, leading to comparable costs.
- Costs related to bandwidth and network usage are disregarded in this analysis. This assumption is made to simplify the cost evaluation and focus solely on other factors.
- The costs associated with security and compliance are not considered in this analysis. It is assumed that each company will already have its dedicated information security team to handle these aspects independently.

To initiate our analysis, we conducted a thorough investigation of the procurement costs associated with each component required to establish the edge and fog environments. It is important to note that both the edge and fog environments consist of a single physical node. To gather pricing information, we referred to reputable online platforms, including Dell and Amazon, specifically focusing on websites based in the United States of America. The price search was conducted between May 25th, 2020, and June 1st, 2020. In Table 12, we present the corresponding online stores and the minimum costs observed for each equipment.

Table 12 – Acquisition Costs

Edge Environment		
Component	**Cost (USD)**	**Store**
Raspberry Pi Zero W	34.99	Amazon
TP-Link AC1750 WiFi	64.99	Amazon
C4Labs Zebra Raspberry Rack	34.99	Amazon
Fog Environment		
Component	**Cost (USD)**	**Store**
PowerEdge R240	499.00	Dell
Switch Huacomm 5-Port	49.99	Amazon
Rack StarTech.com 12U	215.99	Amazon

Source: Elaborated by the author.

The subsequent step entails the computation of the annual energy costs associated with our environment. Each component exhibits distinct power consumption levels (W), necessitating a meticulous examination of the manufacturer's specifications. By employing Equation 6.1[2], we can estimate the energy consumption (kWh) of both the edge and fog environments. It is important to note that Equation 6.1 is also utilized to assess the

[2] http://dell-ui-eipt.azurewebsites.net/

energy consumption of the edge, as it appropriately reflects the energy consumption of the Raspberry[3].

$$\mathrm{E} = \frac{\text{Power } \times \mathrm{NHD} \times \mathrm{NDY}}{1000}(kWh), \tag{6.1}$$

where NHD represents the number of operational hours per day, while NDY corresponds to the number of days per year that the device remains in operation. In accordance with the International System of Units, the unit of power is expressed in Watts. Consequently, in Equation 6.1, we divide the power value by 1000 to convert it into kilowatt-hours (kWh). For our analysis, we assume that each equipment operates continuously, encompassing 24 hours per day and seven days per week. Additionally, we consider the average electricity price of 13.19 cents per kWh in the United States of America, as of April 2020[4]. As a result, we derive the values presented in Table 13, where the *Cost (USD)* column represents the annual expenses in U.S. dollars.

Table 13 – Energy consumption by year

Edge Environment		
Component	**Power (W)**	**Cost (USD)**
Raspberry Pi Zero W	3	3.46
TP-Link AC1750 WiFi	24	27.73
Fog Environment		
Component	**Power (W)**	**Cost (USD)**
PowerEdge R240	65	75.10
Switch Huacomm 5-Port	65	75.10

Source: Elaborated by the author.

As evidenced by the analysis, the cost of establishing and maintaining an edge environment is considerably lower compared to a fog environment. However, the notable disparity lies in the capacity-oriented availability, as demonstrated in the preceding case study. The fog environment, consisting of a solitary physical node employing the server specified in Table 12, possesses the capability to concurrently operate up to 32 applications, owing to its robust processor power. Conversely, the edge environment is only capable of accommodating two applications. Consequently, to support the execution of the 32 applications, it would necessitate the deployment of 16 Raspberry Pi Zero W devices, incurring an approximate cost of US$560.00. For illustrative purposes, let us examine two scenarios: an airport surveillance system and a farm surveillance system.

[3] https://www.pidramble.com/wiki/benchmarks/power-consumption
[4] https://www.electricchoice.com/electricity-prices-by-state/

Scenario #01

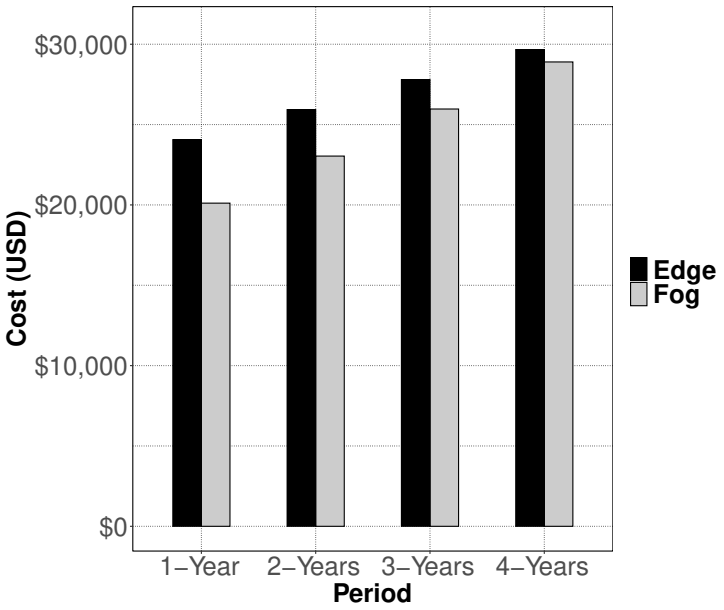
Surveillance systems incorporating facial recognition technology have become ubiquitous in airports worldwide. These systems offer enhanced speed and accuracy in identifying individuals, surpassing human visual capabilities (PETRESCU, 2019). However, when such applications are hosted in a cloud environment, latency can be a critical factor in preventing unauthorized access to airplanes. To mitigate this latency issue, edge and fog computing environments have emerged as viable alternatives.

In the context of a surveillance system employing facial recognition technology within an airport, ensuring maximum availability becomes imperative. Availability plays a crucial role in determining the probability of a system operating during a specified period, accounting for both failures and repairs within this timeframe (MACIEL et al., 2012). In this scenario, we assume the hypothetical airport comprises 1000 cameras, with each camera requiring a dedicated application. Consequently, considering the environments defined by the components listed in Table 12, the edge computing environment can accommodate 2 applications per physical node, while the fog computing environment can support 32 applications (i.e., $L = 2$ and $L = 32$ in Equation 5.2, respectively). Accommodating the 1000 cameras necessitates 500 physical nodes for the edge environment (i.e., $M = 500$ in Equation 5.2), whereas the fog environment requires 32 physical nodes (i.e., $M = 32$ in Equation 5.2). The total cost of equipment acquisition for the edge computing environment would amount to approximately US$22,193.7, encompassing 500 Raspberry Pi devices, 5 TP-Link routers, and 125 Raspberry Pi racks. Conversely, establishing a fog computing environment would incur total equipment acquisition expenses of around US$17,181.89, comprising 32 Dell PowerEdge R240 servers, 7 Huacomm 5-Port switches, and 4 racks. Figure 24 depicts the annual expenses of edge and fog computing, factoring in energy consumption as well. In the first year, accounting for acquisition costs and energy consumption, the edge environment would incur expenses of approximately US$24,062.35, while the fog environment would amount to US$20,110.79.

Assuming that all applications need to be running to consider the system available, we need 1000 applications running, which means 1000 out of 1000 applications need to be working. So using the inverse values in Table 5 as inputs of the model represented by Equation 5.2, which represents the K out of N availability (i.e., $N = M \times L$), we have the values depict in Table 14.

As illustrated in Figure 24, the initial four-year period exhibits higher effective expenses for the edge environment compared to the fog environment. Nevertheless, the discrepancy in effective expenses gradually diminishes over the years. However, it is worth noting that the edge environment, lacking redundancy, demonstrates unacceptable availability levels (as indicated in Table 14). This outcome can be attributed to the presence of numerous components, which inherently increases the likelihood of failures. To address this concern, the utilization of redundancy becomes crucial. However, it is important to

Figure 24 – Effective Expenses Scenario #01 - Edge x Fog.



Source: Elaborated by the author.

Table 14 – Results of the airport facial recognition system.

Node	Metric	Values
Edge	Availability	0.0086
	Unavailability	0.9914
	# of 9's	0.0037
	Downtime (h)	8684.30
	Uptime (h)	75.70
Fog	Availability	0.9994
	Unavailability	0.0006
	# of 9's	3.2717
	Downtime (h)	4.68
	Uptime (h)	8755.32

Source: Elaborated by the author.

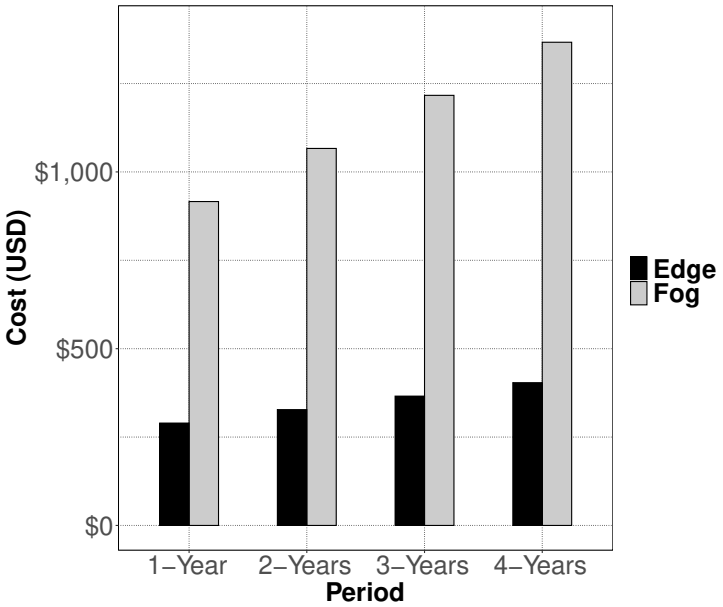
acknowledge that incorporating redundancy in the edge environment would entail additional effective expenses. Therefore, based on the findings of this scenario, deploying the surveillance system using the fog environment is strongly recommended.

Scenario #02

Suppose an individual intends to deploy a surveillance system on their farm to detect intruders. After analysis, the conclusion was reached that ten cameras would suffice to adequately cover the entire area. However, if this intruder detection system were to operate

in a cloud environment, significant challenges might arise, particularly in rural settings where the Internet connection may exhibit even higher latency.

Figure 25 – Effective Cost Scenario #02 - Edge x Fog.



Source: Elaborated by the author.

Taking into account the edge and fog environments with the components specified in Table 12 and their corresponding characteristics, the edge environment supports the execution of two applications per physical node (i.e., $L = 2$ in Equation 5.2), while the fog environment has a capacity of running 32 applications per physical node (i.e., $L = 32$ in Equation 5.2). Consequently, to accommodate ten cameras, the edge environment necessitates five physical nodes (i.e., $M = 5$ in Equation 5.2), while the fog environment requires a single physical node (i.e., $M = 1$ in Equation 5.2). The total cost of acquiring equipment for the edge computing environment amounts to approximately US$275.00, involving five Raspberries, one TP-Link router, and one Raspberry rack. Conversely, the fog computing environment entails a total equipment acquisition cost of around US$766.00, encompassing one Dell PowerEdge R240 server, one Huacomm 5-Port switch, and one rack. The annual cost, considering energy consumption, edge and fog computing, is illustrated in Figure 25.

It should be emphasized that each application is responsible for processing a single camera stream. If we consider the system to be available only when all applications are running simultaneously, a total of ten applications are required. In other words, a 10 out of 10 application availability is needed. By utilizing the analytical model described by Equation 5.2, the availability of each environment can be determined. The outcome is presented in Table 15.

As depicted in Table 15, the availability of the fog environment significantly surpasses

Table 15 – Results of the farm's intruder detection system.

Node	Metric	Values
Edge	Availability	0.9535
	Unavailability	0.0465
	# of 9's	1.3334
	Downtime (h)	406.47
	Uptime (h)	8353.53
Fog	Availability	0.9994
	Unavailability	0.0006
	# of 9's	3.2717
	Downtime (h)	4.68
	Uptime (h)	8755.32

Source: Elaborated by the author.

that of the edge environment, indicating that the fog environment is better suited for a surveillance system that requires high availability. However, it is crucial to consider the substantial disparity in the effective annual expenses between the two environments. The expenses associated with the fog environment are more than double the expenses incurred by the edge environment. This implies that even with the implementation of hot-standby redundancy, the edge environment remains more cost-effective while achieving higher availability. Hence, for this particular scenario, the edge environment is the recommended choice.

6.1.5 Insights from the case study

In this case study, it became clear that continuous-time Markov chains (CTMCs) serve as a powerful mathematical framework for analyzing and predicting the behavior of complex systems with stochastic dynamics and state transitions. The utilization of CTMCs enabled a comprehensive understanding of the intricate nature of face recognition systems, particularly in edge-fog-cloud environments.

Additionally, the case study highlighted the significance of considering factors such as system failures, repair times, and the impact of resource capacity on availability. The examination of various scenarios, including the application of redundancy techniques, shed light on the trade-off between availability and cost.

Ultimately, the case study emphasized the importance of carefully evaluating and selecting the appropriate computational environment based on specific requirements and constraints, taking into account factors such as latency, capacity-oriented availability, and financial considerations. Overall, the case study provided valuable insights into the availability aspects of face recognition systems and the significance of adopting analytical

models for performance evaluation and decision-making processes.

6.2 PERFORMABILITY MODELS

In this section, we leverage a face recognition system to assess the performability models. The significance of ensuring security and safety underscores the importance of evaluating the availability and performance of such systems. By employing modeling techniques, we can comprehensively explore a broad spectrum of possibilities and potential outcomes.

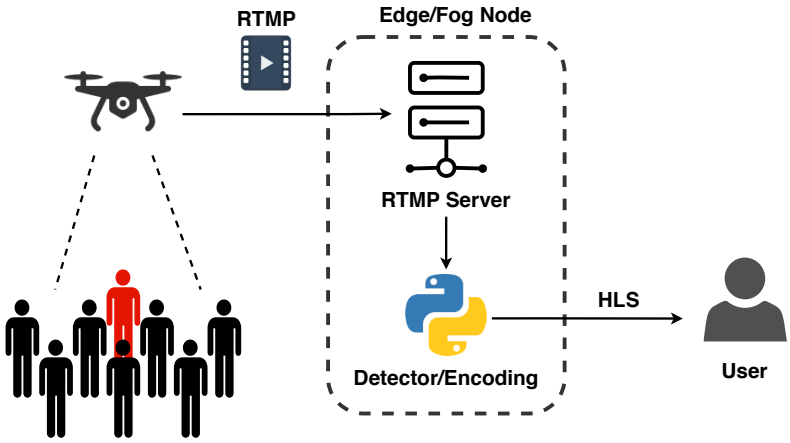
6.2.1 Case Study I - Performability model validation

In this case study, we delineate the experimental procedures employed to assess the representativeness of our performability models in a real-world setting. Our experimental environment comprised a drone, three computers (including a fog node and two clients), and a Raspberry Pi[5] serving as an edge node. The drone utilized was a Dji Mavic Pro Platinum[6], tasked with capturing video footage and transmitting it to the fog and edge infrastructure. The captured video material was formatted as MP4, featuring a frame rate of 30 frames per second and a resolution of 3840×2160. The fog node operated on XenServer 7.2.0, functioning as the hypervisor platform, equipped with 8 GB of RAM, 1 TB of storage, and an Intel Core i5 3.20 GHz quad-core CPU with eight threads. The remaining two client computers boasted 8 GB of RAM, 1 TB of storage, and an Intel i7-4510U 2.00 GHz CPU. As for the Raspberry Pi functioning as the edge node, it possessed a 1 GHz single-core CPU with two threads and 512 MB of RAM. Both the fog node and edge node executed a Python script tailored for face recognition purposes, alongside an RTMP server (e.g., Nginx[7]) responsible for receiving the drone's video stream. Figure 26 and Figure 9 provide visual illustrations of the foundational components within the baseline testbed environment.

The VGGFace2 model, developed by researchers at the Visual Geometry Group at Oxford (CAO et al., 2018), was employed for face recognition in our study. To utilize pretrained models, we employed the Keras framework[8]. Within the same Python script, we encoded the processed video to deliver an HLS (HTTP Live Streaming) stream using the Mux Plugin[9]. The HLS chunk size was set to 3 seconds (HOQUE et al., 2015). After constructing the experimental environment, we conducted multiple case studies by injecting failures through a Python script and sending requests using the HLS JMeter plugin[10], developed by BlazeMeter Labs. This streaming plugin allowed us to evaluate the perfor-

[5] https://www.raspberrypi.org/
[6] https://www.dji.com/br/mavic
[7] https://www.nginx.com/
[8] https://keras.io/
[9] https://mux.com/for/python/
[10] https://www.blazemeter.com/blog/HLS-3.0-Release

Figure 26 – Testbed environment to validate the models.



Source: Elaborated by the author.

mance of live-streaming video servers under various load conditions. Typically, JMeter[11] can handle more than 500 threads from a single machine, depending on the script and the machine's characteristics. It is important to note that we closely monitored the health of the machines generating the workload, as unreliable results would be obtained if they became saturated.

In our evaluation, we focused on a baseline infrastructure consisting of two physical nodes, namely the edge and fog nodes, with each node running a single application. We utilized failure and repair rates obtained from literature (LISBOA et al., 2018; DANTAS et al., 2016; PEREIRA et al., 2021). Assuming exponentially distributed time to failure and repair, the failure and repair rates were derived as the inverses of the mean time to failure (MTTF) and mean time to repair (MTTR) values. Table 16 presents the adopted MTTF and MTTR values.

To evaluate the performability of the systems, we employed a fault injection technique, as faults are often unpredictable or occur infrequently (SOUZA et al., 2013; BRILHANTE et al., 2014; JAMMAL et al., 2018). The fault injection mechanism allowed us to control and monitor the experiment during fault events, accelerating the occurrence of faults and observing their impact on the system. When a fault occurred, it introduced a modification in the system, leading to an erroneous state. The fault could also propagate through the system, affecting the availability of the provided service.

For our baseline infrastructure, we adopted a minimal component configuration for the edge and fog nodes, consisting of the essential elements required to provide the service. The edge node comprises the Raspberry operating system, a Python application responsible for face recognition and HLS stream encoding, and an Nginx RTMP server to receive the drone's video streaming. The fog node shares the same components as the edge node, with

[11] https://jmeter.apache.org/

Table 16 – MTTF and MTTR from literature.

Node	Components	MTTF (h)	MTTR (h)
Edge	Raspberry	4767.8	3.48
	OS	2880	1
	Python App	217.8	0.46
	Nginx App	217.8	0.46
Fog	Hardware	8760	1.67
	OS	2880	1
	Cont. Management	2880	1
	Python App	217.8	0.46
	Nginx App	217.8	0.46

Source: Elaborated by the author based on Dantas et al. (2016).

the only difference being the hardware platform, where a personal computer is utilized.

To verify the validity of our models, it was necessary to subject the system to stress in order to accelerate the mean time to failure (MTTF) of the components. For this purpose, we developed a fault injector capable of introducing faults into each component of the nodes. Similarly to our previous assumptions, we considered the time to failure and repair to follow an exponential distribution, resulting in failure and repair rates being the reciprocals of the MTTF and mean time to repair (MTTR), respectively (LISBOA et al., 2018; DANTAS et al., 2016). In order to expedite the MTTF, we increased its rate by a factor of 876, effectively compressing one year into a 10-hour timeframe. The values employed in the fault injector are presented in Table 17. As a result of this accelerated MTTF, the system's availability experiences a decrease.

Table 17 – MTTF and MTTR in the fault injector.

Node	Components	MTTF (h)	MTTR (h)
Edge	Raspberry	5.4	3.48
	OS	3.28	1
	Python App	0.24	0.46
	Nginx App	0.24	0.46
Fog	Hardware	10	1.67
	OS	3.28	1
	Cont. Management	3.28	1
	Python App	0.24	0.46
	Nginx App	0.24	0.46

Source: Elaborated by the author.

In Table 16 and Table 17, we present the values for the mean time to failure (MTTF)

and mean time to repair (MTTR) obtained from the literature and those accelerated for use in the fault injector, respectively. However, in order to incorporate these values into our performability models, we need to perform the abstraction process described in Section 4. This abstraction involves establishing the relationship between the components listed in Table 17 and the corresponding components specified in Table 18, as depicted in Figure 9. In the abstraction, the physical node in Table 18 represents the hardware and operating system, while the application represents the Nginx server and the Python script. To obtain the corresponding values for Table 18, the abstraction process was conducted using the Reliability Block Diagram (RBD) model, leveraging the Mercury tool (SILVA et al., 2015; PINHEIRO et al., 2021).

Table 18 – MTTF and MTTR for model.

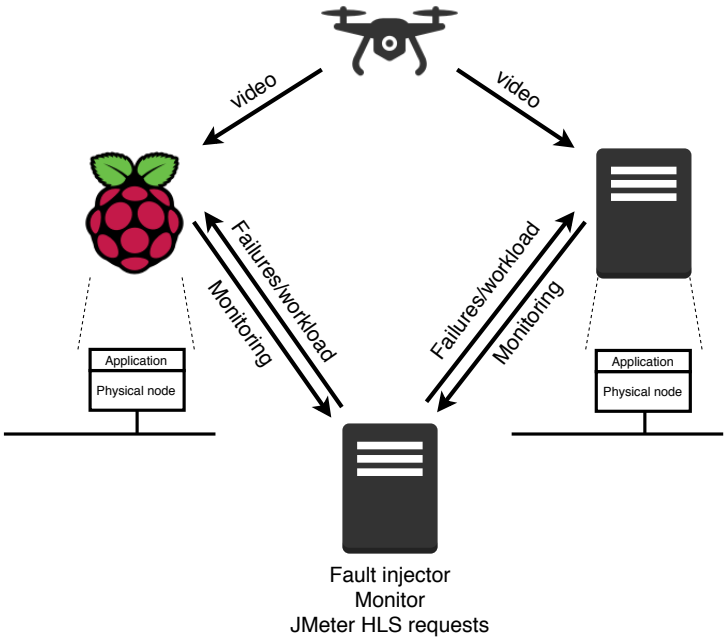
Node	Components	MTTF (h)	MTTR (h)
Edge	Physical Node	2.04	2.34
	Application	0.34	0.44
Fog	Physical Node	2.48	1.29
	Application	0.34	0.44

Source: Elaborated by the author.

Figure 27 provides a visual representation of the deployment of our fault-injection environment and illustrates the workload generated for the video stream. In our experimental setup, we utilized a drone to capture images and transmit them to the designated nodes, as previously described. The fault injector, implemented in Python, plays a pivotal role in this environment by specifying the components that will experience failures and be subject to monitoring. To facilitate fault injection, the fault injector creates a dedicated thread for each component, responsible for simulating and injecting the respective faults. The time to failure for each component follows an exponentially distributed distribution, as indicated in Table 17. Consequently, each thread generates a random number adhering to this distribution, awaiting the appropriate time to initiate fault injection. Additionally, we employ another thread dedicated to monitoring the system's availability. This monitoring thread records the system's state as either "up" (denoted as U) or "down" (denoted as D) and stores this information in a text file. The monitoring process samples the system's state at regular intervals of 10 seconds.

Utilizing the depicted environment illustrated in Figure 27, we conducted two separate experiments to evaluate the behavior of the fog and edge nodes. Each experiment was conducted over a duration of 96 hours, resulting in the generation of two distinct text files: one for the edge node and another for the fog node. These text files contained a series of recorded states denoted by U (indicating system uptime) and D (indicating system downtime). By analyzing the occurrence of these states, we were able to calculate the

Figure 27 – Fault-injection environment and workload generator.



Source: Elaborated by the author.

time to failure and time to repair. For example, if we observe 50 consecutive occurrences of U before transitioning to a D state, we can infer that the system experienced 50 intervals of 10 seconds (the sampling interval), resulting in a cumulative uptime of 500 seconds within that particular interval. Similar calculations were performed to determine the time to repair.

To generate the workload, we employed the JMeter benchmark tool coupled with an HLS JMeter plugin. This combination facilitated the monitoring of response times and discard rates for each node, while utilization was observed through an auxiliary monitoring tool. The JMeter plugin simulated user access to a video streaming server utilizing the HLS protocol. It offered flexibility in configuring various parameters such as stream type, playback time, network bandwidth, and device resolution. Table 19 provides an overview of the specific configuration parameters employed within the HLS plugin to generate the workload. We opted for utilizing the Hypertext Transfer Protocol (HTTP) for video streaming, as it is a prevalent practice for devices with limited bandwidth (PEREIRA et al., 2020).

Consequently, we conducted the experiment and closely monitored the system's performance to obtain performability metrics for both the edge and fog environments. As anticipated, the system's availability significantly impacted its overall performance, as income requests were discarded during periods of system downtime. Upon completing the 96-hour experiment, we accumulated approximately 30,500 sample points for both the edge and fog environments. This substantial amount of data allowed us to calculate 95%

Table 19 – HLS plugin configuration parameters.

Variables	Values
Video Type	live stream
Play Back Time	whole video
Protocol	http
Bandwidth	max available
Resolution	max available

Source: Elaborated by the author.

confidence intervals for the performability metrics.

Assuming rate values for the edge system as $\lambda = 900\,tph$ (transactions per hour) and $\mu = 42.01\,tph$, and corresponding values for the fog system as $\lambda = 900\,tph$ and $\mu = 270.06\,tph$ (YE, 2017; LONGBOTTOM, 2017), we utilized the analytical models described in Section 5.2. With the specific component values provided in Table 18, we derived the performability metrics for both the edge and fog nodes. These point estimations, along with their respective confidence intervals, are summarized in Table 20, alongside the model results.

It is noteworthy that the confidence intervals encompass the estimations derived from our models, indicating that there is no evidence to suggest that our models fail to accurately represent the real environment. Hence, we can reasonably assume that our validated models reliably represent the behavior of the edge and fog nodes. This outcome enables us to utilize these models to calculate performability metrics for future scenarios, develop strategies to enhance performance levels, and conduct cost analyses. It is important to mention that we did not validate the waiting time model, as JMeter does not provide measurements for this particular aspect.

Table 20 – Validation result.

Metrics	Node	Model	CI 95% of the system
Utilization	**Edge**	0.2307	$0.1999 < \hat{\theta} < 0.2616$
	Fog	0.2681	$0.2342 < \hat{\theta} < 0.2944$
Resp. Time (h)	**Edge**	0.0094	$0.0053 < \hat{\theta} < 0.0114$
	Fog	0.0018	$0.0005 < \hat{\theta} < 0.0025$
Disc. Rate	**Edge**	174.20	$172.43 < \hat{\theta} < 175.74$
	Fog	185.55	$184.17 < \hat{\theta} < 187.32$

Source: Elaborated by the author.

The performance of the nodes is observed to have a direct correlation with the system's availability. This relationship is particularly evident in the low utilization experienced by each node. When a node encounters downtime, the utilization metric drops to zero,

significantly impacting the average value of this metric. In the subsequent section, we will delve into a detailed investigation of this phenomenon.

6.2.2 Case Study II - Performability vs performance models, and cost analysis

This case study is structured into two distinct sections. The first section utilizes the proposed analytical model to explore the impact of availability on the performance of the aforementioned environments. The second section focuses on conducting a comprehensive cost analysis to determine the most suitable paradigm for specific scenarios. It is important to reiterate that analytical models offer distinct advantages in terms of scalability compared to other modeling techniques such as Reliability Block Diagram and Stochastic Petri Nets. With analytical models, it becomes feasible to evaluate numerous scenarios with reduced computational costs, as highlighted by Jain et al. (JAIN, 1990).

Comparison between performability and performance models

In the subsequent analysis, we employ the MTTF and MTTR values for the edge and fog nodes as obtained from the literature (see Table 16). These values are utilized in the Mercury tool (SILVA et al., 2015; PINHEIRO et al., 2021) to derive the corresponding abstraction values, which are presented in Table 21.

Table 21 – The abstraction MTTF and MTTR.

Node	Components	MTTF (h)	MTTR (h)
Edge	Physical Node	1795.45	1.93
	Application	108.89	0.46
Fog	Physical Node	2167.42	1.16
	Application	108.89	0.46

Source: Elaborated by the author.

With respect to the performance rates of the nodes, we have adopted specific values for the edge and fog systems. The edge system is characterized by a rate of arrivals, λ, equal to 900 transactions per hour (tph), and a service rate, μ, of 42.01 tph. Similarly, the fog system exhibits a rate of arrivals, λ, of 900 tph, and a service rate, μ, of 270.06 tph. It is important to note that our model assumes a single-edge physical node capable of running two applications, while the single fog physical node can accommodate up to eight applications. Consequently, for our model, the value of L is set to two for the edge environment and eight for the fog environment. Each application in the edge environment has a capacity of two, with one chunk of data being processed and another one in the queue. Similarly, each application in the fog environment has a capacity of two.

In this investigation, we do not consider redundancy, which implies that the value of M is equal to one for both the edge and fog environments. The first two levels of our Markov

chain model, as illustrated in Figure 13, represent the availability of our environments, while the third level pertains to performance. We will analyze the performance of the environments with and without accounting for availability, and subsequently compare the results. For the pure performance models, we employ the parameters $m = 2$ and $C = 4$ for the edge node, and $m = 8$ and $C = 16$ for the fog node. The following equations represent the performance models of the nodes without considering availability:

$$U = \frac{\lambda(1 - \pi_C)}{m \times \mu}, \tag{6.2}$$

$$RT = \frac{\sum_{n=1}^{C} n\pi_n}{\lambda(1 - \pi_C)}, \tag{6.3}$$

$$WT = \frac{\sum_{n=m}^{C} (n - m)\pi_n}{\lambda(1 - \pi_C)}, \tag{6.4}$$

$$DR = \lambda\pi_C. \tag{6.5}$$

By conducting this investigation, we can compare the impact of environment availability on performance. Model-based analysis techniques, such as Markov chains, offer an appealing approach due to their cost-effectiveness in conceptualizing and evaluating various scenarios. These methods enable stakeholders to efficiently analyze multiple scenarios with different parameters (ANDRADE; NOGUEIRA, 2019).

Table 22 – Investigation of performance x performability models.

Node	Metric	Performance	Performability
Edge	Utilization	0.9995	0.9984
	Response (h)	0.04635	0.04650
	Waiting (h)	0.02257	0.02269
	Discard (r/h)	814.014	815.476
Fog	Utilization	0.4277	0.4159
	Response (h)	0.00371	0.00421
	Waiting (h)	1.79×10^{-5}	1.94×10^{-5}
	Discard (r/h)	0.0107	0.0169

Source: Elaborated by the author.

As depicted in Table 22, the system's availability has a notable impact on its performance. Higher availability leads to performability metrics that closely align with the performance model. The utilization in the performability model is lower compared to the performance model due to the direct influence of availability on overall utilization. Consequently, the system fails to achieve its full processing capacity, resulting in a waste of

processing power. Additionally, we observe an increase in overall waiting time, response time, and discard rate in the performability models. Although the differences may not be significant for certain metrics, they become more pronounced when scaling up the number of physical nodes and applications. Performability models provide a more comprehensive representation of the system's total capacity compared to pure performance models (SMITH; TRIVEDI; RAMESH, 1988).

Cost analysis

The utilization of edge and fog computing presents several advantages, such as optimizing resource utilization and reducing latency. However, the actual financial benefits that organizations can derive from adopting these computing paradigms remain uncertain. To shed light on this matter, we conduct a cost analysis in this subsection.

During this analysis, it is important to consider the following assumptions:

- The costs associated with software, training, licensing, and maintenance are assumed to be the same for both edge and fog infrastructures, as we assume the utilization of the same software environment.
- Monitoring costs are considered to be equal since monitoring software may have its own dedicated infrastructure.
- Bandwidth and network costs are disregarded in this analysis.
- Security and compliance expenses are not accounted for, assuming that each company already maintains its information security team.

To initiate our analysis, we meticulously investigate the acquisition costs of each component required to establish the edge and fog environments. In this case study, it is assumed that both the edge and fog environments consist of a solitary physical node. To gather the necessary information, we extensively surveyed various websites based in the United States of America, including reputable sources such as Dell and Amazon. This search was conducted over the period from August 25th to September 2nd. The resulting findings, comprising the store details and the corresponding lowest equipment costs, are presented in Table 23.

The subsequent step entails calculating the annual energy cost of our environment. As each component exhibits distinct power consumption (P - measured in watts), it was imperative to gather the manufacturer's specifications for accurate assessment. Equation 6.6 facilitates the estimation of energy consumption (E - measured in kilowatt-hours) for both the edge and fog environment[12]. Additionally, Equation 6.6 was employed to determine

[12] http://dell-ui-eipt.azurewebsites.net/

Table 23 – Acquisition Costs

Edge Environment		
Component	**Cost (USD)**	**Store**
Raspberry Pi 2 Model B	50.48	Amazon
TP-Link AC1750 WiFi	64.99	Amazon
C4Labs Zebra Raspberry Rack	34.99	Amazon
Fog Environment		
Component	**Cost (USD)**	**Store**
PowerEdge R240 Rack	499.00	Dell
Switch Huacomm 5-Port	49.99	Amazon
Rack StarTech.com 12U	215.99	Amazon

Source: Elaborated by the author.

the edge consumption, as it serves as a representative of Raspberry's energy consumption as well[13].

$$\mathrm{E} = \frac{\text{Power } \times \mathrm{NHD} \times \mathrm{NDY}}{1000}(kWh). \tag{6.6}$$

The variable NHD represents the number of hours per day, while NDY corresponds to the number of days per year. In our analysis, we assume that each equipment operates continuously for 24 hours per day and seven days per week. Furthermore, we consider the average electricity price to be 13.19 cents of American Dollar per kilowatt-hour (kWh) in the United States of America, as updated in April 2020[14]. Based on these assumptions, we have computed the values presented in Table 24, where the *Cost (USD)* column signifies the annual expenses in U.S. dollars.

Table 24 – Energy consumption by year

Edge Environment		
Component	**Power (W)**	**Cost (USD)**
Raspberry Pi 2 Model B	3	3.46
TP-Link AC1750 WiFi	24	27.73
Fog Environment		
Component	**Power (W)**	**Cost (USD)**
PowerEdge R240 Rack	65	75.10
Switch Huacomm 5-Port	65	75.10

Source: Elaborated by the author.

[13] https://www.pidramble.com/wiki/benchmarks/power-consumption
[14] https://www.electricchoice.com/electricity-prices-by-state/

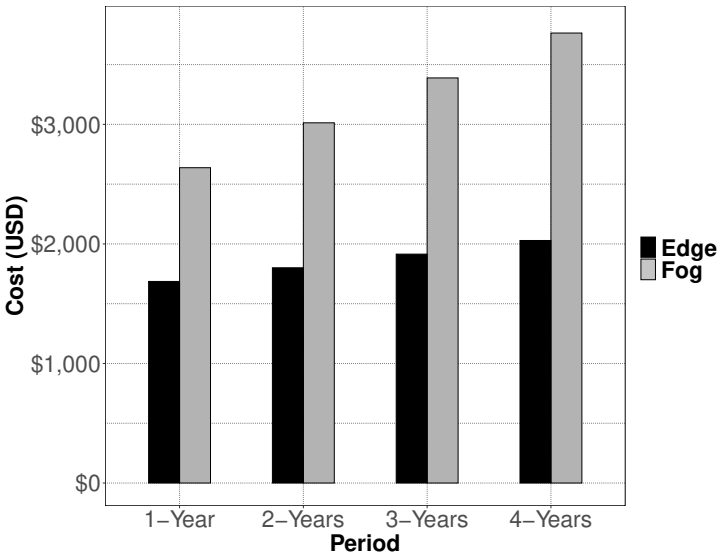
It is evident that constructing and maintaining an edge environment is considerably more cost-effective than a fog environment. However, the key distinction lies in their performance capabilities, as demonstrated in the previous case study. The fog environment, consisting of a single physical node utilizing the server specified in Table 23, can concurrently run up to 32 applications owing to its powerful processor. Conversely, the edge environment has a limited capacity of running only four applications. Therefore, to accommodate the 32 applications, it would necessitate the deployment of 8 Raspberry Pi 2 Model B devices, amounting to an approximate cost of US$400.00. To illustrate this further, let us consider two scenarios: a surveillance system at a corporate facility and a storage facility on a farm.

Scenario #01: in many organizations, the inability of security guards to recognize employees can lead to significant financial losses resulting from potential burglaries (PATIL et al., 2020). To address this issue, facial recognition-based surveillance systems are being implemented in companies worldwide. However, when these surveillance systems are deployed in cloud environments, latency can become a critical factor in preventing intruders from accessing company premises. As a solution, edge or fog computing environments can be employed to mitigate latency issues.

In the context of a company utilizing a facial recognition surveillance system, it is crucial to ensure that the system meets performance requirements specified in service level agreements (SLAs). Performability plays a vital role in this regard, as it assesses the probability of achieving desired performance metrics during the expected operational period, accounting for potential failures and repairs (MACIEL et al., 2012). In our scenario, we consider a hypothetical company with 100 cameras (PATIL et al., 2020), each requiring its dedicated application. Based on this information and the component specifications presented in Table 23, we explore the edge and fog environments. The edge environment can support four applications per physical node (i.e., $L = 4$), while the fog environment can accommodate 32 applications per physical node (i.e., $L = 32$). To cater to the 100 cameras, the edge environment would require 25 physical nodes (i.e., $M = 25$), whereas the fog environment would need 4 physical nodes (i.e., $M = 4$). The total equipment acquisition costs for the edge computing environment are estimated to be approximately US$1,571.92, involving 25 Raspberries, 1 TP-Link router, and 7 Raspberry racks. Conversely, the fog computing environment would incur equipment acquisition expenses of approximately US$2,261.98, comprising 4 Dell PowerEdge R240 servers, 1 Huacomm 5-Port switch, and 1 rack. The yearly expenses for edge and fog computing, considering energy consumption, are illustrated in Figure 28.

Utilizing the reciprocal values provided in Table 16 as parameters for the first two levels, we proceed with the performability analysis. Considering the edge system, we assume an arrival rate of $\lambda = 900\,tph$. Similarly, for the fog system, the corresponding rate is $\lambda = 900\,tph$. However, in this particular scenario, we consider the presence of

Figure 28 – Effective Expenses Scenario #01 - Edge and Fog.



Source: Elaborated by the author.

ten physical machines in the alarm monitoring center. Consequently, the arrival rate for the fog system is multiplied by 10, while the service rate (μ) remains unchanged. The outcomes of this scenario are presented in Table 25.

Table 25 – Results of a hypothetical company facial recognition system.

Node	**Metric**	**Performability**
Edge	Utilization (%)	86.25
	Response Time (ms)	2979.80
	Waiting Time (ms)	3.56×10^{-4}
	Discard (request/hour)	0.5250
Fog	Utilization (%)	65.37
	Response Time (ms)	1223.04
	Waiting Time (ms)	8.10×10^{-6}
	Discard (request/hour)	4.97×10^{-5}

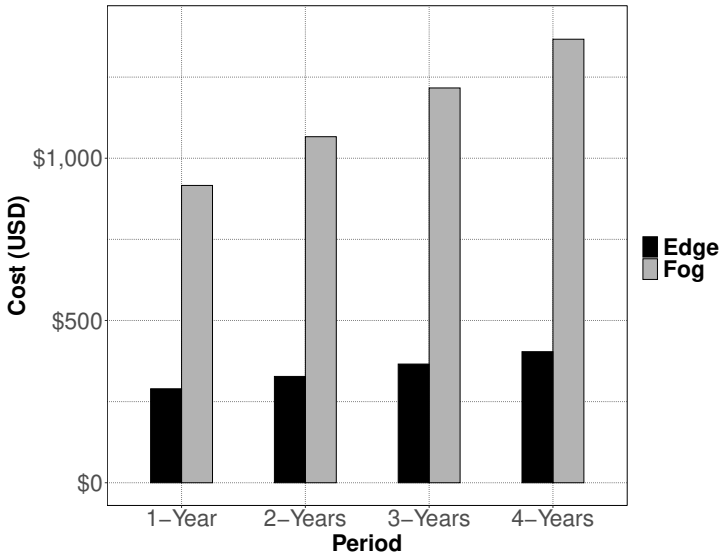
Source: Elaborated by the author.

As depicted in Figure 28, the effective expenses of the fog environment progressively surpass those of the edge environment over the years. However, the performance of the edge environment significantly lags behind that of the fog environment, as indicated in Table 25. Metrics such as utilization, response time, waiting time, and discard rate exhibit higher values in the edge environment compared to the fog environment. This disparity poses a notable challenge, as higher utilization is directly associated with an increased likelihood of elevated discard rates. To address this issue, one possible solution is to introduce redundancy or augment the number of physical machines in the edge environment.

However, both options would also escalate the effective expenses of the edge environment. Consequently, it is advisable to opt for the fog environment for deploying the surveillance system in this particular scenario.

Scenario #02: given the intention to deploy a surveillance system in a farm storage facility for intruder detection, the analyst determined that ten cameras would suffice for comprehensive area coverage. It is worth emphasizing that running this intruder detector in a cloud environment can give rise to significant challenges, particularly in rural areas characterized by potentially higher latency in Internet connections compared to urban areas.

Figure 29 – Effective Cost Scenario #02 - Edge and Fog.



Source: Elaborated by the author.

Considering the edge and fog environments, with the components listed in Table 23 and their respective values, the edge environment has a capacity of running four applications per physical node ($L = 4$), while the fog environment can accommodate 32 applications per physical node ($L = 32$). Consequently, to support ten cameras, the edge environment would require five physical nodes ($M = 3$), whereas the fog environment would only need one physical node ($M = 1$). The total equipment acquisition cost for the edge computing environment is estimated to be around US$250.00, consisting of three Raspberries, one TP-Link router, and one Raspberry rack. Conversely, the total equipment acquisition cost for the fog computing environment is estimated to be around US$766.00, comprising one Dell PowerEdge R240 server, one Huacomm 5-Port switch, and one rack. The yearly cost, considering energy consumption, edge, and fog computing, is presented in Figure 29. Furthermore, Table 26 displays the results obtained from the performability models, utilizing the inverse values from Table 16 as inputs for the first two levels, and $\lambda = 900\,tph$ and $\mu = 42.01\,tph$ for the edge system, and $\lambda = 900\,tph$ and $\mu = 270.06\,tph$ for the fog system

in the third level.

Table 26 – Results of the farm's intruder detection system.

Node	Metric	Performability
Edge	Utilization (%)	59.91
	Response Time (ms)	2856.8
	Waiting Time (ms)	246.6
	Discard (request/hour)	0.0745
Fog	Utilization (%)	10.43
	Response Time (ms)	1328.84
	Waiting Time (ms)	1.15×10^{-17}
	Discard (request/hour)	5.34×10^{-4}

Source: Elaborated by the author.

Analyzing Table 26, we observe that some performability metrics of the fog environment outperform those of the edge environment, suggesting that the fog environment would be the preferred choice for a surveillance system, given its higher performance and availability. However, it is crucial to consider the significant difference in the effective yearly expenses between the fog and edge environments, with the fog costs exceeding double those of the edge environment. Even with the utilization of multiple physical edge nodes, the edge environment offers lower costs. Furthermore, the edge environment already demonstrates adequate performability metrics for a small-scale surveillance system, while the utilization of the fog node remains low, indicating an underutilization of computational resources. Hence, in this scenario, the edge environment is the recommended choice.

6.2.3 Insights from the case study

In this case study, it was evident that availability plays a critical role in determining the system's overall performance. Higher availability directly translates to better performance, as system downtime leads to the rejection of incoming requests. This correlation between availability and performance highlights the importance of ensuring a reliable and accessible system for face recognition applications.

Moreover, the study emphasized the significance of performability analysis, which considers both performance and availability metrics. By incorporating failures and repairs into the analysis, performability models provide a comprehensive understanding of how system behavior evolves over time. This holistic approach enables a more accurate estimation of performance metrics under real-world scenarios, accounting for both expected system operation and unexpected events.

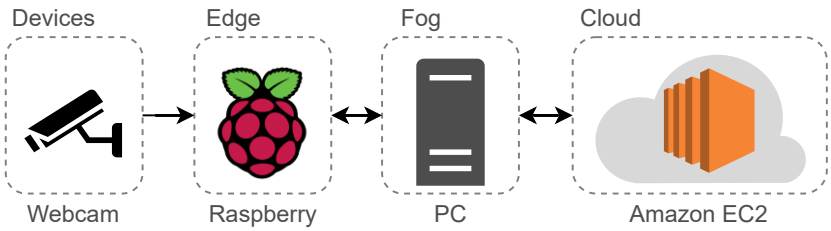
6.3 HIERARCHICAL MODELS

In these case studies, we established an integrated edge-fog-cloud continuum environment to assess the hierarchical availability model. Specifically, we focused on evaluating the availability of a smart traffic management system. This service was motivated by the increasing prevalence of smart city initiatives worldwide, where efficient traffic management plays a vital role. By examining the availability of such systems within the edge-fog-cloud architecture, we aimed to gain insights into their availability characteristics.

6.3.1 Case Study I - Availability model validation

In this case study, we present the experimental setup conducted to assess the representation of our hierarchical availability model in a real-world environment. Our experimental environment consisted of a webcam, a Raspberry Pi (edge node), and a personal computer (fog node). The webcam, an IP Camera D-LINK H.264 DCS-931L, captured video at a rate of 30 frames per second and a resolution of 720×480. The captured video was transmitted to the edge node for initial preprocessing, and subsequently relayed to the fog node, which further processed the data before forwarding it to the cloud infrastructure. The edge node, a Raspberry Pi, was equipped with a 1 GHz single-core processor, two threads, and 512 MB of RAM. The fog node utilized XenServer 7.2.0 as the hypervisor platform, boasting 8 GB of RAM, 1 TB of storage, and an Intel Core i5 3.20 GHz quad-core CPU with eight threads. Both the fog and edge nodes employed a Python script for vehicle recognition, with the fog node additionally running an RTMP server (e.g., Nginx) to receive the video stream from the edge node. To simulate our cloud environment, we utilized a virtual machine (t4g.nano) on Amazon EC2, featuring 2 vCPUs and 0.5 GiB of RAM. The overall experimental setup is illustrated in Figure 30, depicting the foundational configuration of our testbed environment.

Figure 30 – Testbed environment.



Source: Elaborated by the author.

We utilized the AlexNet model developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton (KRIZHEVSKY; SUTSKEVER; HINTON, 2017) for our case studies. The adoption of pre-trained models involved the use of the PyTorch framework to facilitate

the process. Once the experimental environment was established, we conducted multiple case studies by introducing failures through a Python script.

For our availability evaluation, we adopted the failure and repair rates mentioned in prior works such as (NGUYEN; MIN; CHOI, 2020; LISBOA et al., 2018; DANTAS et al., 2016; PEREIRA et al., 2021). These rates were derived from the assumption that the time to failure and repair follow an exponential distribution, with the failure rate being the inverse of the mean time to failure (MTTF), and the repair rate being the inverse of the mean time to repair (MTTR). The specific values of MTTF and MTTR utilized in our study are provided in Table 27.

To assess the availability of systems, fault injection is a commonly employed technique. This approach is particularly useful as faults are often challenging to predict or may take an extended period to occur (RUIZ et al., 2004; SOUZA et al., 2013; BRILHANTE et al., 2014; JAMMAL et al., 2018). By employing fault injection, we can accelerate, control, and monitor experiments during fault events (AGARWAL et al., 2020). The introduction of faults causes modifications within the system, leading to erroneous states and potentially propagating issues that impact the availability of provided services.

Table 27 – MTTF and MTTR from literature.

Env.	Components	MTTF (h)	MTTR (h)
Edge	Camera	8760	1.67
	Raspberry	4767.8	3.48
	OS	2880	1
	Python App	217.8	0.46
	Nginx App	217.8	0.46
Fog	Gateway	70080	1.67
	Hardware	8760	1.67
	OS	2880	1
	Cont. Management	2880	1
	Python App	217.8	0.46
	Nginx App	217.8	0.46
Cloud	Gateway	70080	1.67
	Storage	43800	24
	Hardware	70080	1.67
	OS	2880	1
	Cont. Management	2880	1
	Python App	217.8	0.46
	Nginx App	217.8	0.46

Source: Elaborated by the author based on Dantas et al. (2016).

For the establishment of a baseline infrastructure for a smart traffic management

system, we adopted a minimalistic approach by utilizing nodes with the bare minimum components necessary to provide the desired service. The edge node's essential components encompass the camera, Raspberry Pi, operating system, a Python application for vehicle recognition, and an Nginx RTMP server responsible for receiving the camera's video streaming. Distinct from the edge node, the fog node includes the addition of a gateway and utilizes a personal computer as the hardware platform for the remaining components. Additionally, the cloud node consists of storage, a gateway, and a virtual machine hosted in Amazon EC2.

To ensure the validity of our models, it was imperative to subject the system to stress and accelerate the mean time to failure (MTTF) of its components. To facilitate this, we developed a fault injector capable of introducing faults into each component of our baseline infrastructure. Once again, we made the assumption that the time to failure and repair for each component follows an exponential distribution, allowing us to determine the failure and repair rates as the reciprocals of the MTTF and mean time to repair (MTTR), respectively (NGUYEN; MIN; CHOI, 2020; LISBOA et al., 2018; DANTAS et al., 2016). By accelerating the MTTF, we increased its rate by a factor of 100 compared to the literature values, effectively compressing one year into a mere 87.6 hours. The specific values employed in the fault injector are outlined in Table 28. It is important to note that the accelerated MTTF significantly reduces the system's availability.

By employing the abstraction approach utilized in our Markov chain model, we are able to identify two distinct components within the edge and fog computing environment, namely the application and the physical node. The abstraction process was conducted using the Reliability Block Diagram (RBD) model, leveraging the Mercury tool (SILVA et al., 2015; PINHEIRO et al., 2021). This approach effectively consolidates multiple components into a single representative entity, simplifying the system representation and enabling the application of analytical techniques based on Continuous-Time Markov Chain (CTMC) analysis. To this end, the values provided in Table 28 were employed to derive the corresponding values for the abstraction components, which can then be utilized as inputs in the model. These mapped values for the abstraction components are presented in Table 29.

By combining the Mean Time to Failure (MTTF) and Mean Time to Repair (MTTR) values of serial components, we create an aggregated equivalent component that encapsulates the unified representation of availability characteristics. This condensation procedure significantly simplifies the modeling of individual components, enabling a more streamlined analysis of the entire system. By considering the combined availability behavior of the condensed component, a comprehensive understanding of the system's overall availability can be achieved.

This abstraction technique proves particularly advantageous when employing CTMC analysis, as it allows for the system to be represented as a set of interconnected states and transitions. The consolidation of multiple components into a single entity reduces

Table 28 – MTTF and MTTR in the fault injector.

Env.	Components	MTTF (h)	MTTR (h)
Edge	Camera	87.6	1.67
	Raspberry	47.6	3.48
	OS	28.8	1
	Python App	2.17	0.46
	Nginx App	2.17	0.46
Fog	Gateway	700.8	1.67
	Hardware	87.6	1.67
	OS	28.8	1
	Cont. Management	28.8	1
	Python App	2.17	0.46
	Nginx App	2.17	0.46
Cloud	Gateway	700.8	1.67
	Storage	438	24
	Hardware	700.8	1.67
	OS	28.8	1
	Cont. Management	28.8	1
	Python App	2.17	0.46
	Nginx App	2.17	0.46

Source: Elaborated by the author.

the number of states within the CTMC model, thereby facilitating a more manageable analysis process. As a result, analytical solutions, such as steady-state probabilities and performance measures, can be derived, providing valuable insights into the system's availability.

The deployment of our fault-injection environment is illustrated in Figure 31. To generate the workload, we utilized a webcam to capture images, which were subsequently transmitted to our designated nodes, as previously described. Our fault injector, developed in Python, maintains a list of all components that will be subject to failure and monitoring. For each component, a dedicated thread is created within the fault injector, responsible for injecting the fault. The time to failure for each component follows an exponential distribution, as specified in Table 28. Accordingly, the thread generates a random number based on this distribution and awaits the appropriate moment to inject the fault. Additionally, we have another thread that continuously monitors the system's availability. The monitoring thread records the status of the system as either U (up) or D (down) in a text file, with sample intervals set at 10-second intervals.

Utilizing the depicted environment shown in Figure 31, we conducted an experiment on the edge-fog-cloud system. The experiment extended for a duration of 96 hours, generating

Table 29 – MTTF and MTTR for model.

Env.	Components	MTTF (h)	MTTR (h)
Edge	Camera	87.60	1.67
	Physical Node	17.95	1.98
	Application	1.08	0.50
Fog	Gateway	700.80	1.67
	Physical Node	21.67	1.18
	Application	1.08	0.50
Cloud	Gateway	700.80	1.67
	Storage	438	24
	Physical Node	27.66	1.02
	Application	1.08	0.50

Source: Elaborated by the author.

a text file containing multiple occurrences of system states denoted by U (up) and D (down). By quantifying the number of U and D instances, we were able to compute the time to failure (TTF) and time to repair (TTR). For example, if we observed 50 consecutive U states followed by a D state, the cumulative uptime during that interval would be calculated as 50 multiplied by the 10-second sample collection interval, resulting in 500 seconds of uptime. The same principle applies to calculate the time to repair. Thus, by converting the U and D states into TTFs and TTRs, we obtained a total of approximately 113 sample points.

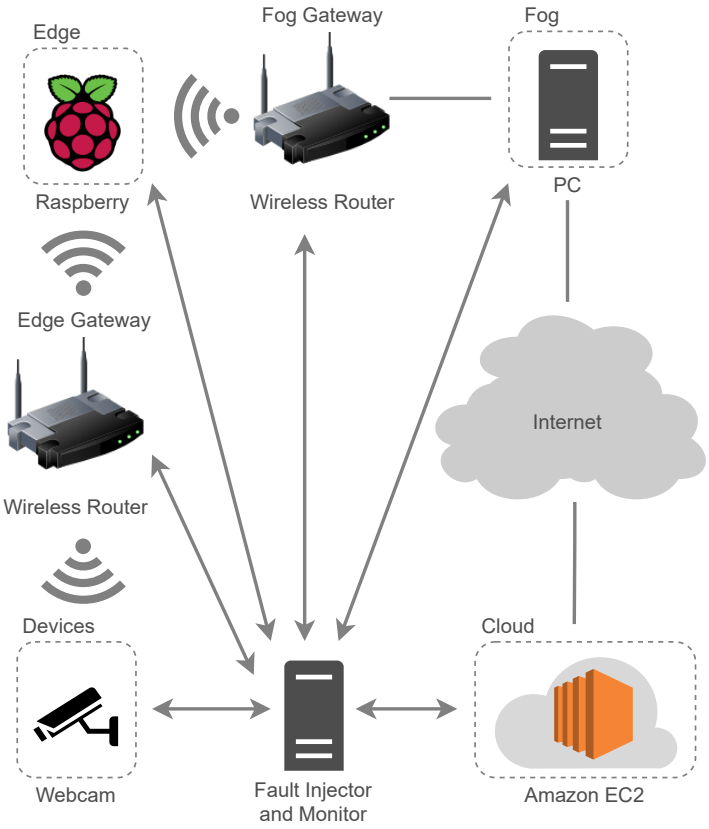
Upon completing the validation procedure outlined in Chapter 4, a comprehensive data analysis was conducted, revealing the absence of a theoretical distribution suitable for representing our sample distribution. Consequently, we resorted to employing the Bootstrap technique, a resampling method used to estimate statistics by sampling datasets with replacement (DEVORE, 2008), to calculate the availability confidence intervals for both the fog and edge nodes.

By utilizing a statistical computer package, we generated 1000 bootstrap samples for the system's availability, resulting in 1000 availability means for each node. Consequently, we derived the confidence interval by selecting the 25th smallest and 25th largest values from these 1000 bootstrap samples. This signifies that, in the long run, if the experiment were repeated numerous times, the availability of the baseline system would fall within these calculated confidence intervals approximately 95% of the time.

Using the analytical and hierarchical models described in Section 5.3, along with the component values outlined in Table 29, we obtained the availability values for the baseline system. The point estimations of availability, along with their corresponding confidence intervals, are summarized in Table 30, alongside the model results.

The presence of confidence intervals containing the estimation of the hierarchical model

Figure 31 – Fault-injection environment.



Source: Elaborated by the author.

Table 30 – Comparison of availability results.

Model Result	System Measurement	CI 95% of the system
0.2434	0.2312	$0.1838 < \theta < 0.2589$

Source: Elaborated by the author.

for the edge-fog-cloud system implies that we cannot assert with certainty that our model does not represent the real environment. Nevertheless, based on the observations, it is reasonable to assume that our model serves as a reliable representation of the edge-fog-cloud environment, thus confirming its validation. Consequently, we can now utilize the hierarchical model to calculate the availability of the edge-fog-cloud system. This enables us to formulate strategies aimed at attaining elevated levels of availability across various scenarios.

6.3.2 Case Study II - Availability evaluation

In this case study, we employ the proposed hierarchical model to estimate the availability of our baseline environment, specifically for a smart traffic management system as described previously. To facilitate this evaluation, we utilize analytical models at the initial

level of the hierarchy due to their inherent scalability advantages over alternative modeling techniques such as Reliability Block Diagrams and Stochastic Petri Nets, as previously highlighted (JAIN, 1990). By leveraging analytical models at this foundational level, we can efficiently assess numerous scenarios at a reduced computational cost. To support this evaluation, we refer to literature values of Mean Time to Failure (MTTF) and Mean Time to Repair (MTTR) for the components within the edge-fog-cloud environment, as outlined in Table 27. These literature values serve as inputs within the Mercury tool (SILVA et al., 2015) to derive the corresponding abstraction values, as presented in Table 31.

Table 31 – The abstraction MTTF and MTTR.

Env.	Components	MTTF (h)	MTTR (h)
Edge	Camera	8760	1.67
	Physical Node	1795.45	1.98
	Application	108.89	0.5
Fog	Gateway	70080	1.67
	Physical Node	2167.42	1.18
	Application	108.89	0.5
Cloud	Gateway	70080	1.67
	Storage	43800	24
	Physical Node	2766.31	1.02
	Application	108.89	0.5

Source: Elaborated by the author.

Considering the capabilities of our single edge, fog, and cloud physical nodes, which can respectively run two, eight, and eight applications, we set the value of L as two for the edge environment and eight for the fog and cloud environments in our model. In this initial evaluation, we do not incorporate redundancy, resulting in $M = 1$ for all the environments when applying Equation 5.12.

Utilizing our hierarchical model for evaluation, we obtained a total downtime of 134.05 hours for our baseline edge-fog-cloud system. This implies that over the course of a year, the system would experience approximately 135 hours of downtime, resulting in an uptime of 8631.76 hours. The calculated availability corresponds to 1.81 nines, as per Table 1. This categorizes our baseline system as falling between the unmanaged and managed levels. In practical scenarios, these results would be deemed unacceptable, particularly for a critical application like a smart traffic management system, where the safety of many lives is at stake. The summary of results for this evaluation is presented in Table 32.

In order to enhance the availability of the system, it is crucial to investigate the critical components, those that exert the greatest influence on the system's availability. Conducting this investigation is essential to avoid indiscriminate redundancy implementation, which can be costly. Without proper analysis, redundancy might be applied to

Table 32 – Results of the baseline environment.

Metric	**Values**
Availability	0.9847
Unavailability	0.0153
# of 9's	1.8155
Downtime (h)	134.05
Uptime (h)	8631.76

Source: Elaborated by the author.

non-critical components, leading to unnecessary expenses. Therefore, to enhance system availability, a sensitivity analysis is performed, utilizing the percentage difference technique. This method allows us to identify the components that have the most substantial impact on system availability.

The Sensitivity Index (SI), as given in Equation 2.28, is employed to establish a ranking of the components based on their impact on system availability. The sensitivity analysis was carried out using the values provided in Table 33, which includes the maximum and minimum values for each system component. These extreme values were calculated based on the baseline values specified in Table 31. Specifically, the maximum values are set to be 50% higher than the baseline values, while the minimum values are set to be 50% lower than the baseline values (MATOS et al., 2015). This analysis aids in identifying the components with the greatest influence on system availability.

By utilizing the Mercury tool (SILVA et al., 2015), with the values provided in Table 28, we are able to calculate the Sensitivity Index (SI). This index assists us in determining the criticality of components concerning the availability of our system in the edge-fog-cloud environment. The rank of component criticality based on the SI is presented in Table 34.

Based on our sensitivity analysis, the parameters that have the greatest impact on the availability of our edge-fog-cloud environment are the MTTF (Mean Time to Failure) and the MTTR (Mean Time to Repair) of the applications. This indicates that in order to achieve higher availability values, it is necessary to focus on improving these parameters within our system. Figures 32, 33, and 34 illustrate the behavior of system availability as we vary the MTTF and MTTR of components in the edge-fog-cloud environment. While some components have a minor impact on availability, there are others where the MTTF and MTTR play a critical role. Enhancing these critical components is essential for increasing system availability.

When considering methods for improving system availability, one commonly employed approach is redundancy. Redundancy can be classified into three types: cold, warm, and hot standby. The choice of redundancy type depends on the criticality of the process and the consequences of equipment failures (MELO et al., 2018). Cold-standby redundancy is

Table 33 – Min. and Max. values for the Sensitivity Analysis.

Env.	Parameter	Min. (h)	Max. (h)
Edge	Camera's MTTF	4380.0	13140.0
	Camera's MTTR	0.83	2.50
	Phy. Node's MTTF	897.72	2693.17
	Phy. Node's MTTR	0.99	2.97
	Application's MTTF	54.44	163.33
	Application's MTTR	0.25	0.75
Fog	Gateway's MTTF	35040.0	105120.0
	Gateway's MTTR	0.83	2.50
	Phy. Node's MTTF	1083.71	3251.13
	Phy. Node's MTTR	0.59	1.77
	Application's MTTF	54.44	163.33
	Application's MTTR	0.25	0.75
Cloud	Gateway's MTTF	35040.0	105120.0
	Gateway's MTTR	0.83	2.50
	Storage's MTTF	21900.0	65700.0
	Storage's MTTR	12.0	36.0
	Phy. Node's MTTF	1383.15	4149.46
	Phy. Node's MTTR	0.51	1.53
	Application's MTTF	54.44	163.33
	Application's MTTR	0.25	0.75

Source: Elaborated by the author.

suitable for non-critical services where human intervention is acceptable, and time is not of utmost importance. Warm-standby redundancy is implemented when timely response to failures is important but not critical, allowing for temporary outages. In contrast, hot-standby redundancy offers instantaneous process correction in the event of a failure, prioritizing time and security, and ensuring uninterrupted operation.

Considering that our environment is dedicated to smart traffic management, where the lives of many depend on its continuous functioning, it is imperative that the system remains operational at all times. Therefore, we have opted for hot-standby redundancy. By implementing hot-standby redundancy on the edge, fog, and cloud physical servers, we were able to significantly reduce the system's downtime to 7.36 hours. The achievement of 3.07 nines in the system's availability rating indicates that it has transitioned from an unmanaged state to a well-managed one. Thus, the addition of a new physical node to each paradigm has transformed the system into a well-managed one. The results of this improvement are summarized in Table 35.

Table 34 – Sensitivity Index ranking.

Parameter	SI
(Edge) Application's MTTF	5.82×10^{-3}
(Cloud) Application's MTTF	5.81×10^{-3}
(Fog) Application's MTTF	5.80×10^{-3}
(Cloud) Application's MTTR	4.05×10^{-3}
(Fog) Application's MTTR	4.04×10^{-3}
(Edge) Application's MTTR	4.03×10^{-3}
(Edge) Phy. Node's MTTF	1.40×10^{-3}
(Edge) Phy. Node's MTTR	9.78×10^{-4}
(Cloud) Storage's MTTF	7.29×10^{-4}
(Fog) Phy. Node's MTTF	7.25×10^{-4}
(Cloud) Storage's MTTR	5.47×10^{-4}
(Fog) Phy. Node's MTTR	4.83×10^{-4}
(Cloud) Phy. Node's MTTF	4.71×10^{-4}
(Cloud) Phy. Node's MTTR	3.68×10^{-4}
(Edge) Camera's MTTF	2.43×10^{-4}
(Edge) Camera's MTTR	1.90×10^{-4}
(Cloud) Gateway's MTTF	3.05×10^{-5}
(Fog) Gateway's MTTF	3.04×10^{-5}
(Fog) Gateway's MTTR	2.38×10^{-5}
(Cloud) Gateway's MTTR	2.37×10^{-5}

Source: Elaborated by the author.

Table 35 – Results of the hot-standby redundancy.

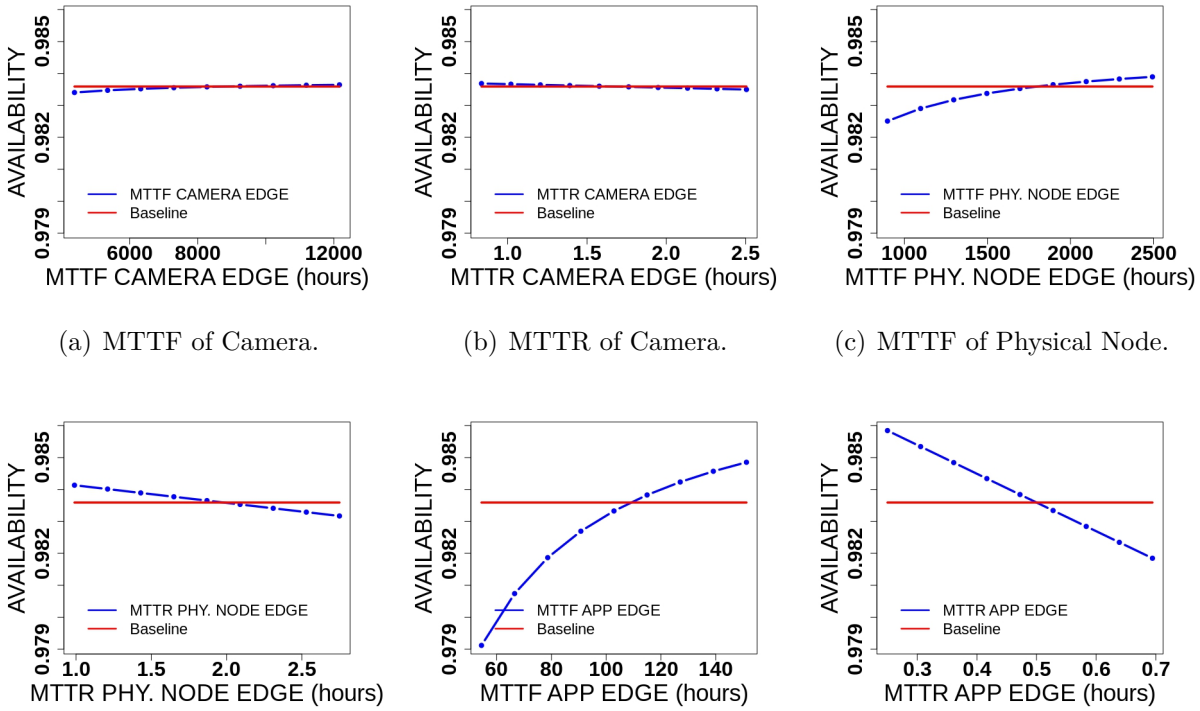
Metric	Values
Availability	0.9991
Unavailability	0.0009
# of 9's	3.0755
Downtime (h)	7.36
Uptime (h)	8758.44

Source: Elaborated by the author.

6.3.3 Insights from the case study

Through the case study conducted on availability using a hierarchical model in the context of smart traffic management, we were able to note that the baseline edge-fog-cloud system exhibited a moderate level of availability, falling between the unmanaged and managed categories. This indicated the need for improvements to ensure a more robust

Figure 32 – Sensitivity analysis of edge computing paradigm.



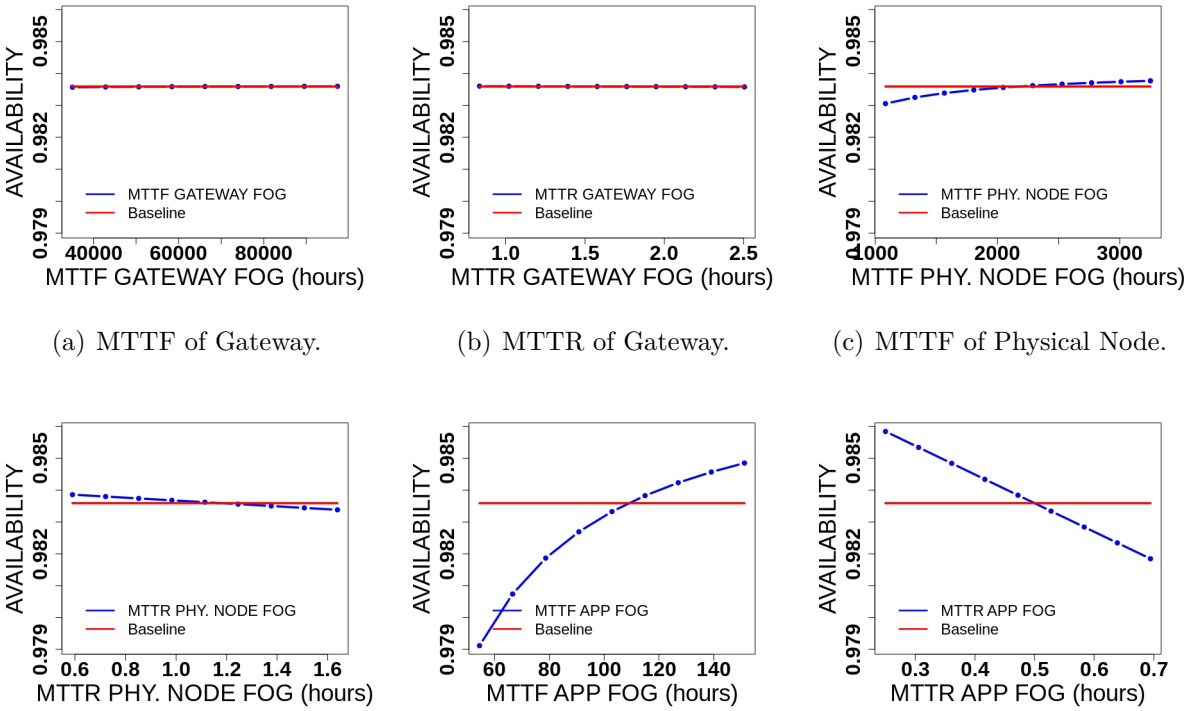
(a) MTTF of Camera. (b) MTTR of Camera. (c) MTTF of Physical Node.

(d) MTTR of Physical Node. (e) MTTF of Application. (f) MTTR of Application.

Source: Elaborated by the author.

and reliable system. Sensitivity analysis allowed for the identification of the most critical components influencing the system's availability, highlighting the importance of focusing on enhancing the MTTF and MTTR of the applications. By implementing hot-standby redundancy, significant improvements were achieved, reducing the system's downtime and elevating its availability to a well-managed level. This study emphasized the significance of redundancy as a mechanism for enhancing system availability and showcased the potential for transforming an initially unmanaged system into a highly reliable and resilient one.

Figure 33 – Sensitivity analysis of fog computing paradigm.

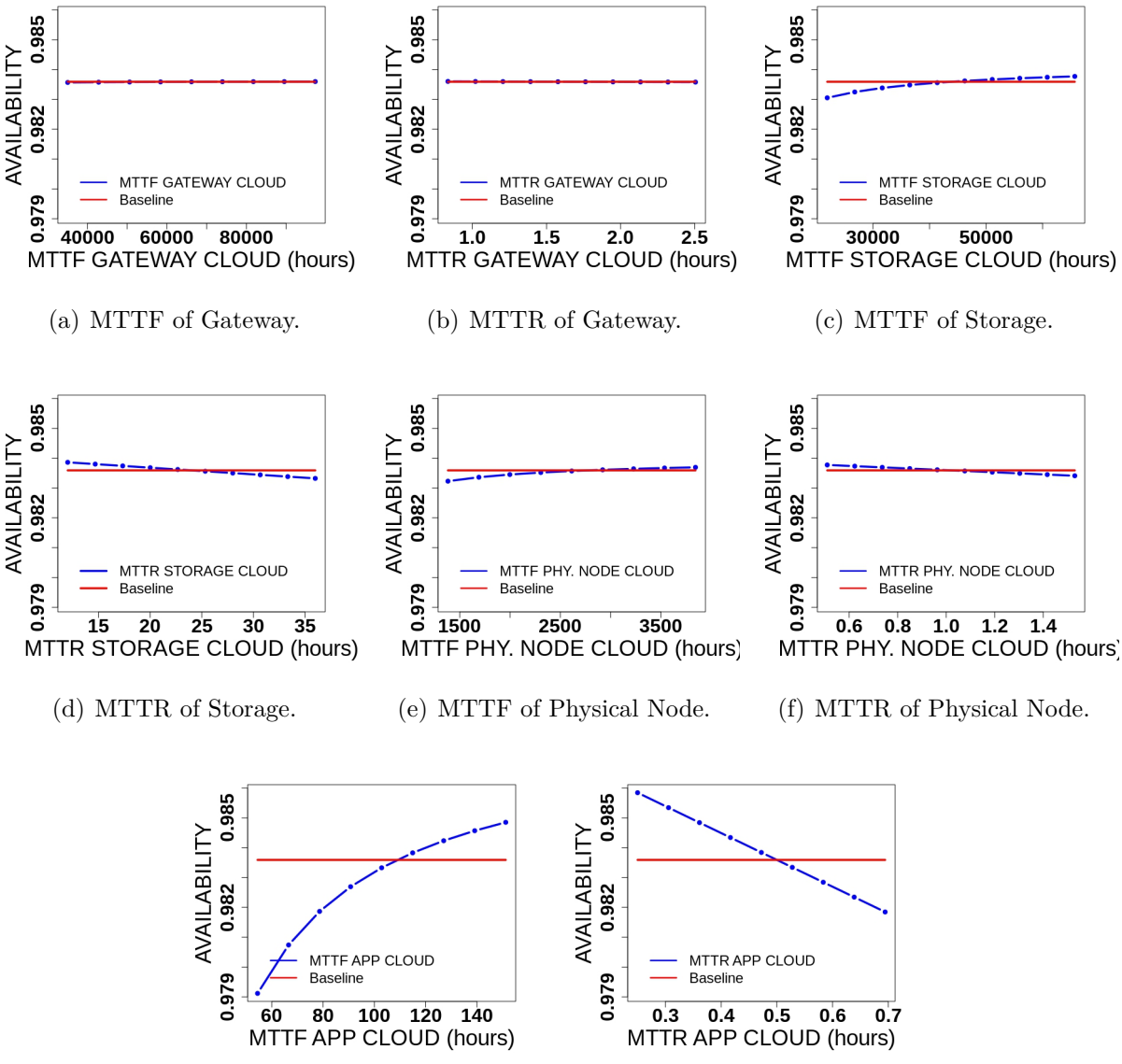


(a) MTTF of Gateway. (b) MTTR of Gateway. (c) MTTF of Physical Node.

(d) MTTR of Physical Node. (e) MTTF of Application. (f) MTTR of Application.

Source: Elaborated by the author.

Figure 34 – Sensitivity analysis of cloud computing paradigm.



(a) MTTF of Gateway. (b) MTTR of Gateway. (c) MTTF of Storage.

(d) MTTR of Storage. (e) MTTF of Physical Node. (f) MTTR of Physical Node.

(g) MTTF of Application. (h) MTTR of Application.

Source: Elaborated by the author.

6.4 HIERARCHICAL AVAILABILITY MODELS WITH FAULT COVERAGE PROBABILITY

In this section, we extend our evaluation to include a hierarchical model with fault coverage probability. We apply this model to analyze a smart building monitoring service, expanding our scope of study. As with previous case studies, we begin by validating the model's accuracy before proceeding to utilize it for service capacity planning purposes. By incorporating fault coverage probability into our hierarchical model, we aim to assess the system's reliability and capacity to effectively monitor and manage a smart building environment.

6.4.1 Case study I - Availability model validation

To ensure the reliability and accuracy of our proposed model, it is essential to validate its performance against real-world data. In this section, we compare the results obtained from our model with the data collected from the baseline service, as described in Section 4. By evaluating the consistency between the model's outputs and the experimental environment, we can assess the model's ability to accurately represent the system's behavior. To perform this validation, we consider the failure and repair rates described in (NGUYEN; MIN; CHOI, 2020; LISBOA et al., 2018; DANTAS et al., 2016), assuming an exponential distribution for these rates. Table 36 presents the specific values of Mean Time to Failure (MTTF) and Mean Time to Repair (MTTR) that were utilized for validating our proposed model.

Table 36 – MTTF and MTTR from literature.

Env.	Components	MTTF (h)	MTTR (h)
Sensors	Hardware	4767.8	3.48
Edge	Raspberry	4767.8	3.48
	OS	2880	1
	Python App	217.8	0.46
Fog	Hardware	8760	1.67
	OS	2880	1
	Python App	217.8	0.46

Source: Elaborated by the author based on Dantas et al. (2016).

During the validation phase of our proposal, it was necessary to accelerate the occurrence of failures in the system in order to accurately measure the availability. To achieve this, we employed a fault-injector mechanism (SOUZA et al., 2013; BRILHANTE et al., 2014; JAMMAL et al., 2018), which allowed us to simulate and control the occurrence of faults in a controlled manner.

For the purpose of validation, we designed a baseline service with a minimal set of components. The edge node consisted of a Raspberry Pi running the Raspberry Pi OS, along with an application responsible for analyzing sensor data. Similarly, the fog node was comprised of a personal computer running a Linux operating system, accompanied by an application specific to its functionality.

In order to validate our model effectively, we accelerated the Mean Time to Failure (MTTF) by a factor of 100 compared to the values reported in the literature. Consequently, within our experiment, the duration of one year corresponded to approximately 87.6 hours. Table 37 provides a comprehensive overview of the values employed during the validation process.

Table 37 – MTTF and MTTR in the fault injector.

Env.	Components	MTTF (h)	MTTR (h)
Sensors	Hardware	47.6	3.48
Edge	Raspberry	47.6	3.48
	OS	28.8	1
	Python App	2.17	0.46
Fog	Hardware	87.6	1.67
	OS	28.8	1
	Python App	2.17	0.46

Source: Elaborated by the author.

The values corresponding to each component are presented in Table 37. The abstraction form of the values is displayed in Table 38 are obtained by the abstraction process conducted using the Reliability Block Diagram (RBD) model, leveraging the Mercury tool (SILVA et al., 2015; PINHEIRO et al., 2021)

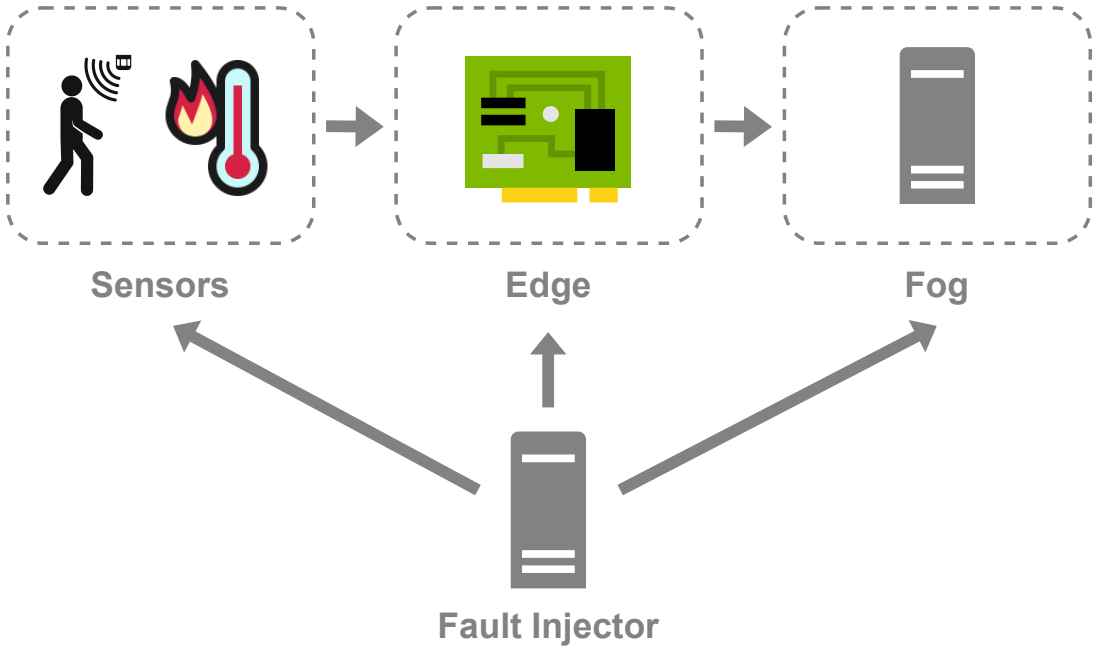
Table 38 – MTTF and MTTR for the model.

Env.	Components	MTTF (h)	MTTR (h)
Sensors	Hardware	47.6	3.48
Edge	Physical Node	17.95	1.98
	Application	2.17	0.46
Fog	Physical Node	21.67	1.18
	Application	2.17	0.46

Source: Elaborated by the author.

The validation environment is illustrated in Figure 35. To simulate failures, our fault injector generates random numbers following an exponential distribution, utilizing the values provided in Table 37. The fault injector then injects failures based on these generated numbers. Additionally, the fault injector continuously monitors the system's status,

Figure 35 – Fault-injection environment.



Source: Elaborated by the author.

checking if it is operational or experiencing downtime. The monitoring process occurs at intervals of 10 seconds.

We conducted a thorough evaluation over a period of 100 hours, collecting multiple samples that indicated the system's operational and downtime states. These samples enabled us to estimate the time to failure (TTF) and time to repair (TTR) for our baseline service.

Upon analyzing the TTF and TTR data, we observed that it does not follow a normal distribution, and the sample size is relatively small. Consequently, we employed the Bootstrap technique to calculate the confidence interval for availability (DEVORE, 2008). The results obtained through the Bootstrap method are presented in Table 39.

Table 39 – Comparison of availability results.

Model	**Real System**	**CI 95% of the system**
0.2169	0.2014	$0.1802 < \theta < 0.2226$

Source: Elaborated by the author.

The confidence intervals presented in Table 39 encompass the availability estimations derived from the hierarchical model. This indicates that our model demonstrates a comparable behavior to the baseline service. With this validation, we can proceed to design infrastructures that aim to achieve higher levels of availability in our smart building monitoring system.

6.4.2 Case study II - Availability evaluation

We will now utilize our hierarchical model to assess the availability of our experimental system, as outlined in Section 4. To accomplish this, we employed the Mercury tool (SILVA et al., 2015; PINHEIRO et al., 2021) to calculate the corresponding abstraction values based on the literature. The abstraction values are presented in Table 40.

Table 40 – The abstraction MTTF and MTTR.

Env.	Components	MTTF (h)	MTTR (h)
Sensors	Hardware	4767.8	3.48
Edge	Physical Node	1795.45	1.98
	Application	217.8	0.46
Fog	Physical Node	2167.42	1.18
	Application	217.8	0.46

Source: Elaborated by the author.

Assuming that L is equal to one for both the edge and fog components, and considering the absence of redundancy, the value of M in Equation 5.24 is also set to one. By applying our hierarchical model to calculate the availability, we obtained a downtime of 70.86 hours in a year, resulting in 8689.13 hours of uptime. The corresponding number of nines is calculated to be 2.2120, indicating that our service falls between the unmanaged and managed categories. However, this level of availability is deemed inadequate for a smart building monitoring system, given the significant reliance on it to safeguard lives. A summary of the evaluation results is provided in Table 41.

Table 41 – Results of the baseline environment.

Metric	Values
Availability	0.9919
Unavailability	0.0080
# of 9's	2.2120
Downtime (h)	70.86
Uptime (h)	8689.13

Source: Elaborated by the author.

Hence, it is imperative to enhance the availability of our system to ensure its suitability for a smart building monitoring system. One well-established approach to achieve this goal is through redundancy. Redundancy encompasses three types: cold, warm, and hot standby, each catering to different service requirements (TRIVEDI et al., 1996). Cold-standby redundancy is typically employed for non-critical services, where human intervention is permissible and time is of lesser importance. Warm-standby redundancy, on the

other hand, serves to address failures promptly, although the service itself is not deemed essential. In contrast, hot-standby redundancy is employed when time and security take precedence, mandating uninterrupted system operation.

Considering the criticality of our scenario, where the system must remain operational due to its impact on lives, we have opted for hot-standby redundancy. By implementing this mechanism and solely applying hot-standby redundancy to the edge and fog nodes, we have successfully reduced the system's downtime to 39.46 hours. This corresponds to an availability achievement of 2.7973 nines, indicating that the system is managed. Consequently, by incorporating an additional physical node in each paradigm, we have effectively transformed an unmanaged system into a managed one. A comprehensive summary of the evaluation outcomes is presented in Table 42.

Table 42 – Results of the hot-standby redundancy.

Metric	**Values**
Availability	0.9954
Unavailability	0.0045
# of 9's	2.7973
Downtime (h)	39.46
Uptime (h)	8720.53

Source: Elaborated by the author.

6.4.3 Case study III - Fault coverage probability

In this particular case study, we examine the impact of fault coverage probability values on the availability of sensors. Our analysis focuses on a building floor scenario, where we explore variations in the number of sensors and the corresponding fault coverage probabilities. The number of sensors under consideration ranges from 5 to 50, while the fault coverage probability varies from 0.5 to 0.99. To provide a comprehensive overview of these evaluation parameters, Table 43 presents a summary of their values.

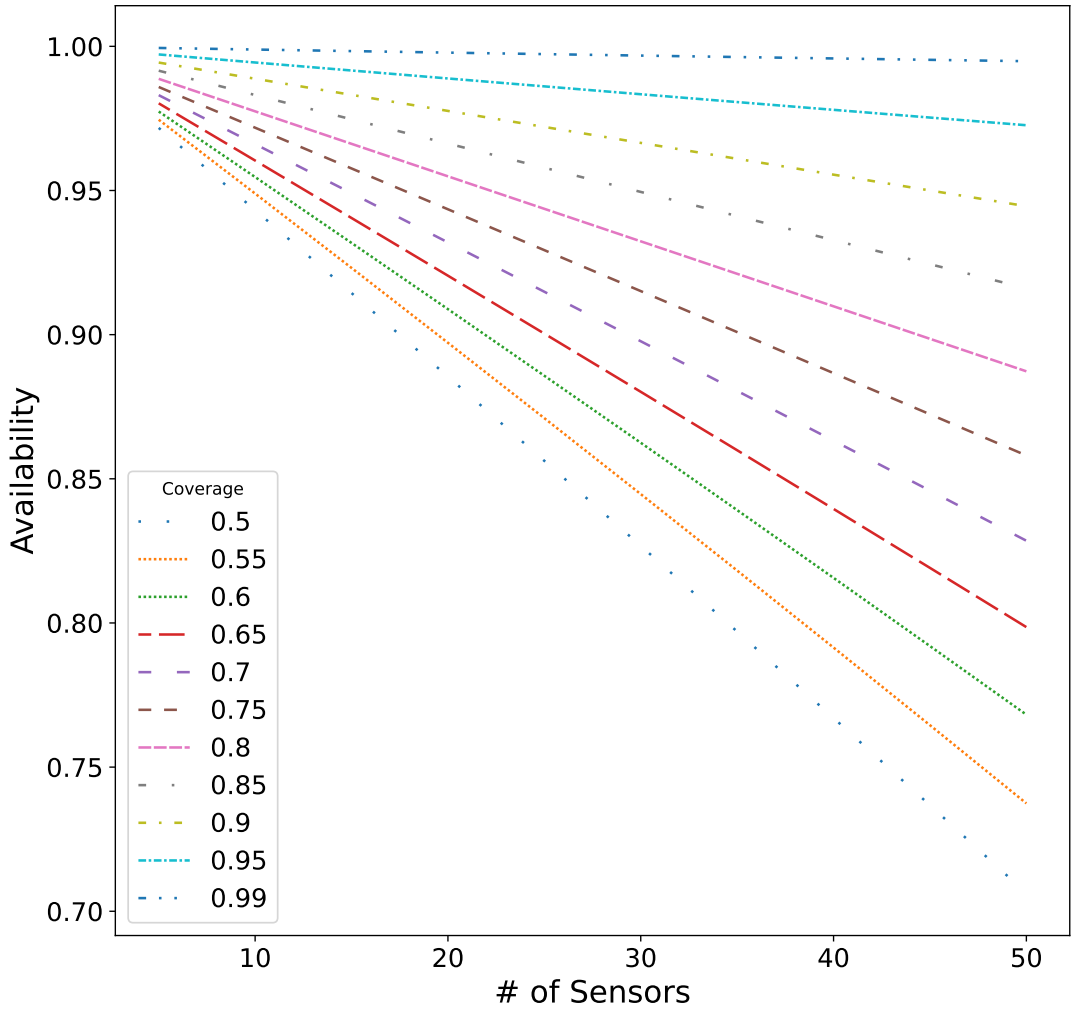
Table 43 – Evaluation parameters variation.

Metric	**Range**
Number of Sensors	from 5 to 50
Coverage Failure	from 0.5 to 0.99

Source: Elaborated by the author.

In this particular case study, we consider a scenario without redundancy, assuming that all sensors are actively utilized. The outcomes of our analysis, as illustrated in Figure 36, provide valuable insights into the relationship between fault coverage probabil-

Figure 36 – Fault Coverage Probability Analysis.



Source: Elaborated by the author.

ity and availability. The results clearly demonstrate that the fault coverage probability significantly impacts availability. Specifically, a lower coverage probability results in a prolonged replacement time for faulty sensors. Additionally, the number of sensors has a direct influence on availability. This correlation arises from the fact that as the number of sensors increases, the probability of encountering a faulty sensor also increases (TRIVEDI; SATHAYE; RAMANI,).

6.4.4 Insights from the case study

Through the case study on availability using a hierarchical model with coverage probability in the smart building context, we could observe that fault coverage probability plays a crucial role in determining the availability of sensors in a smart building. A higher fault coverage probability translates to a shorter time required to replace faulty sensors, thereby improving availability. On the other hand, a lower fault coverage probability results in lengthier replacement times, impacting availability. Additionally, the study highlighted

the significance of the number of sensors in relation to availability. Increasing the number of sensors increases the likelihood of encountering faulty sensors, which can negatively affect availability. These findings underscore the importance of optimizing fault coverage probability and carefully considering the number of sensors to ensure optimal availability in smart building systems.

6.5 FINAL REMARKS

In this chapter, we explored the use of analytical models to analyze and improve the availability of face recognition systems. By employing availability analytical models, we were able to estimate the downtime and uptime of the system, allowing us to assess its reliability and identify potential areas for improvement. Through this case study, we gained a deeper understanding of how availability affects the performance of face recognition systems and the importance of ensuring high availability in such critical applications.

Additionally, we extended our analysis to the performability of face recognition systems, considering both availability and performance metrics. This comprehensive evaluation allowed us to assess the system's overall capability to deliver reliable and efficient face recognition services. The results obtained from this study can guide system designers and operators in making informed decisions to enhance the performability of face recognition systems.

Moving beyond face recognition, we applied hierarchical models to evaluate the availability of systems in different contexts. In the smart traffic management context, the hierarchical models enabled us to estimate the availability of edge, fog, and cloud environments. By considering the impact of different components and their failure and repair rates, we were able to identify critical components that significantly influenced the availability of the system. Furthermore, the analysis revealed the potential benefits of implementing redundancy to improve availability in smart traffic management systems.

In the smart building context, we explored the use of hierarchical models with coverage probability to assess the availability of sensors. By varying the number of sensors and fault coverage probabilities, we investigated their impact on system availability. The results demonstrated that the fault coverage probability and the number of sensors are crucial factors in determining the availability of smart building systems. These findings highlight the need to optimize fault coverage probabilities and carefully consider the number of sensors to ensure optimal availability in smart building environments.

This chapter highlights the significance and efficacy of our analytical models in the assessment and evaluation of edge, fog, and cloud computing environments. The utilization of these models enables the development of improved infrastructures and cost-effective systems. By leveraging our models, stakeholders can make informed decisions and optimize their resources to create efficient and economically viable computing setups.

7 CONCLUSIONS AND FUTURE WORK

Numerous IT companies worldwide are actively developing Internet of Things (IoT) solutions for various applications (STATISTA, 2022). While cloud computing has been widely adopted by many companies, certain IoT solutions are sensitive to latency, requiring data processing within defined time intervals to avoid system failure. To address this latency issue, two emerging paradigms, namely edge and fog computing, have been introduced. Although these paradigms mitigate latency concerns, the cloud still offers superior availability and performance, emphasizing the need for further research on availability and performance in these contexts.

Furthermore, this thesis has yielded significant positive outcomes in the examined domains, with notable contributions in the form of models to enhance and evaluate availability and performability. The case studies conducted have demonstrated the effectiveness of our proposed models in evaluating diverse scenarios, enabling improved capacity planning and mitigating financial losses. These models can be employed to assess various infrastructures implemented within the edge-fog-cloud continuum, thereby directly impacting the quality of service (QoS) metrics and reducing the likelihood of QoS violations.

7.1 CONTRIBUTIONS

This thesis introduces analytical and hierarchical models for evaluating the availability and performability of applications within the edge-fog-cloud continuum. The proposed models aim to assist engineers in capacity planning and minimizing financial losses. The models utilize formalisms such as Markov chains, fault trees, and reliability block diagrams. Markov chains are employed to develop closed-form equations that assess the availability and performability of individual components. The fault tree formalism represents the interoperation between the edge, fog, and cloud paradigms.

Furthermore, an investigation was conducted to analyze how availability impacts the performance of edge, fog, and cloud environments, referred to as performability. Sensitivity analysis was employed to identify the most critical elements in the system. Performability plays a crucial role as it provides insight into the probability of performance metric behavior, considering system operation over an expected period while accounting for failures and repairs (MACIEL et al., 2012). Additionally, a capacity-oriented availability model was proposed to measure system availability based on average resource capacity. A cost evaluation was performed to compare the performance and availability costs among edge, fog, and cloud nodes. The study also presents key research studies and concepts concerning the availability and performability of cloud, fog, and edge computing environments.

In summary, this work makes several significant contributions:

- The development of analytical availability and performability models specifically designed for infrastructure planning in edge and fog environments. These models provide valuable insights for engineers in optimizing resource allocation and minimizing financial losses.

- The proposal of hierarchical models that enable the evaluation of the entire edge-fog-cloud continuum. These models offer a comprehensive approach to assess the availability and performability of complex computing environments, taking into account the interdependencies between different layers.

- The introduction of hierarchical models with fault coverage probability, which are particularly relevant for environments where failures may not be immediately detected. These models provide a more accurate representation of the system's behavior and enhance the evaluation of availability.

- An investigation into the impact of availability on the performance of edge and fog environments. This analysis sheds light on the relationship between these two crucial factors and offers insights for optimizing system design and operations.

Additionally, this work includes an analytical capacity-oriented availability model for efficient service provisioning in edge and fog nodes. A comprehensive cost evaluation is conducted, comparing the performance and availability costs across edge, fog, and cloud computing. Furthermore, a thorough comparison of existing works related to availability and performance assessment in edge, fog, and cloud computing is presented, highlighting the utilization of analytical and simulation-based models. Finally, the work provides a detailed differentiation between edge and fog computing, delving into the distinct characteristics and intertwined nature of these two paradigms.

7.2 SUMMARY OF RESULTS AND CONSTRAINTS

We have organized our case studies into four distinct sections to address various aspects of availability and performance evaluation. In Section 6.1, our focus was on analytical availability models applied to edge and fog nodes in a security face recognition system. Through these case studies, we successfully transformed the system from a well-managed state to a high-availability state. Moreover, we conducted a cost analysis that compared the cost-benefit aspects of using edge and fog nodes, providing valuable insights for infrastructure capacity planning.

In Section 6.2, our case studies explored the relationship between availability and system performance. By contrasting the results obtained from performability models with those from pure performance models, we demonstrated the importance of considering availability in performance evaluations. The outcomes of these case studies showcased

improvements in infrastructure performance by taking availability into account, thereby providing a more realistic assessment.

Section 6.3 introduced hierarchical models specifically designed to support a smart traffic management application. Apart from evaluating availability, we conducted a sensitivity analysis to identify the most critical components impacting availability. This analysis aids in more informed capacity planning decisions by highlighting the key elements influencing system availability.

In Section 6.4, we extended our hierarchical model to incorporate the evaluation of fault coverage probability in sensor networks. Through this expansion, we achieved significant improvements in system availability. These case studies demonstrated the effectiveness of our models in enhancing availability outcomes.

However, it is important to acknowledge the limitations of our work. We did not consider the impact of storage mechanisms on the performance and availability of our environments, which is a relevant aspect to explore in future studies. Additionally, our analysis did not encompass modeling techniques such as Stochastic Petri Nets, which offer more detailed representations of the environments. Furthermore, we did not address sensitivity analysis techniques like Regression, Correlation, and Perturbation Analysis in this study, leaving room for future exploration of these valuable methods.

7.3 FUTURE WORK

In Section 6.2, we focused on a steady workload for our evaluations. However, as part of future work, we aim to explore different workloads to gain a deeper understanding of the system's behavior under varying conditions. Additionally, we plan to incorporate machine learning techniques along with statistical methods to predict how the system will perform when availability and performance parameters are modified. Furthermore, we envision extending our research to include sensitivity analysis approaches that encompass alternative modeling formalisms and explore diverse compositions for sensitivity indices. Another intriguing avenue for future work is the development of a capacity planning tool, allowing users to input specific parameters and obtain results in the form of tables and charts, aiding in efficient resource allocation and decision-making processes.

REFERENCES

ADDIS, B.; ARDAGNA, D.; PANICUCCI, B.; SQUILLANTE, M. S.; ZHANG, L. A hierarchical approach for the resource management of very large cloud platforms. *IEEE Transactions on Dependable and Secure Computing*, IEEE, v. 10, n. 5, p. 253–272, 2013.

AGARWAL, P. K.; NAUGHTON, T.; PARK, B. H.; BERNHOLDT, D. E.; HURSEY, J. J.; GEIST, A. Application health monitoring for extreme-scale resiliency using cooperative fault management. *Concurrency and Computation: Practice and Experience*, Wiley Online Library, v. 32, n. 2, p. e5449, 2020.

AHMED, M.; CHOWDHURY, A. S. M. R.; AHME, M.; RAFEE, M. M. H. An advanced survey on cloud computing and state-of-the-art research issues. *International Journal of Computer Science Issues (IJCSI)*, International Journal of Computer Science Issues (IJCSI), v. 9, n. 1, p. 201, 2012.

ALDAĞ, M.; KIRSAL, Y.; ÜLKER, S. An analytical modelling and qos evaluation of fault-tolerant load balancer and web servers in fog computing. *The Journal of Supercomputing*, Springer, v. 78, n. 10, p. 12136–12158, 2022.

ANDRADE, E.; NOGUEIRA, B. Performability evaluation of a cloud-based disaster recovery solution for it environments. *Journal of Grid Computing*, Springer, v. 17, n. 3, p. 603–621, 2019.

ANDRADE, E.; NOGUEIRA, B.; JÚNIOR, I. de F.; ARAÚJO, D. Performance and availability trade-offs in fog–cloud iot environments. *Journal of Network and Systems Management*, Springer, v. 29, n. 1, p. 1–27, 2020.

ANGIN, P.; BHARGAVA, B.; JIN, Z. A self-cloning agents based model for high-performance mobile-cloud computing. In: IEEE. *2015 IEEE 8th International Conference on Cloud Computing*. [S.l.], 2015. p. 301–308.

ARAUJO, E.; DANTAS, J.; MATOS, R.; PEREIRA, P.; MACIEL, P. Dependability evaluation of an iot system: A hierarchical modelling approach. In: IEEE. *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. [S.l.], 2019. p. 2121–2126.

ATAIE, E.; ENTEZARI-MALEKI, R.; RASHIDI, L.; TRIVEDI, K. S.; ARDAGNA, D.; MOVAGHAR, A. Hierarchical stochastic models for performance, availability, and power consumption analysis of iaas clouds. *IEEE Transactions on Cloud Computing*, IEEE, 2017.

AVIŽIENIS, A. Design of fault-tolerant computers. In: *Proceedings of the November 14-16, 1967, fall joint computer conference*. [S.l.: s.n.], 1967. p. 733–743.

AVIZIENIS, A.; LAPRIE, J.-C.; RANDELL, B. Fundamental concepts of computer system dependability. In: CITESEER. *Workshop on Robot Dependability: Technological Challenge of Dependable Robots in Human Environments*. [S.l.], 2001. p. 1–16.

BAKHSHI, Z.; RODRIGUEZ-NAVAS, G. A preliminary roadmap for dependability research in fog computing. *ACM SIGBED Review*, ACM New York, NY, USA, v. 16, n. 4, p. 14–19, 2020.

BANKOLE, A. A.; AJILA, S. A. et al. Cloud client prediction models for cloud resource provisioning in a multitier web application environment. In: IEEE. *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*. [S.l.], 2013. p. 156–161.

BARZEGARAN, M.; POP, P. Communication scheduling for control performance in tsn-based fog computing platforms. *IEEE Access*, IEEE, v. 9, p. 50782–50797, 2021.

BATTULA, S. K.; O'REILLY, M. M.; GARG, S.; MONTGOMERY, J. A generic stochastic model for resource availability in fog computing environments. *IEEE Transactions on Parallel and Distributed Systems*, IEEE, v. 32, n. 4, p. 960–974, 2020.

BAUER, E.; ADAMS, R. et al. *Reliability and availability of cloud computing*. [S.l.]: John Wiley & Sons, 2012.

BOLCH, G.; GREINER, S.; MEER, H. D.; TRIVEDI, K. S. et al. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. [S.l.]: John Wiley & Sons, 2006.

BONOMI, F.; MILITO, R.; ZHU, J.; ADDEPALLI, S. Fog computing and its role in the internet of things. In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. [S.l.: s.n.], 2012. p. 13–16.

BOUKERCHE, A.; SOTO, V. An efficient mobility-oriented retrieval protocol for computation offloading in vehicular edge multi-access network. *IEEE Transactions on Intelligent Transportation Systems*, IEEE, 2020.

BRILHANTE, J.; SILVA, B.; MACIEL, P.; ZIMMERMANN, A. Eucabomber 2.0: A tool for dependability tests in eucalyptus cloud infrastructures considering vm life-cycle. In: IEEE. *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. [S.l.], 2014. p. 2669–2674.

CAI, H.; XU, B.; JIANG, L.; VASILAKOS, A. V. Iot-based big data storage systems in cloud computing: perspectives and challenges. *IEEE Internet of Things Journal*, IEEE, v. 4, n. 1, p. 75–87, 2016.

CALIRI, G. V. Introduction to analytical modeling. In: *Int. CMG Conference*. [S.l.: s.n.], 2000. p. 31–36.

CAMPOS, E.; MATOS, R.; MACIEL, P.; COSTA, I.; SILVA, F. A.; SOUZA, F. Performance evaluation of virtual machines instantiation in a private cloud. In: IEEE. *2015 IEEE World Congress on Services*. [S.l.], 2015. p. 319–326.

CAO, Q.; SHEN, L.; XIE, W.; PARKHI, O. M.; ZISSERMAN, A. Vggface2: A dataset for recognising faces across pose and age. In: IEEE. *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*. [S.l.], 2018. p. 67–74.

CASSANDRAS, C. G.; LAFORTUNE, S. et al. *Introduction to discrete event systems*. [S.l.]: Springer Science & Business Media, 2009.

CHATFIELD, C.; ZIDEK, J.; LINDSEY, J. et al. *An introduction to generalized linear models*. [S.l.]: Chapman and Hall/CRC, 2010.

CHEN, X.-W.; LIN, X. Big data deep learning: challenges and perspectives. *IEEE access*, Ieee, v. 2, p. 514–525, 2014.

CHIANG, M.; HA, S.; CHIH-LIN, I.; RISSO, F.; ZHANG, T. Clarifying fog computing and networking: 10 questions and answers. *IEEE Communications Magazine*, IEEE, v. 55, n. 4, p. 18–20, 2017.

CONSORTIUM, O. et al. Openfog reference architecture for fog computing. *Architecture Working Group*, p. 1–162, 2017.

CONSTANTIN, H. Markov chains and queueing theory. *Simulating Queuing Systems: A Test of Parameter Change*, p. 1–13, 2011.

COPPOLA, G. K. *Both Ends of the Rainbow: Lomilomi˜ a Healing Journey*. [S.l.]: Balboa Press, 2013.

DANTAS, J.; ARAUJO, E.; MACIEL, P.; MATOS, R.; TEIXEIRA, J. Estimating capacity-oriented availability in cloud systems. *International Journal of Computational Science and Engineering*, Inderscience Publishers (IEL), v. 22, n. 4, p. 466–476, 2020.

DANTAS, J.; MATOS, R.; ARAUJO, J.; MACIEL, P. An availability model for eucalyptus platform: An analysis of warm-standy replication mechanism. In: IEEE. *2012 IEEE international conference on Systems, man, and cybernetics (SMC)*. [S.l.], 2012. p. 1664–1669.

DANTAS, J.; MATOS, R.; ARAUJO, J.; OLIVEIRA, D.; OLIVEIRA, A.; MACIEL, P. Hierarchical model and sensitivity analysis for a cloud-based vod streaming service. In: IEEE. *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*. [S.l.], 2016. p. 10–16.

DASTJERDI, A. V.; GUPTA, H.; CALHEIROS, R. N.; GHOSH, S. K.; BUYYA, R. Fog computing: Principles, architectures, and applications. In: *Internet of things*. [S.l.]: Elsevier, 2016. p. 61–75.

DASTJERDI, A. V.; GUPTA, H.; CALHEIROS, R. N.; GHOSH, S. K.; BUYYA, R. Fog computing: Principles, architectures, and applications. *arXiv*, p. arXiv–1601, 2016.

DEHNAVI, S.; FARAGARDI, H. R.; KARGAHI, M.; FAHRINGER, T. A reliability-aware resource provisioning scheme for real-time industrial applications in a fog-integrated smart factory. *Microprocessors and Microsystems*, Elsevier, v. 70, p. 1–14, 2019.

DEVORE, J. L. Probability and statistics for engineering and the sciences. Spinger, 2008.

DUTREILH, X.; MOREAU, A.; MALENFANT, J.; RIVIERRE, N.; TRUCK, I. et al. From data center resource allocation to control theory and back. In: IEEE. *Cloud Computing (CLOUD), 2010 IEEE 3rd Int. Conf. on*. [S.l.], 2010. p. 410–417.

EVER, E. Performability analysis methods for clustered wsns as enabling technology for iot. In: *Performability in Internet of Things*. [S.l.]: Springer, 2019. p. 1–19.

EVER, E.; SHAH, P.; MOSTARDA, L.; OMONDI, F.; GEMIKONAKLI, O. On the performance, availability and energy consumption modelling of clustered iot systems. *Computing*, Springer, v. 101, n. 12, p. 1935–1970, 2019.

FACCHINETTI, D.; PSAILA, G.; SCANDURRA, P. Mobile cloud computing for indoor emergency response: the ipsos assistant case study. *Journal of Reliable Intelligent Environments*, Springer, v. 5, n. 3, p. 173–191, 2019.

Fernandes, S.; Tavares, E.; Santos, M.; Lira, V.; Maciel, P. Dependability assessment of virtualized networks. In: *2012 IEEE International Conference on Communications (ICC)*. [S.l.: s.n.], 2012. p. 2711–2716.

GAMATIÉ, A.; DEVIC, G.; SASSATELLI, G.; BERNABOVI, S.; NAUDIN, P.; CHAPMAN, M. Towards energy-efficient heterogeneous multicore architectures for edge computing. *IEEE Access*, IEEE, v. 7, p. 49474–49491, 2019.

GHOSH, R.; LONGO, F.; FRATTINI, F.; RUSSO, S.; TRIVEDI, K. S. Scalable analytics for iaas cloud availability. *IEEE Transactions on Cloud Computing*, IEEE, v. 2, n. 1, p. 57–70, 2014.

GHOSH, R.; LONGO, F.; XIA, R.; NAIK, V. K.; TRIVEDI, K. S. Stochastic model driven capacity planning for an infrastructure-as-a-service cloud. *IEEE Transactions on Services Computing*, IEEE, v. 7, n. 4, p. 667–680, 2013.

GOKHALE, S. S.; TRIVEDI, K. S. Analytical modeling. *Encyclopedia of Distributed Systems*, Citeseer, 1998.

GOMES, E.; COSTA, F.; ROLT, C. D.; PLENTZ, P.; DANTAS, M. A survey from real-time to near real-time applications in fog computing environments. In: MDPI. *Telecom*. [S.l.], 2021. v. 2, n. 4, p. 489–517.

GORBENKO, A.; ROMANOVSKY, A.; TARASYUK, O. Fault tolerant internet computing: Benchmarking and modelling trade-offs between availability, latency and consistency. *Journal of Network and Computer Applications*, Elsevier, v. 146, p. 102412, 2019.

GUAN, S.; GRANDE, R. E. D.; BOUKERCHE, A. A novel energy efficient platform based model to enable mobile cloud applications. In: IEEE. *2016 IEEE Symposium on Computers and Communication (ISCC)*. [S.l.], 2016. p. 914–919.

HA, K.; CHEN, Z.; HU, W.; RICHTER, W.; PILLAI, P.; SATYANARAYANAN, M. Towards wearable cognitive assistance. In: *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. [S.l.: s.n.], 2014. p. 68–81.

HAMBY, D. et al. A review of techniques for parameter sensitivity analysis of environmental models. *Environmental monitoring and assessment*, Springer, v. 32, n. 2, p. 135–154, 1994.

HARDESTY, L. *Fog computing group publishes reference architecture*. 2017.

HOQUE, M. A.; SIEKKINEN, M.; NURMINEN, J. K.; AALTO, M.; TARKOMA, S. Mobile multimedia streaming techniques: Qoe and energy saving perspective. *Pervasive and Mobile Computing*, Elsevier, v. 16, p. 96–114, 2015.

HUANG, C.-F.; HUANG, D.-H.; LIN, Y.-K. Network reliability evaluation for a distributed network with edge computing. *Computers & Industrial Engineering*, Elsevier, v. 147, p. 106492, 2020.

HUANG, J.; LIANG, J.; ALI, S. A simulation-based optimization approach for reliability-aware service composition in edge computing. *IEEE Access*, IEEE, v. 8, p. 50355–50366, 2020.

JAIN, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. [S.l.]: John Wiley & Sons, 1990.

JAMMAL, M.; HAWILO, H.; KANSO, A.; SHAMI, A. Ace: Availability-aware cloudsim extension. *IEEE Transactions on Network and Service Management*, IEEE, v. 15, n. 4, p. 1586–1599, 2018.

JAMMAL, M.; KANSO, A.; SHAMI, A. Chase: Component high availability-aware scheduler in cloud computing environment. In: IEEE. *2015 IEEE 8th International Conference on Cloud Computing*. [S.l.], 2015. p. 477–484.

JAYASHREE, L.; SELVAKUMAR, G. Edge computing in iot. In: *Getting Started with Enterprise Internet of Things: Design Approaches and Software Architecture Models*. [S.l.]: Springer, 2020. p. 49–69.

JIA, C.; LIN, K.; DENG, J. A multi-property method to evaluate trust of edge computing based on data driven capsule network. In: IEEE. *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. [S.l.], 2020. p. 616–621.

KAFHALI, S. E.; SALAH, K. Efficient and dynamic scaling of fog nodes for iot devices. *The Journal of Supercomputing*, Springer, v. 73, n. 12, p. 5261–5284, 2017.

KOLMOGOROFF, A. Über die analytischen methoden in der wahrscheinlichkeitsrechnung. *Mathematische Annalen*, Springer, v. 104, p. 415–458, 1931.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, AcM New York, NY, USA, v. 60, n. 6, p. 84–90, 2017.

LAPRIE, J.-C. Dependability: Basic concepts and terminology. In: *Dependability: Basic Concepts and Terminology*. [S.l.]: Springer, 1992. p. 3–245.

LECLERC, B.; CALE, J. *Big Data*. [S.l.]: Routledge, 2020.

LEE, J.-w.; JANG, G.; JUNG, H.; LEE, J.-G.; LEE, U. Maximizing mapreduce job speed and reliability in the mobile cloud by optimizing task allocation. *Pervasive and Mobile Computing*, Elsevier, v. 60, p. 101082, 2019.

LI, C.; WANG, Y.; TANG, H.; ZHANG, Y.; XIN, Y.; LUO, Y. Flexible replica placement for enhancing the availability in edge computing environment. *Computer Communications*, Elsevier, v. 146, p. 1–14, 2019.

LI, J.; ZHANG, T.; JIN, J.; YANG, Y.; YUAN, D.; GAO, L. Latency estimation for fog-based internet of things. In: IEEE. *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*. [S.l.], 2017. p. 1–6.

LI, S.; HUANG, J. Gspn-based reliability-aware performance evaluation of iot services. In: IEEE. *2017 IEEE International Conference on Services Computing (SCC)*. [S.l.], 2017. p. 483–486.

LIANG, W.; MA, Y.; XU, W.; JIA, X.; CHAU, S. C.-K. Reliability augmentation of requests with service function chain requirements in mobile edge-cloud networks. In: *49th International Conference on Parallel Processing - ICPP*. New York, NY, USA: Association for Computing Machinery, 2020. (ICPP '20). ISBN 9781450388160.

LILJA, D. J. et al. *Measuring computer performance: a practitioner's guide*. [S.l.]: Cambridge university press, 2005.

LISBOA, M. F. F. da S.; SANTOS, G. L.; LYNN, T.; SADOK, D.; KELNER, J.; ENDO, P. T. et al. Modeling the availability of an e-health system integrated with edge, fog and cloud infrastructures. In: IEEE. *2018 IEEE Symposium on Computers and Communications (ISCC)*. [S.l.], 2018. p. 00416–00421.

LIU, B.; CHANG, X.; HAN, Z.; TRIVEDI, K.; RODRÍGUEZ, R. J. Model-based sensitivity analysis of iaas cloud availability. *Future Generation Computer Systems*, Elsevier, v. 83, p. 1–13, 2018.

LIU, F.; NARAYANAN, A.; BAI, Q. Real-time systems. Citeseer, 2000.

LIU, Y.; WANG, K.; GE, L.; YE, L.; CHENG, J. Adaptive evaluation of virtual machine placement and migration scheduling algorithms using stochastic petri nets. *IEEE Access*, IEEE, v. 7, p. 79810–79824, 2019.

LONGBOTTOM, R. *Roy Longbottom's Raspberry Pi, Pi 2 and Pi 3 Benchmarks*. 2017.

LONGO, F.; GHOSH, R.; NAIK, V. K.; TRIVEDI, K. S. A scalable availability model for infrastructure-as-a-service cloud. In: IEEE. *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*. [S.l.], 2011. p. 335–346.

MACHIDA, F.; ANDRADE, E.; KIM, D. S.; TRIVEDI, K. S. Candy: Component-based availability modeling framework for cloud service management using sysml. In: IEEE. *2011 IEEE 30th International Symposium on Reliable Distributed Systems*. [S.l.], 2011. p. 209–218.

MACIEL, P. R.; TRIVEDI, K. S.; MATIAS, R.; KIM, D. S. Dependability modeling. In: *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*. [S.l.]: IGI Global, 2012. p. 53–97.

MACIEL, P. R. M. *Performance, reliability, and availability evaluation of computational systems, Volume 2: Reliability, availability modeling, measuring, and data analysis*. [S.l.]: CRC Press, 2023.

MACIEL, P. R. M. *Performance, reliability, and availability evaluation of computational systems, volume I: performance and background*. [S.l.]: CRC Press, 2023.

MAHMOOD, Z.; RAMACHANDRAN, M. Fog computing: concepts, principles and related paradigms. In: *Fog Computing*. [S.l.]: Springer, 2018. p. 3–21.

MAO, K.; ZHU, Y.; CHEN, Z.; TAO, X.; XUE, Q.; WU, H.; MAO, Y.; HOU, J. A visual model-based evaluation framework of cloud-based prognostics and health management. In: IEEE. *2017 IEEE International Conference on Smart Cloud (SmartCloud)*. [S.l.], 2017. p. 33–40.

MARIAPPAN, M.; THONG, L. K.; MUTHUKARUPPAN, K. A design methodology of an embedded motion-detecting video surveillance system. *International Journal of Integrated Engineering*, v. 12, n. 2, p. 55–69, 2020.

MARJANI, M.; NASARUDDIN, F.; GANI, A.; KARIM, A.; HASHEM, I. A. T.; SIDDIQA, A.; YAQOOB, I. Big iot data analytics: architecture, opportunities, and open research challenges. *ieee access*, IEEE, v. 5, p. 5247–5261, 2017.

MATOS, R.; ARAUJO, J.; OLIVEIRA, D.; MACIEL, P.; TRIVEDI, K. Sensitivity analysis of a hierarchical model of mobile cloud computing. *Simulation Modelling Practice and Theory*, Elsevier, v. 50, p. 151–164, 2015.

MATOS, R.; DANTAS, J.; ARAUJO, J.; TRIVEDI, K. S.; MACIEL, P. Redundant eucalyptus private clouds: Availability modeling and sensitivity analysis. *J. Grid Comput.*, Springer-Verlag, v. 15, n. 1, p. 1–22, mar. 2017. ISSN 1570-7873.

MATOS, R.; DANTAS, J.; ARAUJO, J.; TRIVEDI, K. S.; MACIEL, P. Redundant eucalyptus private clouds: Availability modeling and sensitivity analysis. *Journal of Grid Computing*, Springer, v. 15, n. 1, p. 1–22, 2017.

MATOS, R. d. S. M. J. et al. Identification of availability and performance bottlenecks in cloud computing systems: an approach based on hierarchical models and sensitivity analysis. Universidade Federal de Pernambuco, 2016.

MAURO, M. D.; GALATRO, G.; LONGO, M.; POSTIGLIONE, F.; TAMBASCO, M. Ip multimedia subsystem in a containerized environment: availability and sensitivity evaluation. In: IEEE. *2019 IEEE Conference on Network Softwarization (NetSoft)*. [S.l.], 2019. p. 42–47.

MELL, P.; GRANCE, T. et al. The nist definition of cloud computing. In: . [S.l.]: Computer Security Division, Information Technology Laboratory, National . . . , 2011.

MELO, C.; DANTAS, J.; MACIEL, P.; OLIVEIRA, D. M.; ARAUJO, J.; MATOS, R.; FÉ, I. Models for hyper-converged cloud computing infrastructures planning. *International Journal of Grid and Utility Computing*, Inderscience Publishers (IEL), v. 11, n. 2, p. 196–208, 2020.

MELO, C.; DANTAS, J.; MACIEL, R.; PEREIRA, P.; QUESADO, E.; MACIEL, P. Blockchain provisioning over private cloud computing environments: Availability modeling and cost requirements. In: IEEE. *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*. [S.l.], 2019. p. 1–3.

MELO, C.; DANTAS, J.; MACIEL, R.; SILVA, P.; MACIEL, P. Models to evaluate service provisioning over cloud computing environments-a blockchain-as-a-service case study. *Revista de Informática Teórica e Aplicada*, v. 26, n. 3, p. 65–74, 2019.

MELO, C.; DANTAS, J.; OLIVEIRA, D.; FÉ, I.; MATOS, R.; DANTAS, R.; MACIEL, R.; MACIEL, P. Dependability evaluation of a blockchain-as-a-service environment. In: IEEE. *2018 IEEE Symposium on Computers and Communications (ISCC)*. [S.l.], 2018. p. 00909–00914.

MELO, C.; MATOS, R.; DANTAS, J.; MACIEL, P. Capacity-oriented availability model for resources estimation on private cloud infrastructure. In: IEEE. *Dependable Computing (PRDC), 2017 IEEE 22nd Pacific Rim International Symposium on*. [S.l.], 2017. p. 255–260.

MELO, R.; PAULO, F. Vicente de; FILHO, I. J. de M.; FELICIANO, F.; MACIEL, P. R. M. Redundancy mechanisms applied in cloud computing infrastructures. In: IEEE. *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*. [S.l.], 2018. p. 1–6.

MENASCE, D. A.; ALMEIDA, V. A.; DOWDY, L. W.; DOWDY, L. et al. *Performance by design: computer capacity planning by example*. [S.l.]: Prentice Hall Professional, 2004.

MIAO, W.; MIN, G.; ZHANG, X.; ZHAO, Z.; HU, J. Performance modelling and quantitative analysis of vehicular edge computing with bursty task arrivals. *IEEE Transactions on Mobile Computing*, IEEE, 2021.

NAHA, R. K.; GARG, S.; GEORGAKOPOULOS, D.; JAYARAMAN, P. P.; GAO, L.; XIANG, Y.; RANJAN, R. Fog computing: Survey of trends, architectures, requirements, and research directions. *IEEE access*, IEEE, v. 6, p. 47980–48009, 2018.

NGUYEN, T. A.; MIN, D.; CHOI, E. A hierarchical modeling and analysis framework for availability and security quantification of iot infrastructures. *Electronics*, Multidisciplinary Digital Publishing Institute, v. 9, n. 1, p. 155, 2020.

ÖHMANN, D.; SIMSEK, M.; FETTWEIS, G. P. Achieving high availability in wireless networks by an optimal number of rayleigh-fading links. In: IEEE. *2014 IEEE Globecom Workshops (GC Wkshps)*. [S.l.], 2014. p. 1402–1407.

PATEL, P.; RANABAHU, A. H.; SHETH, A. P. Service level agreement in cloud computing. In: . [S.l.: s.n.], 2009.

PATIL, V.; ASHRA, J.; GAJINKAR, M.; FAIZEE, S. A review: Office monitoring and surveillance system. *Available at SSRN 3561737*, 2020.

PEREIRA, P.; ARAUJO, J.; MACIEL, P. A hybrid mechanism of horizontal auto-scaling based on thresholds and time series. In: IEEE. *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. [S.l.], 2019. p. 2065–2070.

PEREIRA, P.; ARAUJO, J.; MATOS, R.; PREGUIÇA, N.; MACIEL, P. Software rejuvenation in computer systems: An automatic forecasting approach based on time series. In: IEEE. *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*. [S.l.], 2018. p. 1–8.

PEREIRA, P.; ARAUJO, J.; MELO, C.; SANTOS, V.; MACIEL, P. Analytical models for availability evaluation of edge and fog computing nodes. *The Journal of Supercomputing*, Springer, p. 1–29, 2021.

PEREIRA, P.; ARAUJO, J.; TORQUATO, M.; DANTAS, J.; MELO, C.; MACIEL, P. Stochastic performance model for web server capacity planning in fog computing. *The Journal of Supercomputing*, Springer, p. 1–25, 2020.

PETRESCU, R. V. Face recognition as a biometric application. *Journal of Mechatronics and Robotics*, v. 3, p. 237–257, 2019.

PINHEIRO, T.; OLIVEIRA, D.; MATOS, R.; SILVA, B.; PEREIRA, P.; MELO, C.; OLIVEIRA, F.; TAVARES, E.; DANTAS, J.; MACIEL, P. The mercury environment: A modeling tool for performance and dependability evaluation. In: *Intelligent Environments 2021*. [S.l.]: IOS Press, 2021. p. 16–25.

PRAVEEN, D.; RAJ, D. P. Smart traffic management system in metropolitan cities. *Journal of Ambient Intelligence and Humanized Computing*, Springer, p. 1–13, 2020.

QIU, X.; DAI, Y.; XIANG, Y.; XING, L. Correlation modeling and resource optimization for cloud service with fault recovery. *IEEE Transactions on Cloud Computing*, IEEE, 2017.

RAMUNDO, L.; OTCU, G. B.; TERZI, S. Sustainability model for 3d food printing adoption. In: IEEE. *2020 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. [S.l.], 2020. p. 1–9.

ROY, N.; DUBEY, A.; GOKHALE, A. et al. Efficient autoscaling in the cloud using predictive models for workload forecasting. In: IEEE. *Cloud Computing (CLOUD), 2011 IEEE Int. Conf. on*. [S.l.], 2011. p. 500–507.

RUIZ, L. B.; SIQUEIRA, I. G.; OLIVEIRA, L. B. e.; WONG, H. C.; NOGUEIRA, J. M. S.; LOUREIRO, A. A. Fault management in event-driven wireless sensor networks. In: *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*. [S.l.: s.n.], 2004. p. 149–156.

SAMIE, F.; BAUER, L.; HENKEL, J. Iot technologies for embedded computing: A survey. In: IEEE. *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. [S.l.], 2016. p. 1–10.

SANTOS, G. L.; ENDO, P. T.; LISBOA, M. F. F. da S.; SILVA, L. G. F. da; SADOK, D.; KELNER, J.; LYNN, T. et al. Analyzing the availability and performance of an e-health system integrated with edge, fog and cloud infrastructures. *Journal of Cloud Computing*, Springer, v. 7, n. 1, p. 16, 2018.

SANYAL, S.; ZHANG, P. Improving quality of data: Iot data aggregation using device to device communications. *IEEE Access*, IEEE, v. 6, p. 67830–67840, 2018.

SHARKH, M. A.; KALIL, M. A quest for optimizing the data processing decision for cloud-fog hybrid environments. In: IEEE. *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*. [S.l.], 2018. p. 1–6.

SHARMA, P. K.; CHEN, M.-Y.; PARK, J. H. A software defined fog node based distributed blockchain cloud architecture for iot. *Ieee Access*, IEEE, v. 6, p. 115–124, 2017.

SHI, W.; CAO, J.; ZHANG, Q.; LI, Y.; XU, L. Edge computing: Vision and challenges. *IEEE internet of things journal*, IEEE, v. 3, n. 5, p. 637–646, 2016.

SILVA, B.; MATOS, R.; CALLOU, G.; FIGUEIREDO, J.; OLIVEIRA, D.; FERREIRA, J.; DANTAS, J.; LOBO, A.; ALVES, V.; MACIEL, P. Mercury: An integrated environment for performance and dependability evaluation of general systems. In: *Proceedings of Industrial Track at 45th Dependable Systems and Networks Conference, DSN*. [S.l.: s.n.], 2015.

SIVASUBRAMANIAM, A.; RAMACHANDRAN, U.; VENKATESWARAN, H. A comparative evaluation of techniques for studying parallel system performance. *Georgia Institute of Technology*, p. 1–24, 1994.

SMITH, R.; TRIVEDI, K. S.; RAMESH, A. Performability analysis: measures, an algorithm, and a case study. *IEEE Transactions on Computers*, IEEE, v. 37, n. 4, p. 406–417, 1988.

SOTOMAYOR, B.; MONTERO, R. S.; LLORENTE, I. M.; FOSTER, I. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet computing*, IEEE, v. 13, n. 5, p. 14–22, 2009.

Sousa, E.; Lins, F.; Tavares, E.; Cunha, P.; Maciel, P. A modeling approach for cloud infrastructure planning considering dependability and cost requirements. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, v. 45, n. 4, p. 549–558, 2015.

SOUZA, D.; MATOS, R.; ARAUJO, J.; ALVES, V.; MACIEL, P. Eucabomber: Experimental evaluation of availability in eucalyptus private clouds. In: IEEE. *2013 IEEE International Conference on Systems, Man, and Cybernetics*. [S.l.], 2013. p. 4080–4085.

STATISTA. *Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2021, with forecasts from 2022 to 2030*. 2022. Disponível em: <https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/>.

STEWART, W. J. *Probability, Markov chains, queues, and simulation: the mathematical basis of performance modeling*. [S.l.]: Princeton university press, 2009.

SUN, W.; LIU, J. Coordinated multipoint-based uplink transmission in internet of things powered by energy harvesting. *IEEE Internet of Things Journal*, IEEE, v. 5, n. 4, p. 2585–2595, 2017.

SUNYAEV, A. Fog and edge computing. In: *Internet Computing*. [S.l.]: Springer, 2020. p. 237–264.

TIAN, Y.; TIAN, J.; LI, N. Cloud reliability and efficiency improvement via failure risk based proactive actions. *Journal of Systems and Software*, Elsevier, v. 163, p. 110524, 2020.

TORQUATO, M.; UMESH, I.; MACIEL, P. Models for availability and power consumption evaluation of a private cloud with vmm rejuvenation enabled by vm live migration. *The Journal of Supercomputing*, Springer, v. 74, n. 9, p. 4817–4841, 2018.

TRIVEDI, K. S.; BOBBIO, A. *Reliability and availability engineering: modeling, analysis, and applications*. [S.l.]: Cambridge University Press, 2017.

TRIVEDI, K. S.; HUNTER, S.; GARG, S.; FRICKS, R. *Reliability analysis techniques explored through a communication network example*. [S.l.], 1996.

TRIVEDI, K. S.; SATHAYE, A.; RAMANI, S. Availability modeling in practice.

WEBER, T.; FERRIN, M.; SWEENEY, F.; AHMED, S.; SHARMIN, M. Towards identifying the optimal timing for near real-time smoking interventions using commercial wearable devices. In: IEEE. *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. [S.l.], 2020. p. 429–438.

YAKUBU, J.; CHRISTOPHER, H. A.; CHIROMA, H.; ABDULLAHI, M. et al. Security challenges in fog-computing environment: a systematic appraisal of current developments. *Journal of Reliable Intelligent Environments*, Springer, v. 5, n. 4, p. 209–233, 2019.

YE, Z. *Performance Analysis of HTTP Adaptive Video Streaming Services in Mobile Networks*. Tese (Doutorado) — Université d'Avigno, 2017.

YI, S.; HAO, Z.; QIN, Z.; LI, Q. Fog computing: Platform and applications. In: IEEE. *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*. [S.l.], 2015. p. 73–78.

YOUSEFPOUR, A.; DEVIC, S.; NGUYEN, B. Q.; KREIDIEH, A.; LIAO, A.; BAYEN, A. M.; JUE, J. P. Guardians of the deep fog: Failure-resilient dnn inference from edge to cloud. In: *Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*. New York, NY, USA: Association for Computing Machinery, 2019. (AIChallengeIoT'19), p. 25–31. ISBN 9781450370134.

YOUSEFPOUR, A.; FUNG, C.; NGUYEN, T.; KADIYALA, K.; JALALI, F.; NIAKANLAHIJI, A.; KONG, J.; JUE, J. P. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, Elsevier, v. 98, p. 289–330, 2019.

ZHANG, Y.; THORBURN, P. J. Handling missing data in near real-time environmental monitoring: A system and a review of selected methods. *Future Generation Computer Systems*, Elsevier, v. 128, p. 63–72, 2022.

ZILIC, J.; ARAL, A.; BRANDIC, I. Efpo: Energy efficient and failure predictive edge offloading. In: *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. New York, NY, USA: Association for Computing Machinery, 2019. (UCC'19), p. 165–175. ISBN 9781450368940.