Learning based Adaptation for Fog and Edge Computing Applications and Services

Mauricio Fadel Argerich

Thesis presented for the
Master's Degree in Data Science

SAPIENZA
UNIVERSITÀ DI ROMA

NEC

Facoltà I3S
Sapienza - Università di Roma
Rome, Italy

IOT Group
NEC Laboratories Europe
Heidelberg, Germany

October 2018

Dedicated to

My family, especially my parents Adriana and Mario, and my aunt Analia; who have always supported me and encouraged my desire to discover, learn and create.

Learning based Adaptation for Fog and Edge Computing Applications and Services

Mauricio Fadel Argerich

Submitted for the Master's Degree in Data Science
October 2018

Abstract

Increasing data traffic and network utilization are one of the biggest challenges for network operators nowadays, due to the massive amount of data generated by devices in the edge in the context of the Internet of Things (IoT). Edge and Fog computing allow network operators to reduce network stress and improve the responsiveness of the services by allocating computation closer to data producers and consumers. However, developing and managing applications is a challenging task because of hardware heterogeneity, limited elasticity and unstable network connections that characterize the edge.

In order to facilitate the development, as well as improving the efficiency of applications and services, an adaptive framework is proposed. This adaptive framework, uses Reinforcement Learning to combine adaptation mechanisms, specified in the development phase, during runtime in order to optimize the performance of applications and services. An implementation of this approach is realized on Python along with AdAS, an Adaptive Applications Simulator to evaluate its efficiency.

Throughout simulations, the adaptive framework manages to achieve a requirement satisfaction of almost 85% while keeping a high precision in unstable execution contexts on low power devices (i.e., changing networking bandwidths and available resource). In addition, the Reinforcement Learning approach shows to be more flexible and efficient than a pre-programmed adaptive logic, yielding a precision between 10% and 25% higher.

Acknowledgements

I would like to thank my university supervisor Ioannis for his guidance throughout the development of this work and his never ending enthusiasm and positivism. I also thank my external supervisors Jonathan and Bin, who have given me the chance of being part of their research group, and have extensively collaborated with my work while sharing their knowledge and experience with me.

I am grateful to my colleagues Davide, Yoann, Stefano and Julia for the long discussions and their brilliant insights that have helped me to overcome the challenges I have faced in the development of this work.

I want to thank La Sapienza, Università di Roma, for giving me the opportunity of accomplishing my Master's degree at one of the most prestigious institutions of Europe, as well as NEC Laboratories, for giving me the chance to be a part of their research team.

Throughout the development of my Master's Degree and in particular this thesis, I have had the opportunity to meet and work with many wonderful people, many more than the ones named here. I have learnt from all of them, professionally and personally, and for this I will be forever grateful.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

The last decade has been marked by a steady increase in the number of devices located at the user-facing end of the Internet. New smartphones, tablets and a plethora of smart, always-on and connected devices have proliferated, creating a new scenario where enormous amounts of data are generated at the edge. However, in order to be processed, all of these data are usually sent to the cloud.

Because of this, increasing data traffic and network utilization is one of the biggest challenges for network operators nowadays. Many mobile and IoT applications depend on remote services, creating high network load due to data traffic. In fact, global internet traffic is estimated to grow nearly threefold in the next 5 years according to recent research by Cisco Systems [1].

In addition, sending these data back and forth to the Cloud poses several security and privacy concerns. New encryption mechanisms have risen to secure the end-to-end communication with the Cloud, but even assuming a safe communication, sensitive and private data has to be shared with the Cloud in order to be processed.

All of these reasons have led to Edge and Fog Computing, two new paradigms which allow network operators to reduce network stress and improve the responsiveness of the services provided, by allocating computational efforts closer to data producers and consumers.

Edge and Fog Computing do not aim to replace Cloud Computing but to complement

it. They implement the fundamentals of the Cloud Computing paradigm, but do this as close as possible to the sources of the data. The Cloud is also seen as an available resource, but its availability is not considered as permanent; this is due to the fact that most edge devices are mobile and have wireless connections that are prone to network outages as well as speed and bandwidth fluctuations.

These technologies have given light to novel applications such as FAST, a fog computing assisted distributed analytics system to monitor fall for stroke mitigation[2] and Car2Car, which distributes traffic and meteorological alerts on the highway from car to car, instead of connecting to a central node. Furthermore, edge and fog frameworks, like FogFlow[3], have been created to facilitate the development of applications and services that use these paradigms.

1.2 Problem

Even though the adoption of Edge and Fog computing brings many benefits to network operators and developers, they also impose new challenges:

Hardware Heterogeneity. Edge and Fog applications and services must run on different devices, in a wide range of contexts. Because of this, applications and services might provide very different performance characteristics according to where they are deployed.

Mobility. The connectivity of the devices in the edge is usually wireless, which means the connectivity to edge nodes is intermittent or highly variable. This requires distributed applications to be deployed taking into account scenarios with low and no connectivity in order to avoid failing to provide the expected service in these scenarios.

Elasticity. The cloud provides elasticity mechanisms, that the edge cannot provide [4]. Services in the cloud scale horizontally over multiple machines or vertically by increasing their RAM and CPU share, this is in contrast with edge deployments in which the processing might be done in a single machine with limited hardware capabilities.

Programmability: to take advantage of Fog and Edge computing, developers must partition the functions of their applications between the Fog, the Edge and the Cloud. This is generally done manually, which is not scalable or extensible [5].

Privacy and Security: the data sensed in the edge is usually private and sensitive. Privacy and security practices represent big challenges for services and application developers because of the lack of efficient tools[5].

These challenges have been the focus of extensive research. Saurez et al. [6] present a container based distributed execution framework for edge-cloud applications that aims to optimize the task placement by using Quality of Service (QoS). Villari et al. [7] envision the concept of osmotic computing, where micro services are deployed opportunistically in both cloud and edge based on QoS requirements and current execution context. Nonetheless, these works assume that the Cloud is an always available resource and as it has discussed this is often not true in practice.

These characteristics of Fog and Edge computing make developing applications and services a complex task. Complying with established application requirements or Service Level Objectives (SLOs) is unfeasible when the application or service deployment environment is unknown:

Is there a a better way – in terms of effort as well as efficiency – to ensure application and services requirements in the complex setting of edge and fog computing?

In order to deliver high performance and requirement satisfaction rates in such a challenging environment, applications need to adapt. This adaptation involves changing the behavior of the application, which will also impact in its results.

For instance, in an application that processes a video stream for estimating the incoming highway traffic, different computer vision algorithms can be used, each of them have different accuracy and demand different levels of resources. In general, we can talk about a trade-off between accuracy and resources demanded: usually the more accurate an algorithm is, the more resources it demands and viceversa. Cameras have a low power processing node and the car counting task is performed in this processing node in order to not overload the network with images. When there are few cars on the highway, an accurate algorithm that counts each car can be performed. However, when the traffic is high, counting each car is too demanding for the low power nodes that will not be able to deliver timely updates on the traffic status. It is possible in these cases to use a less accurate algorithm that enables the processing nodes to keep up delivering timely updates, even if the accuracy is slightly relegated. With this approach, the application

adapts by using a different computer vision algorithm. Note that the different options for the computer vision algorithm should have been given by the developer, while the application itself must choose the most appropriate alternative for the current context during runtime.

Choosing the best behavior for a given context it is a complex task, because it is hard to anticipate the scenarios that the application might encounter and it is also difficult to assess the impact that each behavior change will have on its results. However, an AI-based approach might offer a good solution to this problem: several AI systems have been implemented when it is not possible to manually determine rules for the application to take decisions. An example of this is spam detection, where AI-based spam filters have been able to keep up with the changing spam techniques while traditional rule-based spam filters have failed to do so[8].

1.3 Hypotheses

The hypotheses of this work are that:

- A framework to implement adaptive applications and services provides an efficient solution to the complex challenges of Edge and Fog computing. This will facilitate the development tasks and provide better suited applications for the edge.
- Reinforcement learning provides an efficient methodology to find the best adaptive strategy during runtime, thus optimizing the behavior of the application to the current execution context and achieving applications and services with higher satisfaction of its requirements than what can be achieved with a pre-programmed logic.

1.4 Approach

In order to evaluate these hypotheses, an experimental computer science[9] approach is taken. A prototype of the adaptive framework is developed, along with a simulator to test the implementation of different logics to find the best adaptive strategy. In addition, an edge application use case is presented and implemented using this prototype. It is

then possible to evaluate the effectiveness of the approach throughout the simulation of the execution of the application in different contexts, to finally reach to conclusions about the aforementioned hypotheses.

1.5 Contribution

The aim of this work is to provide a better, more efficient way to ensure applications requirements and SLOs in Edge and Fog computing. In an attempt to achieve this, different artifacts have been developed and combined in order to build a framework that allows developers to create self-adaptive applications with little effort.

The contributions of this thesis can be summarized as:

1. Formulation of a general definition for expressing and optimizing the application objective while respecting application requirements and constraints.

2. Implementation of an application profiler and simulator, to test applications and adaptation mechanisms on different devices and registering its results in a simple and straight forward manner, which is freely available to the public.

3. A prototype of a logic that automatically adapts applications by using reinforcement learning to find the best adaptive strategy during runtime and its evaluation.

1.6 Outline

The rest of this thesis is organized as follows:

Firstly, several research works are reviewed to give the reader a background about the technologies used. Secondly, the challenges faced in the development of Edge and Fog applications is analyzed more in depth. Then, a new formal general method to define the problem is introduced, followed by the design of a framework to build adaptive applications and its implementation. Subsequently, this framework is tested with the implementation of a use case, results are displayed and discussed. Finally, related work is considered and conclusions are reached.

Chapter 2

Background

2.1 Machine Learning

There are different algorithms in machine learning which can be classified according to several aspects. One of the most common classifications used is according to their learning style. Here it is possible to classify the approaches in Supervised Learning, Unsupervised Learning and Reinforcement Learning.

Most of the machine learning systems use Supervised Learning. The algorithm is expected to classify elements according to their features. To do so in Supervised Learning, the algorithm must learn from a (usually large) number of available samples. These samples contain information about the features of each element and the classification they belong to. The challenge for the algorithm then, is to learn from the samples and generalize its knowledge so that when it is faced with a new element, with an unknown classification, it can "guess" the proper classification for the element.

It is called Supervised Learning because the training process of using already existing samples can be seen as the supervision of a teacher for the learning process. The algorithm iteratively makes predictions on the already known samples and is corrected by the teacher. The learning process stops when an acceptable performance is achieved.

Unsupervised Learning is not as widespread and frequently used as Supervised Learning. In Unsupervised Learning there are no samples to learn from. Instead, the algorithm is expected to model the underlying structure or distribution in the data, to be able to classify the elements.

It is called Unsupervised Learning because there are no correct (or incorrect) answers and there is no teacher to supervise the learning process.

Reinforcement Learning, on the other hand, combines concepts of Unsupervised Learning, by giving the algorithm the capability to determine what the correct answer it is, with concepts from Supervised Learning, by giving a clear goal to the algorithm, or agent, so that each decision has a positive or negative feedback.

Particularly, Reinforcement Learning is the chosen machine learning technique to develop this thesis' hypothesis, and because of this, its concepts are examined in detail in the next section.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a machine learning technique in which the agent – or learner – learns what actions to take based on the situation it encounters, with a determined goal. For each action the agent performs, it receives a numerical reward signal.

The distinguishing feature of RL is that the learning is perform by using information that evaluates the actions once they have been taken, instead of learning from a set of correct actions, such as the samples used in supervised learning. Purely evaluative feedback indicates how good or bad the taken decision is, but does not indicate if the action is the best or the worst. Because of this, the agent needs to employ active exploration, in a trial-and-error fashion to find a good behavior.

RL frames the problem of interactive learning, in which an agent is the learner and decision-maker, that interacts with its environment. The environment is considered as anything that cannot be changed arbitrarily by the agent and is considered to be outside of it. The interaction between agent and environment is continuous; the agent selects and performs an action and the environment responds to this action, presenting a new situation, or state, to the agent. The environment also delivers a reward to the agent, that the agent tries to maximize over time.

Moreover, the agent and environment interact with each other at discrete time steps, $t = 0, 1, 2, ...$ At each step the agent observes the environments states, $s_t \in S$, where S is the set of possible states, and on basis of its observations, selects an action, $a_t \in A(s_t)$,

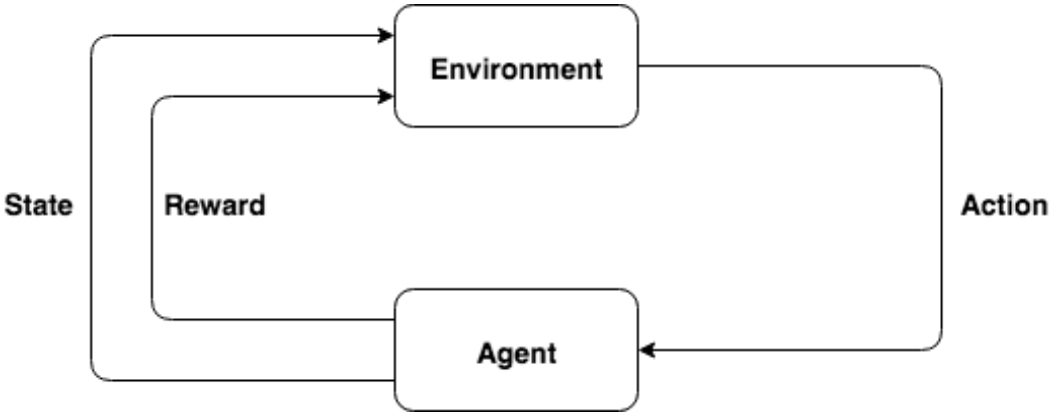

Figure 2.1 The Reinforcement Framework

where A is the set of all possible actions in state s_t. In the next step, the agent receives the reward $r_{t+1} \in R$, and finds itself in the state s_{t+1}.

To select the action that should be taken in each state, the agent implements a mapping from states to probabilities of selection each action. This mapping is called the agent's policy, π_t, where $\pi_t(a|s)$ is the probability that action $a_t = a$ if $s_t = s$. There exist different RL methods that specify how the agent changes its policy as a result of its experience. The goal of the agent is to maximize the sum of the rewards obtained over the long run.

2.2.1 Goals and Rewards

The reward is a signal fed to the agent by the environment, and it is used to formalize the purpose of the agent. At each time step, the reward is a simple number $r_t \in R$. The agent's goal is to maximize the total accumulated reward, this means that the agent should not try to maximize the reward at every step, but in the long run. It is possible to define different goals by using different reward functions, this is a task which is highly dependent on the problem that needs to be solved and has proved to be flexible and widely applicable. [10]. For instance, for an agent that plays chess, his goal is to win the match, so the reward from each movement might +1 if it wins the match and -1 if it looses or 0 if the movement is a nonterminal one.

The reward is always considered to calculated and delivered by the environment because it defines the progress in the task that the agent must perform, and it has to be beyond its reach to arbitrarily change it. The reward signal tells the agent what to

achieve, but not how. Because of this, the agent needs to explore the different actions in each state to find out what actions are yield a positive reward.

It has been said that the agent's goal is to maximize the reward over the long run. If the sequence of rewards received after time step t is denoted as $r_{t+1} + r_{t+2} + r_{t+3} + ...$, the goal is to maximize the expected return G_t, defined as some specific function of the reward sequence. In the simplest case, this is the sum of the rewards:

$$G_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T \tag{2.2.1}$$

where T is the final step. This makes sense when there is a natural notion of final time step and the agent-interaction can be broken into subsequences, called episodes. An episode can represent a chess match for the previous example. In cases in which there is no natural breaks and the interaction goes on continually, a slightly different approach is needed because otherwise the sum of rewards will be infinite. The different approach includes the concept of discounting. The agent then, selects actions in order to maximize the sum of the discounted rewards. The discounted return is expressed mathematically as:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2.2.2}$$

where γ is a parameter, $0 \leq \gamma \leq 1$, called the discount rate. By assigning values closer to 1 to γ, the agent values the future steps more, giving more importance to the long run, while if this value is closer to 0, then the agent becomes more shortsighted.

2.2.2 Markov Decision Processes

A stochastic process has the Markov property if the conditional probability distribution of future states of the process (conditional on both past and present states) depends only upon the present state, not on the sequence of events that preceded it[11].

The agent makes its decision based on the environment state, the state is whatever information that is available to the agent. A state signal that retains all the relevant information is said to be Markov, or to exhibit the Markov property, because it contains all the information that really matters for the future step. The Markov property is important in RL because decisions and values are assumed to be a function only of the current state.

A RL environment that satisfies the Markov property is called a Markov Decision Process (MDP). Given any state and action, s and a, the probability of each possible next state, s', is defined as:

$$Pr(s'|s,a) = Pr\{S_{t+1} = s'|S_t = s, A_t = a\}. \tag{2.2.3}$$

Similarly, the expected value of the next reward is:

$$R(s,a,s') = \mathbb{E}[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s']. \tag{2.2.4}$$

2.2.3 Value Functions

In order to maximize the accumulated reward, the agent needs to know how "good" it is to be in a given state and take a given action in terms of expected future rewards. For this, the agent estimates value functions, or functions of state-action pairs. The value of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter. For MDPs, $v_\pi(s)$ is formally defined as

$$v_\pi(s) = \mathbb{E}[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|S_t = s\right], \tag{2.2.5}$$

where $\mathbb{E}_\pi[\cdot]$ denotes the expected value given that the agent follows policy π and t is any time step. Similarly, the value of taking action a in state s under a policy π, denoted $q_\pi(s,a)$, as the expected return starting from s, taking action a and thereafter following policy π is defined as:

$$q_\pi(s,a) = \mathbb{E}[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|S_t = s, A_t = a\right]. \tag{2.2.6}$$

The action-value function for policy π is indicated as q_π. The value functions v_π and q_π can be estimated from experience.

2.2.4 Optimality and Approximations

The goal of solving a RL task is to find a policy that achieves a large reward over the long run. A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states, i.e. $\pi \geq \pi'$ if and only if

$v_\pi(s) \geq v_{\pi'}(s) \; \forall \; s \in S$. There is always an optimal policy π_*, i.e. a policy that is better or equal than all other policies.

Finding the optimal policy in practice is very hard, if not impossible due to computational constraints. However, it is possible to approximate optimal policies with different techniques. In tasks with small, finite state sets, it is possible to use tables with one entry for each state or state-action pair. This is called tabular methods. In many practical cases this is not possible because the state space is too large, and then it is necessary to use some sort of more compact parameterized function representation.

2.3 Edge and Fog Computing

The Internet of Things (IoT) was first coined in 1999 in reference to automated supply-chain management[5]. Since then, the concept has been used in several fields such as health care, domotic, environmental engineering, transportation and safety. The idea behind IoT is to connect physical items to the virtual world, so they can be controlled remotely and act as physical access points to Internet services[12].

Nowadays, IoT implementation has become wide-spread and devices at the edge of the network are generating huge amounts of data that need to be processed in the Cloud[5]. Because of this, increasing data traffic and network utilization are one of the biggest challenges for network operators, and this trend is expected to continue as global internet traffic is estimated to grow nearly threefold in the next 5 years[1].

Until recently, moving computing tasks to the Cloud has been an efficient way to process data because of the high availability of computing resources in the Cloud compared to the computing power of devices in the edge. However, with data generation growing steadily while links' capacity for upload has stagnated; the network is becoming a bottleneck.

Edge Computing is a paradigm that has allowed network operators and application developers to reduce network stress and improve the responsiveness of the services provided, by allocating computational efforts closer to the consumers.

Edge Computing does not aim to replace Cloud computing, but to complement it. It is comprised of the following principles or elements[13]:

- **Proximity is in the edge**: it is more efficient to communicate and distribute information between close-by nodes than to use far-away centralized intermediaries. This same argument is used by peer-to-peer networks, which gained popularity in the 1980s and 1990s.

- **Intelligence is in the edge**: as we have said before, new devices offer great computing power in the edge and this trend will continue. This opens the way to autonomous decision-making in the edge, such as distributed crowdsensing applications or agents reacting to incoming information flows.

- **Trust is in the edge**: the data sensed and stored in the edge is generally private and sensitive. Because of this, it the control and management of these data should be assigned to the edge.

- **Control is in the edge**: the management and coordination of the application is also performed in the devices located at the edge. These devices can assign or delegate computation, synchronization or storage to other peers or to the Cloud.

- **Humans are in the edge**: to keep humans in control of their information, the processing and storage of this information should be performed close to them.

Edge Computing merges the concepts and benefits of peer-to-peer networks with cloud computing and enables the creation of hybrid architectures that take advantage of the available computing power in the edge while still using the powerful resources of the Cloud. For tasks that do not require considerable computing power and can be executed on the Edge, the response time is improved because of reduced communication delay. For more computationally intensive tasks instead, the processing can still be performed in the Cloud and the response time will be the same as in Cloud computing.

With Edge Computing, end users benefit mostly from reduced communication delay, network operators benefit from bandwidth reduction and scalability, and application and services providers profit from scalability and faster services[14].

Despite the many advantages of Edge Computing, there is still more computing power which is not being fully harnessed such as smartphones and user devices which lie even closer to end users. To take full advantage of the available computing power in the

network, another novel paradigm emerged: Fog Computing. This paradigm shares the same objectives of Edge Computing, but pushes the boundaries even further; by shifting the processing closer to the user. The term Fog is used because the fog is a cloud close to the ground.

Computing, storage and networking are the building blocks of Cloud and Fog paradigms, while "the edge of the network", implies a number of characteristics that make the Fog a non-trivial extension of the Cloud[15]. These characteristics can be listed as:

- **Edge location, location awareness and low latency**. Fog computing concepts can be traced to proposals to support endpoints with rich services at the edge of the network, such as applications with low latency requirements, e.g. gaming, video streaming, augmented reality.
- **Geographical distribution**. In contrast to the centralized architecture of the Cloud, the services and applications targeted by the Fog are widely distributed. The Fog plays an active role in delivering content and services to mobile devices. In addition, large-scale sensor networks to monitor the environment require distributed computing and storage resources that the Fog can provide.
- **Very large number of nodes**. This is a consequence of the wide geographical distribution.
- **Support for mobility**. As it has been said, Fog application will communicate directly with mobile devices, which will change its location in the network regularly. Because of this, it is necessary for Fog computing to support mobility techniques that allow to separate host identity from location identity.
- **Real-time interactions**. Fog applications usually involve real-time interactions instead of batch processing, which implicate that a fast response is needed.
- **Heterogeneity**. Fog nodes have different hardware capabilities and are deployed in a wide variety of environments. In addition, the nodes change their environment often because of their mobile nature.
- **Predominance of wireless access**. Few assumptions can be made about the link availability and characteristics.

- **Interoperability and federation**. Fog components need to interoperate between them and services must be federated across domains, because the cooperation of different provides is needed to provide seamless support of services such as streaming.

- **Support for online analytics and interplay with the Cloud**. Many Fog applications gather information from several sensors and are used to provide analytics. The Cloud provides centralization for the data that needs to be shared across devices.

2.3.1 Analytics and Edge, Fog and Cloud Computing

Analytics and Big Data applications require low latency and context awareness while still having global centralization for its data. For this scenario, Fog computing offers the localization, low latency and context awareness, while the Cloud provides the required globalization.

Fog nodes collect, and might even preprocess, data in the edge of the network, generated by a plethora of sensors and devices. Some of these data might relate to tasks that need real-time processing capabilities, such as control loops, and might be deployed in Fog nodes. Other data instead, might not need real-time processing and thus can be processed in the Cloud.

In this way, the tier of Fog nodes collects, processes the data and issues control command to the actuators. It also filters the data to be consumed locally, and sends the data that higher tiers will process. The Cloud deals with visualization and reporting, as well as systems and processes.

Chapter 3

Gap Analysis of Current Applications and Services at the Edge

3.1 Implementing applications and services for Edge and Fog Computing

Even though Edge and Fog Computing bring many benefits to end users, network operators and developers, their implementation is more challenging because of their more complex structure. Some of these challenges are:

Hardware Heterogeneity. Edge and Fog applications and services must run on different devices, in a wide range of contexts. This might include low power devices, traditional x86–64 machines, high-performance servers equipped with powerful GPUs, or highly optimized application-specific integrated circuits (e.g., Google's tensorflow processing unit [16]). Because of this, applications and services might provide very different performance characteristics according to where they are deployed. In addition, the connectivity of the devices might be cabled or more often wireless, so no assumptions should be made upon the availability of the connection, its bandwidth, speed nor latency.

Mobility. Edge computing advocates that services run on close-by resources. As it has been mentioned in the previous item, the connectivity of the devices (e.g., autonomous cars, smartphones) in the edge is usually wireless, which means the connectivity to edge nodes is intermittent or highly variable. This is in contrast to highly con-

nected and centralized cloud data centers, where communication and data movement is relatively cheap and reliable. This requires distributed applications to be deployed taking into account scenarios with low and no connectivity in order to avoid failing to provide the expected service in these scenarios.

Elasticity. The cloud provides elasticity mechanisms, that the edge cannot provide [4]. Services in the cloud scale horizontally over multiple machines or vertically by increasing their RAM and CPU share. The edge might consist of a single machine with limited hardware capabilities. To make matters worse, edge load is highly dynamic (e.g., determined by the number of close-by users), which results in further limiting of available resources.

Programmability: to take advantage of Fog and Edge computing, developers must partition the functions of their applications between the Fog, the Edge and the Cloud. This is generally done manually and it needs to be carefully tuned because of the trade-off between computing capabilities and latencies to the servers, which vary for each different scenario, and even varies during runtime. This approach is not scalable or extensible[5].

Privacy and Security: the data sensed in the edge is usually private and sensitive. For instance, in the case of a smart-home system, a hacker could learn a lot of knowledge from reading the data from its users, such as when is the best time to burglar a house because there is nobody at home. Privacy and security practices represent big challenges for services and application developers because of the lack of efficient tools[5].

These characteristics of Fog and Edge computing make developing applications and services a complex task. Complying with established application requirements or Service Level Objectives (SLOs) is unfeasible when the application or service deployment environment is unknown.

3.2 An application use case

In order to better comprehend the challenges of developing applications and services for Edge and Fog computing, a practical use case is introduced and analyzed.

Modern cities contain a wide range of cameras that generate data which is used for

surveillance. Edge and fog computing have enabled the development of applications that exploit these data, as well as the distributed processing power available in smart cities. These applications make use of the video feed for different purposes, e.g., to count the people in particular areas and understand crowd mobility [17], or also to locate specific persons, like a lost or abducted child [3]. We consider this **"lost child use case"** as an application scenario.

The application works in the following way: when a child goes missing, law enforcement asks their parents for photographs of the child and feed these images to the application. Then, a service is deployed using existing connected cameras and edge servers in the city, to analyze images – captured by the cameras – and locate the child by using face recognition. When a match is found, a notification is sent to nearby law enforcement officers. In this way, we can split the application in two: an offline module; which is trained with pictures of the child, most likely in a server, and the online module; which is deployed in several devices and is in charge of finding the child. We will focus our attention on the latter.

The online module, as seen in figure 3.1 is composed by the following steps, translated as functions on the application:

1. **Capture image**: the image is captured by the camera.

2. **Preprocess image**: different preprocessing steps such as resizing, colorization, etc., are performed to improve the accuracy of the face detection and recognition.

3. **Extract faces**: a face detection classifier is applied to the preprocessed image and each face is extracted and returned in an array.

4. **Match face**: for each face that was found on the image, the previously trained classifier that matches the face of the lost child is applied. This function returns true if the face matches and false otherwise.

5. **Alert law enforcement**: a notification is sent to nearby police officers indicating the location of the child.

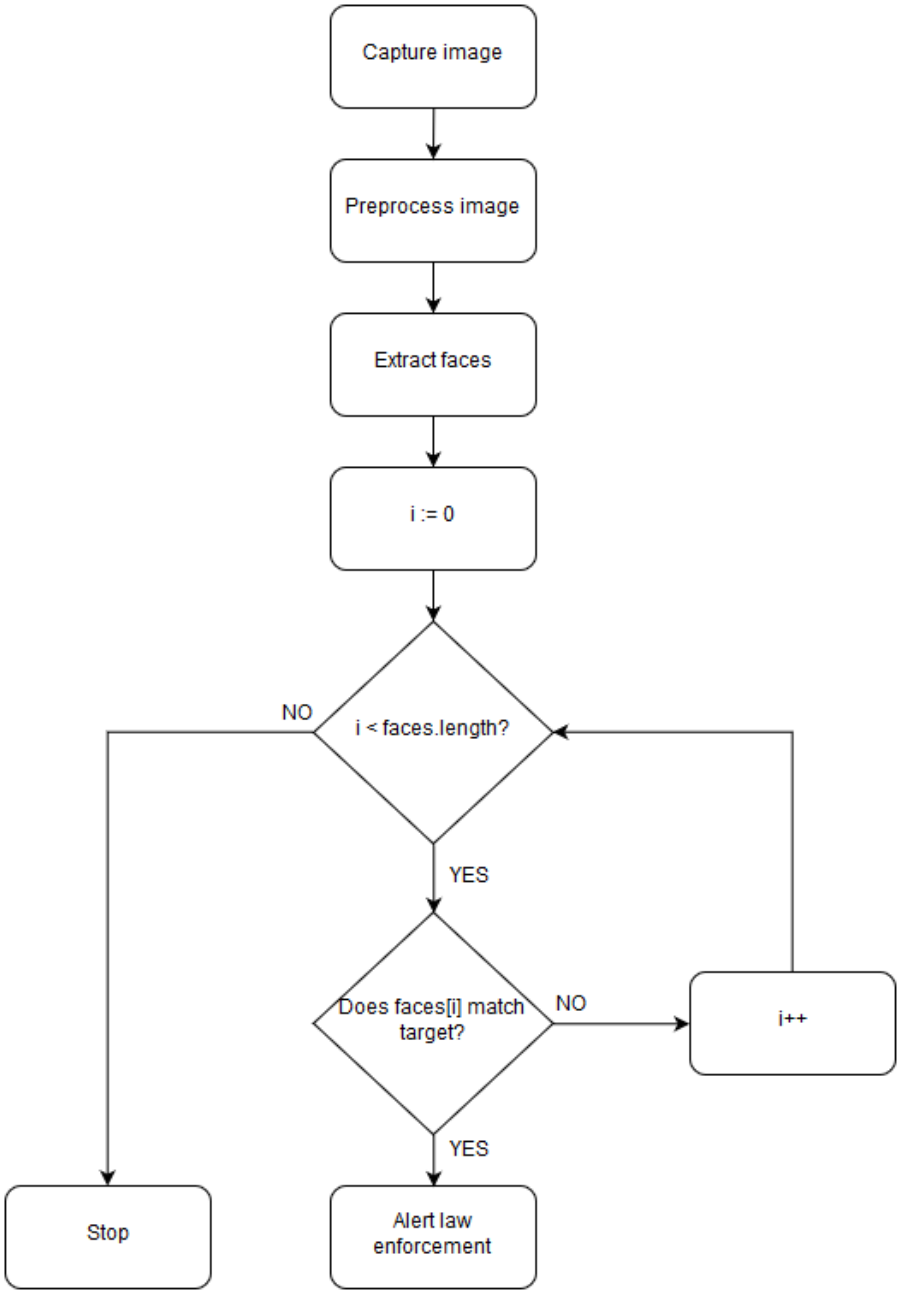

Figure 3.1 Control flow diagram of the online module for the Lost Child application

3.2.1 Application requirements

A small end-to-end latency is crucial to ensure a high frame sampling rate, so the event of missing the child is unlikely to happen, even if it only appears briefly on the video feed, e.g., when the child is moving. Because of this, a Service Level Objective (SLO) is set to having a minimum rate of one frame analyzed per second, which is translated as having a maximum end-to-end latency of 1 s for each analyzed frame. At the same time,

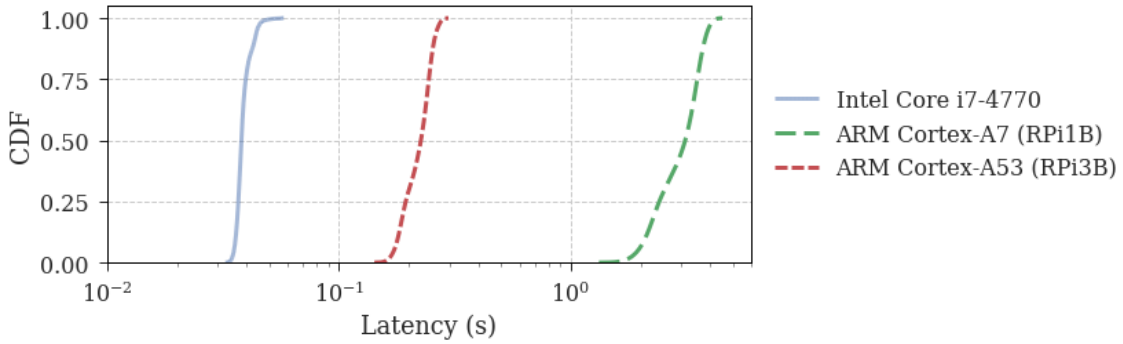

Figure 3.2 Hardware heterogeneity. Comparison of running times for Haar Face Detection + LBPH Face Recognition applied to different 640x480 PNG images with 1-6 faces running on different devices.

it is desired that the application performs with the highest possible accuracy for face detection and face matching.

More accurate algorithms are computationally intensive, causing them to be slow, while faster algorithms sacrifice accuracy in order to have a lower latency. This creates an accuracy-latency tradeoff, in which the optimal tradeoff that the application developer wants to find, is the one that enables keeping the end-to-end latency under the $1\,\mathrm{s}$ target while getting the highest accuracy possible.

However, the optimal accuracy-latency tradeoff can vary. Existing cameras are connected in different manners to upper layers (wireless, Ethernet) and also have heterogeneous hardware capabilities [18]. What is more, there are no guarantees that the application will have dedicated resources, so the processing capabilities of the devices might be even further constrained and vary over time, e.g., when other applications run concurrently on the same device.

In addition, the optimal tradeoff does not only depend on the execution context in which the application is deployed. While some cameras might be deployed in small venues with few people, other cameras might be located in highly crowded areas. This input data variability radically affects the performance of the application: processing facial recognition on 2 faces is very different than performing it on 100 faces.

3.3 Attempting to ensure requirements

In this setting, it is very hard – if not impossible – for the developer to take into account all of the different scenarios in which its application will run and therefore ensuring the application requirements is a very challenging task. As it has been said in 2.3.1 developers usually partition and tune the implementation of their applications manually, which involves a great deal of effort and is not effective in practice. In doing this, the developer can choose different paths.

Different implementations of the application can be tested in an offline manner, and the most likely implementation to comply with requirements in different devices can be used in the final product. This path leads to undesired results: it is very hard – if not impossible – for developers to take into account all of the different devices that their application will be deployed on, and so for power-constrained devices requirements might still fail to be achieved, while in more powerful devices, the application's performance could be improved by making better use of its capabilities.

If the developer has direct contact with the application stakeholders, the application can be customized for running under each different configuration or for each specific user. This involves an enormous effort, and in most cases, it is simply not possible due to time and cost limitations.

One last option, is that the developer includes some kind of adaptive logic in the application, so it will adapt to the context in which it will be deployed. This might be effective, but demands more knowledge and effort from the developer. Moreover, the complexity of this task moves the developer's focus away from the application logic that contributes to the business goals.

3.4 Adaptation mechanisms

In order to overcome the hardware heterogeneity and the wide range of contexts in which the application can be deployed, it is necessary to change the application behavior by using adaptation mechanisms. These adaptation mechanisms depend on the specific application or service and its purpose. However, there are some strategies that can be generalized for most applications:

1. **Different implementations of the same function**: this is probably the most radical option and it means changing the implementation of an algorithm in order to optimize the performance with respect to some constraints. As it has been shown by [19], different implementations of the same functionality might be used in different runtime scenarios according to the available resources. If the available memory of the system is low, an implementation which prioritizes low memory usage while sacrificing more processing power will have a better performance than a different implementation that uses less computing power but requires more memory.

2. **Varying function parameters**: most algorithms use different parameter values that define their performance in terms of required computing resources and final results. For instance, an audio processing application will vary its performance according to the **window size** that is used for the samples to be processed. If a window size of $0.1\,\mathrm{s}$ is used to split a $30\,\mathrm{s}$ audio, 300 samples will be analyzed, whereas if the window size is set to $1\,\mathrm{s}$, only 30 samples will. This window size is a parameter that the developer can change in order to comply with different application requirements.

3. **Task placement**: in the current edge and fog computing scenario, the developer could decide to execute different parts of the application in different nodes, but this could also be decided by the platform itself.

For instance, in the Lost Child application3.2, different classifiers can be used for detecting faces (i.e., Haar Cascade, LBP) and also different classifiers for matching faces (i.e., LBPH, Fisherfaces). In addition, the preprocessing step can be modified, by resizing the captured image to different sizes (i.e., 1080p, 720p, 480p). The use of different combinations of classifiers and parameters produces different accuracies as well as different end-to-end latencies (e.g., Haar Cascade + LBPH + 1080p is more accurate but slower than LBP + Fisherfaces + 480p).

By modifying its behavior, the application can meet its requirements in spite of being deployed in different hardware platforms. For example, if the Lost Child application runs on the same configuration as above – a Raspberry Pi 1B – and its end-to-end latency

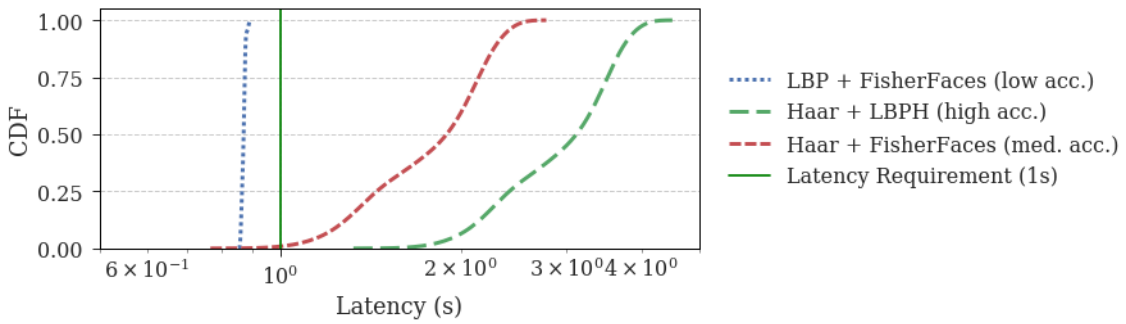

Figure 3.3 Algorithm latency variation. Comparison of running times for different combinations of Face Detection and Face Recognition algorithms applied to different 640x480 PNG images with 1-6 faces. The test was executed on a Raspberry Pi 1B device.

requirement is 1 s, the combinations of Haar + LBPH and Haar + FisherFaces will violate the application requirements as we can see in figure 3.3, so the implementation chosen should be LBP + Fisherfaces.

However, the "ideal" behavior varies according to the application deployment and the resources availability during runtime. The decision of which behavior to use must be taken during runtime, because it is impossible to anticipate all of the different configuration and scenarios in which the application will run during development. In addition, the behavior should also adapt dynamically, adjusting to the changing availability of resources the node might experiment. The way to choose the appropriate behavior will be addressed in following sections.

3.5 Conclusion

As it has been shown, there is no efficient approach to ensure application or service requirements in Edge and Fog computing. Neither of the approaches currently used provide an effective solution, and all of them incur considerable additional effort during development. This results in higher development costs and lower satisfaction levels for the consumers.

Is there a a better way – in terms of effort as well as effectiveness – to ensure application requirements and SLOs in the complex setting of Edge and Fog computing? To answer this question, a framework to program adaptive applications is proposed in the next chapter.

Chapter 4

Design and Implementation

4.1 Design

An adaptive framework is proposed to facilitate the development efforts and improve the performance of Edge and Fog applications and services. The aim of this framework is to simplify development tasks and to achieve a higher satisfaction of requirements and objectives for Edge and Fog applications and services.

4.1.1 Formally defining requirements and objectives

As it has been shown with the Lost Child use case, the application has certain requirements that need to be guaranteed to ensure its correct functioning. However, because of the high complexity of the application's running environment, it is not possible to guarantee the compliance of these requirements during runtime.

Even though in the use case presented the requirement is to have an end-to-end latency under certain target, there could be multiple requirements that should be guaranteed by the application, e.g., memory usage should be lower than 100MB, minimum quality of transmitted images should be 720p, etc. All of these requirements should be clearly expressed by defining a metric that can be used during runtime to monitor its compliance, as well as the range of values in which this metric should be. These requirements can be seen as constraints of the problem, and will be generally regarded as **costs** of the application.

The application is also expected to have a high accuracy. This can be seen as an objec-

tive instead; it is a characteristic of the application that it is desired it will be maximized (or minimized). This quantity cannot be objectively monitored by the system during runtime, but the developer can set as input mechanisms to measure it in real-time, e.g., providing values that measure the expected accuracy according to which classifier is used for face detection. In the case of the "Lost Child" the objective is the accuracy, but it could also be energy efficiency or Quality of Experience (QoE). This value can be considered as the **utility** of the application.

Moreover, the utility and the costs that the application yields are dependent on its implementation. As it has been stated in 3.4, there are a number of ways in which the developer might modify the costs and utility of the application: using different implementations for the same function or using different parameter values for the same function. In this framework, we can consider that the different implementations are an extra parameter for the function, in which different values for the parameter correspond to different implementations, e.g., 'lbp' corresponds to LBP and 'haar' corresponds to Haar cascade for a face detection function.

By using this setting, it is possible to frame the problem as an optimization problem in the following way:

$$\begin{aligned} &\underset{\theta}{\text{maximize}} && U(\theta) \\ &\text{subject to} && c_i(\theta) \leq C_i,\ i = 1, \ldots, N \end{aligned}$$

where:

θ : is the combination of parameters used for all of the functions by which the application is composed

$U(\theta)$: represents the utility of the application, which is determined by the combination of parameters used

$c_i(\theta)$: is a constraint to the function (such as latency), also determined by θ

C_i : is the constraint target (e.g., $1\,\mathrm{s}$)

N : is the total number of constraints.

By using this expression, it is possible to frame different applications with diverse requirements. For instance, in the specific case of the "Lost Child" application, its requirements and objectives can be defined as:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & accuracy(\theta_{ImagePreprocessing}, \theta_{FaceDetection}, \theta_{FaceRecognition}) \\ \text{subject to} \quad & latency(\theta_{ImagePreprocessing}, \theta_{FaceDetection}, \theta_{FaceRecognition}) \leq 1\,\mathrm{s} \end{aligned}$$

Approach assumptions

Note that with this expression, it is possible to define as many constraints as necessary, but only one utility objective. The utility represents a quantity that cannot be measured directly by the system and the decision of what it represents and how it is measured its a decision that the developer must take depending on the specific application that is being developed.

The application utility that results from the use of a combination of parameters, is calculated as a linear combination of the utility values assigned to each individual parameter value by the developer. As an example, for the face detection function the parameter "classifier" can assume the values 'lbp' or 'haar'. The utility value for each value represents how accurate this classifier is, e.g., 'lbp': 5, 'haar': 10. The actual utility value is determined by the developer and can be estimated in different ways, such as offline profiling for accuracy as described in [20].

From now on, the utility value for any combination of parameters is assumed to be known or easily computed. While the values for the costs (e.g., end-to-end latency, RAM usage) can be measured and monitored during runtime.

4.1.2 Adaptive Logic (AL)

Finding the best combination of adaptation mechanisms, or adaptation strategy, is a complex task. The best combination varies from deployment to deployment, and to make matter worse, it also varies during runtime according to the variability of available resources. Because of this, the decision of which behavior should be used at any given time, must be taken during runtime.

This is the task of the Adaptive Logic (AL). The AL must be able to combine the different adaptation mechanisms in order to achieve the highest utility possible while satisfying the application requirements or SLOs.

The implementation of the AL is challenging because of its two main requirements:

- **Rapid response.** It must adjust its behavior rapidly to keep up with resources availability and application input as they change during runtime. A slow response to changes might mean the violation of application requirements or SLOs, if the resources are further constrained, or the loss of improvement in utility, if more resources become available.

- **Low overhead.** Another important consideration for the AL is that it must not create too much overhead for the system in order to be able to perform in very low power devices.

Discrete Optimization

In optimization problems, the aim is to find solutions which are optimal or near-optimal to an objective while respecting certain constraints[21, p. 7]. Discrete Optimization (DO) works with optimization problems in which some or all of its variables assume discrete values.

Even though our problem could be formulated as a DO problem, it would force us to make some simplifying assumptions. When formulating a DO problem, it is assumed that the values for the variables involved are known, whereas in our problem this is not true, even with the assumption that it is possible to know the utility value that we want to maximize, because the cost (e.g., latency) remains unknown.

In addition, DO aims to find a single set of values that optimizes the function, while our goal is to find the best action for each different state, (i.e., what parameter should be changed if the available memory has dropped?). Because of this, DO ignores the state of the system at a given point when adaptation is necessary, missing useful information about our problem.

Bayesian Optimization

One of the most common problems in machine learning is to find the best set of hyperparameters – or at least a “good enough set" – for a model. This is an important task, because the performance of the model greatly depends on the set of hyperparameters. Unfortunately, finding the optimal set of values can also be overwhelming; usually the set of hyperparameters is not small, and trying each set of values for the model, such as a Neural Network, might take hours or even days.

Bayesian Optimization (BO) has emerged as an automatic approach to optimize the performance of a model by finding a “good" set of hyperparameters in an intelligent manner. BO works in a “black-box" manner: it makes the assumption that the unknown function that needs to be optimized was sampled from a Gaussian process, and maintains a posterior distribution for this function as observations are made[22]. In the setting of model optimization, this means that after every training with a new set of values for the model hyperparameters, the posterior distribution is updated, the most likely values for a new maximum are calculated, and the model is trained again. BO has been shown to find the optimal hyperparameter values for many black-box functions in an efficient manner [22]–[24].

However, BO is restricted to solve problems of moderate dimension[25]. When developing adaptable applications, the number of adaptation options can be very large: each function can have several implementations and each implementation can have several parameters that need to be tuned in order to find the best performance.

What is more, just like DO, BO aims to find a single set of values to optimize the objective function. Again, this means that BO is missing useful information to solve our problem.

Heuristics

Another approach for defining the AL is to create a logic using heuristics. This logic is based on the assumption of a linear tradeoff between utility and constraints, therefore it might not be universally applicable. However, it is implemented to provide a baseline to compare the results of the other AL approaches.

Before starting to run the pipeline, all of the different possible configurations of

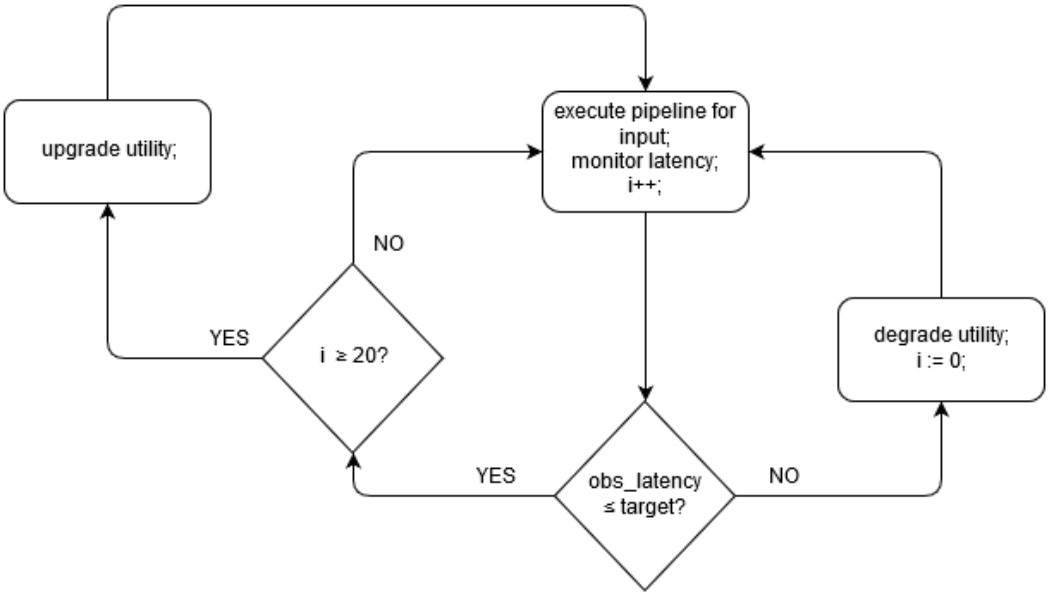

Figure 4.1 Control flow diagram of the heuristics based approach for the Adaptive Logic

adaptation mechanisms are made and sorted by utility value in descending order. It is assumed that the configuration with the highest utility is the one with the highest demand of resources.

When the pipeline starts, the configuration with the highest utility is used first. The logic monitors the application requirement(s), in this case the latency; if the latency is above the target, then utility is degraded by using the immediate lower configuration, this is expected to improve the latency. This process is repeated until the latency is less or equal than the targeted latency. If the latency is below the target for a number of continuous steps, i.e. of fully processed images, then the utility is upgraded. If this upgrade still works within the latency target, then the utility is upgraded again until the latency is beyond the target, and then the utility is degraded once by utilizing the last configuration that worked within the latency target. Figure 4.1 shows a control flow diagram for this logic.

The duration of this period of continuous steps is a parameter of the model and it was adjusted to 20 because it proved to be a good value in practice.

Reinforcement Learning

As described in section 2.2, RL offers a good framework to solve our problem. The agent represents the application and its environment can be seen as the context in which the application is running. In this setting, the agent can take actions to adapt to different states of its context, in order to achieve its goal of performing with the best possible utility under the given constraints.

In comparison with DO and BO, RL can deal with problems when some of the variable values are unknown, by exploring these values and then taking decisions with its experience. In addition, RL is designed to take into account the state of the agent, which gives it the ability of taking better decisions depending on the current state. This way, all of the available information is used unlike in DO and BO.

In order for the RL approach to be computationally inexpensive, as described in the AL requirements' in 4.1.2, tabular Q-learning was used. Tabular Q-learning defines what action should be taken in each state by maintaining a table in which there is one row for each state, and one column for each action. The value in each cell is the expected reward of being in a given state, and taking a specific action.

Before learning begins, the table Q is initialized to arbitrary fixed values. Then, at each time step t, the agent selects an action a_t, receives the reward r_t and the environment changes its state to s_{t+1}. After this, the Q table is updated by using the following update rule:

$$Q^{new}(s_t, a_t) = (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a)) \tag{4.1.1}$$

where r_t is the reward observed for the current state s_t and α is the learning rate ($0 < \alpha < 1$). γ is the discount factor and has the effect of valuing the rewards received earlier higher than the rewards that are expected to be received in future steps. This is known as delayed reward, and thanks to it the agent take into account how each action will affect the environment's state, and how much reward it is possible to get in the next state.

When the agent has just started, actions are chosen at random from each state, this is called exploration. After this process has be performed a number of times, the agent starts exploiting the Q table. To do this, when the agent is in state s_t, it chooses action

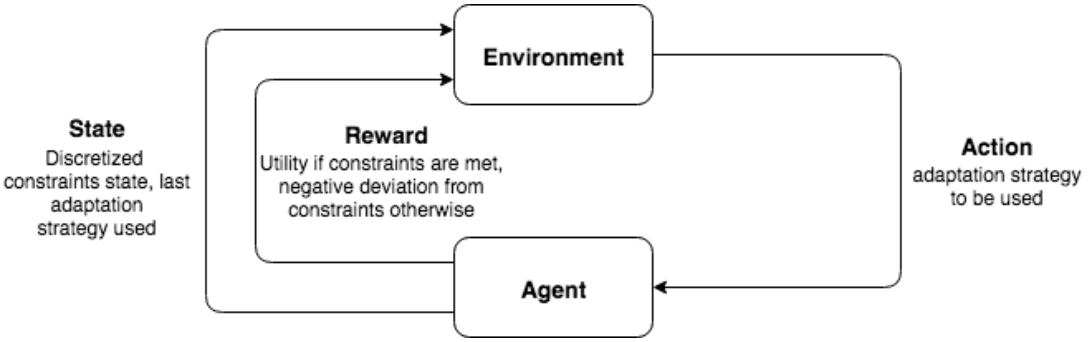

Figure 4.2 Diagram of RL approach. This approach uses the discretized last latency and configuration of adaptation mechanisms as state and the configuration of adaptation mechanisms to as action for the AL.

a_j, that has the highest expected reward by finding the cell $q_{t,j}$ with the greatest value.

Two different configurations of states and actions were defined and implemented:

Configuration 1

- States: Last latency as % of target [3 values (0-80, 80-100, 100- ∞)], current cpu availability % [3 values (0-50, 50-80, 80-100)], number of last configuration used
- Actions: Number of configuration to be used
- Dimension of π: (total configurations$\times 9$, total configurations)

Configuration 2

- States: Last latency as % of target [3 values (0-80, 80-100, 100- ∞)], number of last configuration used
- Actions: Number of configuration to be used
- Dimension of π: (total configurations$\times 3$, total configurations)

Before starting to run the pipeline, all of the different possible configurations of adaptation mechanisms are made, but unlike for the heuristics approach the configurations do not need to be sorted by utility value. While the heuristics based approach makes the assumption that the configuration with the highest utility is the one with the highest demand of resources, the RL based approach is free of these assumption, which makes it a better choice.

In addition, an important consideration in tabular Q-learning is the number of states and actions because the dimension of the Q table is defined as $(states, actions)$. The

more states or actions, the more q values that need to be kept in memory and calculated. Because of this, in order to use the tabular Q-learning approach it is necessary to carefully choose the definitions of states and actions. In this sense, configuration 2 is more efficient than configuration 1, but in order to do so, uses less information about the environment.

4.2 Implementation

All of these concepts have been implemented in the Adaptive Applications Simulator developed in Python which is publicly available as a Github repository[26]. By using the simulator, it is possible to define a pipeline and its inputs and experiment with different adaptation mechanisms and ALs.

Any developer can use the simulator by following the next steps:

1. The developer defines the pipeline which will process the inputs as a collection of functions and implements the functions normally.

2. The simulator functions (objects) are declared, along with the adaptation mechanisms.

3. The inputs for the pipeline are defined, using the provided IO objects.

4. The pipeline is profiled on the real device that will be simulated, by using the Profiler. The Profiler automatically tests all of the different combinations of adaptation mechanisms for each input and saves execution statistics that will be later used by the simulator. All of this data is saved in a cloudpickle object, called profile.

5. The Simulator is created and the profile is loaded (also multiple profiles can be loaded for multiple devices).

6. Simulations for the pipeline can be tested by indicating the set of adaptation mechanisms and device that should be used. The actual result of the pipeline, the utility and the execution statistics for each function in the pipeline, such as the latency, are returned.

In addition, a RL environment has been implemented to test the different AL implementations. The environment implements the same interface as the widely used OpenAI Gym environments[27] and uses the aforementioned simulator to provide an easy way to test different RL implementations.

4.2.1 Architecture overview

The simulator is an API that can be imported by any other Python script. The API defines a number of classes to define a pipeline with multiple functions and inputs that will be run on different devices:

- **AdasCPU**: it defines a CPU by the number of cores and speed.
- **AdasDevice**: identifies deployment devices, it has a device id, a Cpu and the memory in bytes.
- **AdasIO**: defines a functions' input/output (IO). It contains the actual input/output value (e.g. a png image) and an AdasIOData object.
- **AdasIOData**: defines the IO's metadata. It contains an id to identify the IO, the size in bytes and format (if the IO is an image, the format can by 'jpg', 'png', etc.).
- **AdasFunction**: The Function object contains the actual implementation of the function and a AdasFunctionData object.
- **AdasFunctionData**: defines the metadata for a function. Each function has a name, parameters' data and a dictionary called deployments, which contains a key for each device and the values are all of the executions' statistics for every IO and configuration of parameters. In addition, this class contains the method 'sim', used to simulate the execution of a function with the given parameters and input. A description of how parameters, in particular adaptable parameters, can be defined can be found in next section 4.2.2.
- **AdasExecutionData**: this object defines the data saved for a function execution. It contains the AdasIOData used as input for the function, the parameters' data used to execute the function, the output data that resulted from the execution as

another AdasIOData object, and the resulting statistics. The statistics are currently simply the utility and latency, but can be extended to any number of other statistics that should be monitored by the profiler and can be later used by the simulator.

The developer only needs to import the API and define the pipeline as a list of Functions, in which each Function takes the output of the previous function as input, and the inputs for the pipeline as a list of IO objects. After this, the pipeline can be used with the Profiler to profile a real device and, once the profile data is available, with the Simulator to simulate the execution of the pipeline.

4.2.2 Declaration of adaptation mechanisms

The adaptation mechanisms, i.e. adaptable parameters or different function implementations, can be specified using the AdasFunction class. All of the adaptation mechanisms are considered as adaptable parameters; in order to include a different function implementation as an adaptation mechanism, an extra parameter that defines which implementation to use can be defined, thus converting this mechanism in an adaptable parameter.

In the class AdasFunction, parameters' data is contained in a dictionary, in which the keys are the names of the parameters and the value of each item is another dictionary with one item per each possible parameter value along with its utility. In this way, a function with two adaptable parameters can be defined in the following way:

```
def detect_faces(image, scale_factor, img_resize):
    ...
    return faces

adaptive_detect_faces = AdasFunction(function = detect_faces,
                    params= {'scale_factor': {1.2:10, 1.33:4},
                             'img_resize':  {(1080,1920):10,
                                             (720,1280):8,
                                             (640,480):3}}
                    })
```

Note that each parameter has a dictionary in which the key of the item is the parameter value option and the value of the item is the utility expected for using the function

with this parameter. The total utility of the function is calculated as a sum of the value parameters used.

4.2.3 Profiler

The Profiler can be simply used in the following way:

```
from adas_profiler import profile

...

profile('device', pipeline, pipeline_inputs, n)
```

where `pipeline` is a list of Functions and `pipeline_input` is a list of IOs as defined before, and n is the number of times each parameters' configuration will be tested for each input.

The data for the pipeline, by using the AdasFunctionData, AdasIOData and Device are saved, along with the AdasExecutionData for each independent execution performed in a cloudpickle file, called 'profile'.

4.2.4 Simulator

After profiling a device, the execution of the pipeline can be simulated for any configuration of the adaptation mechanisms and any of the previously defined inputs with the following code:

```
from adas_simulator import AdasSimulator

simulator = AdasSimulator()
simulator.load_profile('profile.pkl')
simulator.sim('device', pipeline_input,
                {'img_resize':'(1080, 1920)',
                 'scale_factor':'1.2'})
```

This call returns a dictionary with the statistics for the execution of the method, the utility and the output as an IO object. The simulation returns a random sample from the execution data with the same parameter values, device and input as the one specified in the call.

4.2.5 Environment

In addition, the class Environment makes it simple to test different implementations of the AL. The implemented interface has been inspired by OpenAI Gym environments[27] and shares the definitions of the method `step` and `reset`.

Different implementations of the Environment have been defined in order to try the different configurations of states and actions, which are detailed in section 4.1.2. The initialization of the environment is made by specifying a Simulator object, a latency target and the pipeline_inputs. Each step is defined as the execution of a full pipeline for an input. For each step, the CPU availability is simulated as a Markov Chain, where the CPU availability at a given time step t, C_t, is calculated as follows:

$$C_t = \begin{cases} C_{t-1} + \theta, & \text{if } C_{t-1} + \theta > 0.3 \text{ and } C_{t-1} + \theta < 1 \\ C_{t-1}, & \text{otherwise} \end{cases} \tag{4.2.2}$$

where $\theta \sim N(0.1, 0.1)$.

The code to declare an Environment and make a step with it is the following:

```
from adas_simulator import AdasEnvironment

...

env = AdasEnvironment(simulator,
                      1.0,
                      pipeline_inputs,
                      cpu_availability)
env.reset()
new_s, r, done, inf = env.step(action)
```

This method, just like OpenAI Gym environments, returns four objects: the new state (`new_s`), the reward (`r`), a boolean indicating if the environment has finished (if all of the `pipeline_inputs` have been processed) (`done`) and a dictionary with the resulting execution stats and utility of the given step (`inf`).

4.3 Conclusion

By using the artifacts developed in this chapter, it is possible to implement applications that incorporate adaptation mechanisms. After doing this, the applications can be run on different devices registering its results and execution statistics with the profiler. Then, by loading the profiles to the simulator, different ALs can be tested and evaluated.

In the next chapter, an adaptive application is implemented by using these tools and its performance is tested in different scenarios under the Heuristics and RL based AL.

Chapter 5

Experimental Evaluation

In order to test the approach presented in the previous chapter, the use case application of the "Lost Child" is implemented using the adaptive framework. Different adaptation mechanisms are implemented for the application so it can vary its behavior during runtime.

The performance of the application in terms of its objective and its constraint is put to test during different execution conditions. In addition, the heuristics and the RL based ALs are tested to shed light on how well do the approaches fulfill its previously defined requirements.

5.1 Implementation of the Lost Child application

The Lost Child application described in section 3.2 was implemented by making use of the software developed in section 4.2 and OpenCV[28]. A Raspberry Pi 3 model B+ was used to profile (and later on simulate) the behavior of the application and its different configurations of adaptation methods. There are 16 possible different configurations, made by combining the adaptation mechanisms.

To measure the accuracy, a dataset of 500 images with a random number of between 6 and 24 faces was created, by using the collection of facial images Faces94[29]. The collection contains 20 images from 153 subjects. 5 randomly chosen subjects were used as targets to be identified and the other subjects were used to complete the rest of the faces in each image. The images were composed in a way that there is a 0.5 probability

of containing a target. From the 5 target subjects, 75 % of the images were used to train two different facial recognizers from OpenCV: FisherFaces and LBPH. The rest of the images were used to build the images in the dataset.

The whole dataset was analyzed with each of the 16 different combinations and the average accuracy, precision and recall were calculated for each. The results are shown in table 5.1.

To test the end-to-end latency, a different dataset with very different amount of faces in each images was created with 12 different images. The images used were also created using the Faces94 face collection. 6 different pairs of images contain different number of faces; one pair contains 6 faces, another pair 12 and so on for 24, 48, 96 and 192 faces. Then, the end-to-end latency was recorded 5 times for each different configuration. The average latency recorded for each number of faces and configuration is shown in figure 5.1.

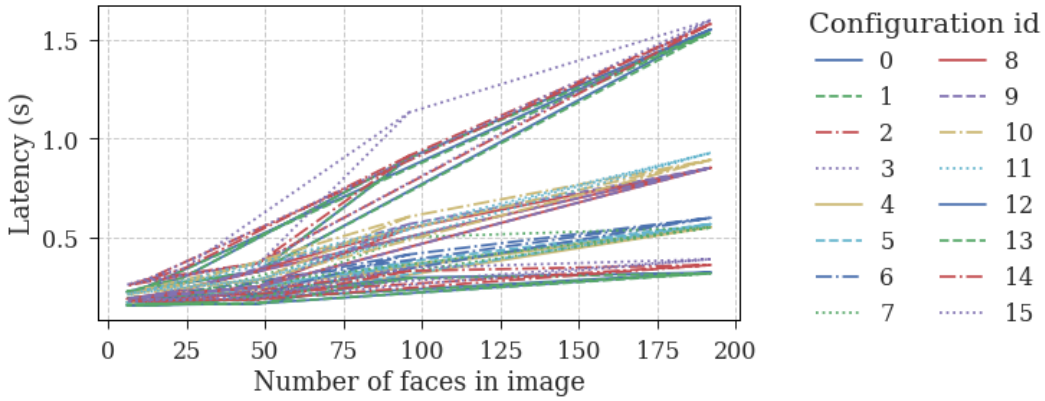

Figure 5.1 Latency variation according to input. The configurations vary their latencies differently according to the number of faces on the images. The latencies were measured on a Raspberry Pi 3 B+.

In the implementation of the application, the developer wishes to have the highest possible precision when detecting the target. The precision is defined as the number of times that the target was identified correctly divided in the number of times it appeared in the frames. Because of this, the offline profiled precision is used as the utility for the application, which represents the expected precision for each analyzed image using a given configuration, that the AL will try to maximize. As it has been said before, the constraint is to keep the end-to-end latency in a value of less than 1 s, in order to analyze

at least 1 FPS of the capture images.

5.2 Environments and Datasets

Since the application does not run in exclusive hardware, i.e., other applications are running in the same processing node, the CPU availability varies over time. As it has been explained in section 4.2.5, the environment simulates the CPU availability as a Markov Chain, where the CPU availability varies from 0.3 to 1.0.

As for the datasets, three different options were used for the simulations:

1. Fixed input: 1000 frames with 48 faces each.

2. Variable input: 1100 frames with varying number of faces in each image, from 6 faces to 192. The dataset is composed by blocks of 100 continuous frames with the same number of faces. The blocks are arranged in the following way: 6, 12, 24, 48, 96, 192, 96, 48, 24, 12, 6; where each number represents the amount of faces in the block of 100 frames.

3. Full day input: a dataset that simulates the whole input of a full day at a train station was built. There are 86400 images, one for every second of the day. Again, the inputs from the previous item were used, but the number of faces varies over time to simulate peak hours as well as little traffic hours as it can be seen in figure 5.2.

In addition, another version of the RL environment was implemented. This version models the random CPU availability as the previous one, and also incorporates random inputs. The inputs are simulated by taking random samples from the ones used by the Variable input dataset. Because the input does not change every second in reality, in each step there is a 0.1 probability of changing the input with a random sample and 0.9 probability of keeping the same input as in the previous step.

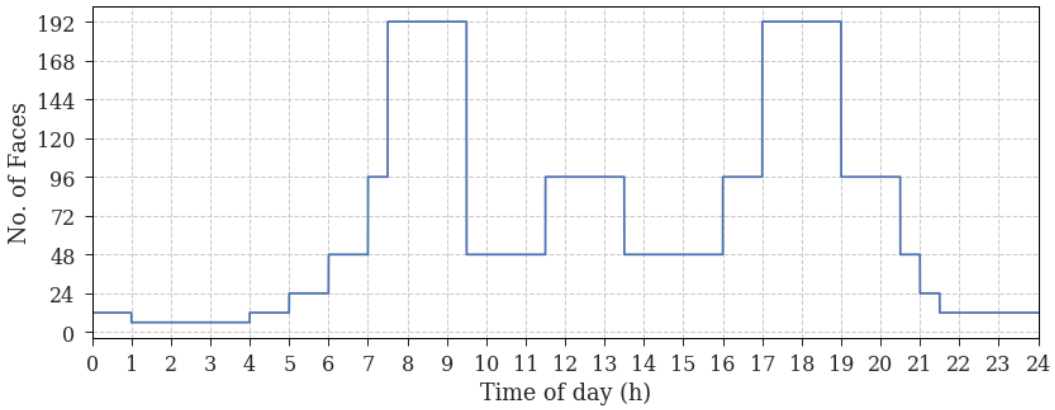

Figure 5.2 Simulation of camera input in train station. Peak hours are visible around 9 AM and 6 PM, while valleys are visible during nighttime.

5.3 Results

5.3.1 Baseline: Heuristics based AL

The baseline for the AL is the Heuristics based approach as described in section 4.1.2. Each of the three different datasets and the random input environment were used to simulate the pipeline of the application 50 times and its results were recorded in table 5.2. In addition, in figures 5.3 and 5.4, a histogram was plotted to show the utility (or expected precision) and the latency of the processed images during the 50 runs for the Fixed input dataset and the Full day dataset.

As it can be seen in table 5.2, the average utility and the latency satisfaction drop as the complexity of the processed inputs vary. While for the Fixed input 93% of the inputs are processed in $1\,\mathrm{s}$ or less, for the Full day and Random input simulations this value drops to around 80%. This is also noticeable in the histograms, while plot 5.3 shows very little inputs over the limit of $1\,\mathrm{s}$, plot 5.4 shows a longer tail after the $1\,\mathrm{s}$ mark.

With the utility a similar trend is seen; the Fixed input simulation finished with a 0.82 utility while the Full day and Random input are between 15% and 20% down, to 0.69 and 0.66 respectively.

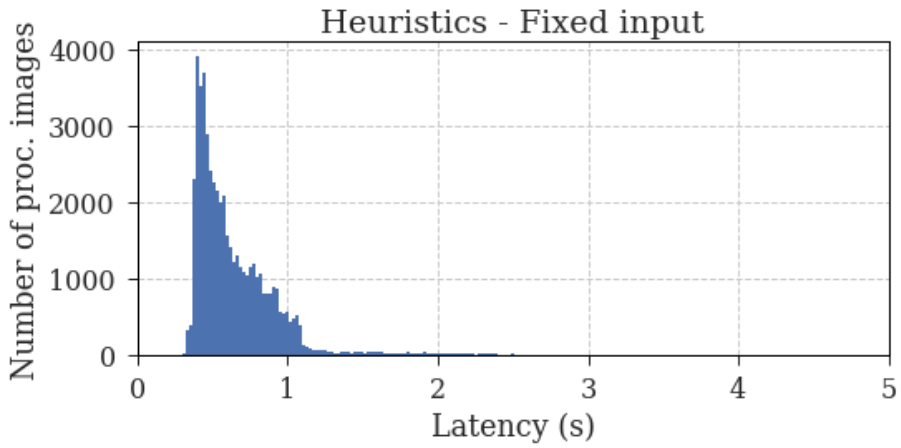

Figure 5.3 Histogram of obtained utility and latency for each processed imaged during 50 simulations using the Heuristics based AL and Fixed input dataset.

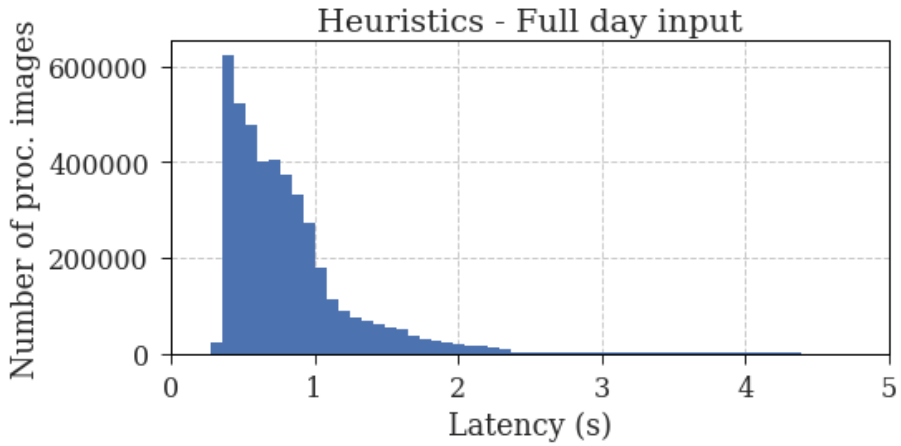

Figure 5.4 Histogram of obtained utility and latency for each processed imaged during 50 simulations using the Heuristics based AL and the Full day input dataset.

5.3.2 RL based AL

The same simulations were performed for each state-action configuration of RL approaches. Each simulation was repeated 50 times. Since the RL approach is a learning approach, the RL logic starts using random adaptation configurations and learns over time which configurations perform better in which situations. Because of this, each simulation was done using the Q-table of the previous simulation, starting from a Q-table full of zeroes. In figure 5.5 it is possible to see how the moving average, with a window of 5, of utility and latency improve over the simulations for configuration 1.

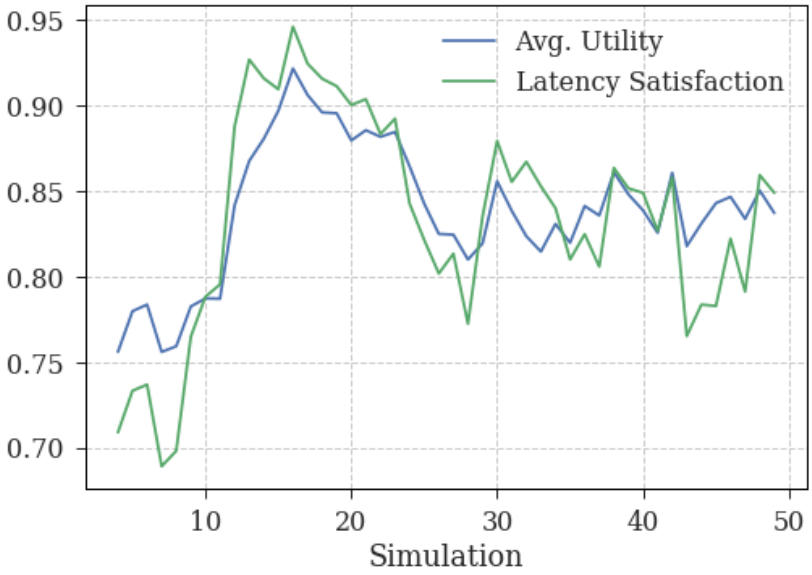

Figure 5.5 Average Utility and Latency Satisfaction during continuous simulations using the Random input environment and RL Configuration 1. The curve of the moving average with a window of 5 is shown for each metric. Learning improves results until the 10th simulation, then because of the random nature of the environment, both average utility and latency satisfaction vary while tending to stabilize around the 0.80-0.85 range.

Configuration 1

In the tests, Configuration 1 showed the best results. This configuration uses the last end-to-end latency registered, the last configuration of adaptation mechanisms and the current CPU availability to decide which configuration of adaptation mechanisms should be used next.

The results for the simulations using this configuration are shown on table 5.3. It is possible to see a similar trend to the heuristics case (table 5.2, where the average utility and latency satisfaction drop as the complexity of the inputs increases. However, with RL the Latency Satisfaction is slightly higher for all cases but Variable input, in which is slightly lower, and the average utility is always higher by a large margin of between 10% and 25%. It is noticeable that this margin increases as the complexity of the inputs increases; the improvement is close to 10% for the Fixed input dataset while it reaches a 25% improvement for the random input environment.

In histograms 5.6 and 5.7 it is possible to see how there is once more a longer tail for the Full day dataset than for the Fixed input dataset. This is again produced by the

higher complexity of the dataset and the exploration phase of the RL approach. However, the histogram shows two peaks: one around $0.4\,\mathrm{s}$ and the other one around $0.8\,\mathrm{s}$, while in the Heuristics histograms (5.6 and 5.4) there is only one peak around $0.4\,\mathrm{s}$. This is because the RL approach is able to use the adaptation mechanisms' configurations in a more optimal way, yielding the highest utility possible while respecting the constraints. This is why the average utility is higher for this approach as shown on table 5.3.

Moreover, it is interesting to see how the learning processes improves the results of average utility and latency satisfaction over time. This is shown in figure 5.5 for the Random input environment, which offers the highest variability – and complexity – of scenarios that the AL has to deal with.

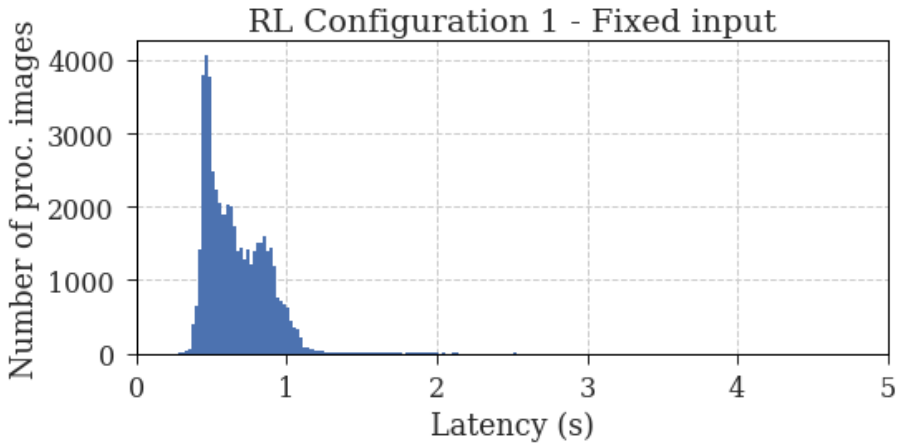

Figure 5.6 Histogram of obtained utility and latency for each processed imaged during 50 simulations using the RL Configuration 1 based AL and Fixed input dataset.

Configuration 2

Configuration 2 uses the last end-to-end latency registered and the last configuration of adaptation mechanisms to decide which configuration of adaptation mechanisms should be used next. This configuration showed better results than the Heuristics approach, but with lower average utility than Configuration 1. This is a logical outcome since this approach uses less information about the environment to take its decision.

The results for the simulations using this configuration are shown on table 5.4. A similar trend to the heuristics and configuration 1 is visible, where the average utility and latency satisfaction drop as the complexity of the inputs increases. The latency sat-

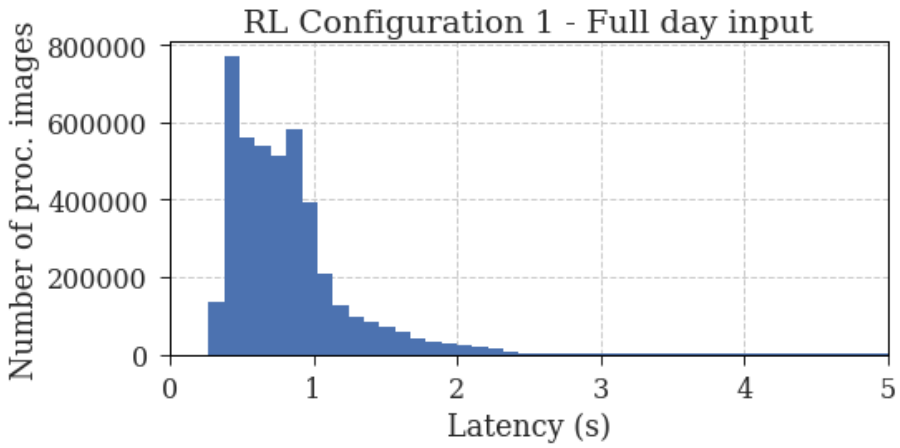

Figure 5.7 Histogram of obtained utility and latency for each processed imaged during 50 simulations using the RL Configuration 1 based AL and the Full day input dataset.

isfaction performance is similar to configuration 1, but the average utility improvement is smaller.

5.3.3 AL Requirements Fulfillment

The AL has managed to achieve a high requirement satisfaction while keeping a high utility throughout the simulations, but it is still necessary to verify it complies with the requirements for the AL, exposed in section 4.1.2:

- **Rapid response.** Plots 5.8 and 5.9 show a fragment of simulations processing the Fixed input dataset. This dataset was chosen because the number of persons in the input images is constant, so the latency varies only according the CPU availability. It is possible to see how the configuration of the application is adapted to perform under the $1\,\mathrm{s}$ latency requirement when the CPU availability varies, degrading the utility when the CPU availability is low and upgrading it when the CPU availability increases. RL offers a faster response to changes, achieving a higher requirement satisfaction as well as a higher average utility.

- **Low overhead.** Table 5.5 shows the average time it takes to make a decision each time that an image will be processed (including the update of the Q-table values in RL) and compares it to the time it takes to process the an image with 6 faces with the fastest configuration of adaptation mechanisms. As it can be seen, the time

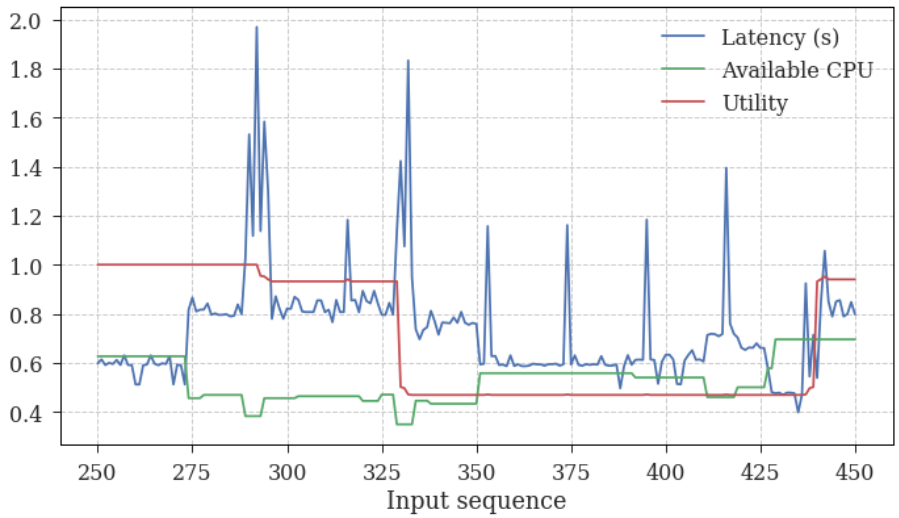

Figure 5.8 Rapid response for the Heuristics based AL. A fragment of a simulation processing the Fixed input dataset is shown. The available CPU drops around step 275 generating a higher latency that ends up being higher than the 1 s latency requirement. After step 325 the AL degrades the utility to be able to perform under the latency requirement. From step 425 the available CPU increases and after 20 steps the AL changes its configuration to achieve a higher utility taking advantage of this situation.

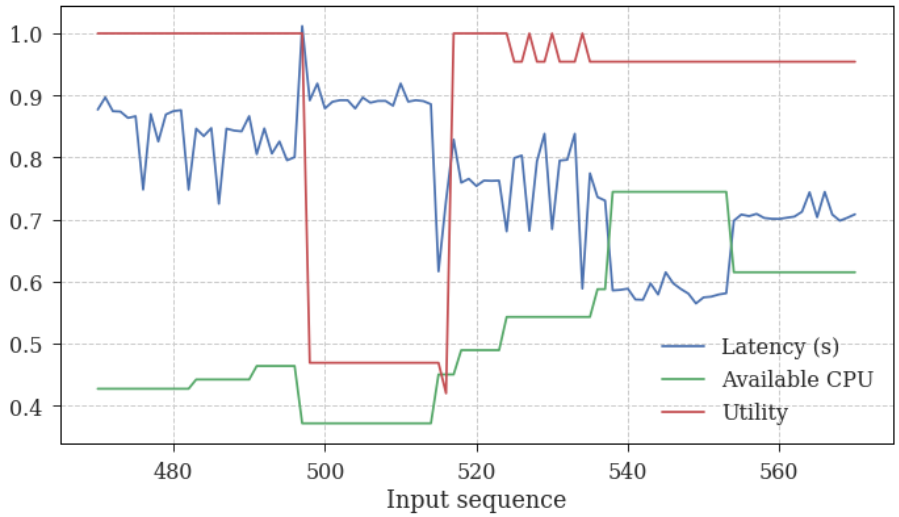

Figure 5.9 Rapid response for the RL Configuration 1 based AL. A fragment of a simulation processing the Fixed input dataset is shown. From step 495 the CPU availability drops, increasing the latency until just above 1 s, but the RL AL acts almost immediately, using a configuration of adaptation mechanisms which is a faster but also produces a lower utility. When the CPU availability increases, from step 515, the AL changes the configuration once again, almost immediately, to obtain a higher utility.

to take the decision is negligible, involving less than 0.3% of the total time in the case of RL. It is interesting to note that the RL approach produces even a lighter overhead when compared to the Heuristics one.

5.4 Conclusion

Throughout the simulation of the Lost Child application, the efficiency of an adaptive framework to develop applications for Fog and Edge computing has been shown. The developer only needs to define the functions for the application, the adaptation mechanisms, i.e. adaptive parameters and alternative implementations, and its expected utility. During runtime the AL in the framework will combine these mechanisms in configurations to provide the highest possible utility in the given execution context.

Moreover, the RL based AL improves over the Heuristics based AL as the results haven shown. The utility of the application, defined in this case as the expected precision, shows a gain between 10% and 25%; the response of the RL approach is also faster than its counterpart and finally, the overhead incurred is also smaller than the one produced by the Heuristics approach.

Configuration					Results		
	Face Detection			Face Recognition			
Id	Face Detector	Scale Factor	Face Resize Dimensions	Face Recognizer	Accuracy	Precision	Recall
0	Haar	1.2	(100,100)	Low	0.822	0.439661516	0.987055016
1	Haar	1.2	(100,100)	Medium	0.8372	0.470588235	0.98381877
2	Haar	1.2	(100,100)	High	0.992	1	0.938511327
3	Haar	1.2	(120,120)	High	0.9912	0.951781971	1
4	LBP	1.2	(100,100)	Low	0.8444	0.465599706	0.993527508
5	LBP	1.2	(100,100)	Medium	0.8588	1	0.993527508
6	LBP	1.2	(100,100)	High	0.9952	0.496049661	0.962783172
7	LBP	1.2	(120,120)	High	0.9904	0.954352442	0.993527508
8	Haar	1.33	(100,100)	Low	0.8292	0.447118891	0.987055016
9	Haar	1.33	(100,100)	Medium	0.8556	0.501331811	0.993527508
10	Haar	1.33	(100,100)	High	0.9984	1	0.987055016
11	Haar	1.33	(120,120)	High	0.9896	0.939958592	1
12	LBP	1.33	(100,100)	Low	0.8252	0.419501134	0.838727077
13	LBP	1.33	(100,100)	Medium	0.844	0.468774861	0.845199569
14	LBP	1.33	(100,100)	High	0.9764	1	0.841963323
15	LBP	1.33	(120,120)	High	0.9672	0.931818182	0.848435814

Table 5.1 Adaptive configurations for the Lost Child Application. Note: some combinations are not possible due to constraints of face recognizers training, e.g. low face recognizer with face size (120,120).

	Avg. Utility (Expected Precision)	Latency Satisfaction (%)
Fixed input	0.8271	93.02
Variable input	0.7581	82.43
Full day input	0.6999	79.20
Random input	0.6626	80.22

Table 5.2 Average utility and latency satisfaction of Heuristics based approach

	Avg. Utility (Expected Precision)	Latency Satisfaction (%)
Fixed input	0.9144	95.00
Variable input	0.8266	79.04
Full day input	0.7779	79.06
Random input	0.83455	83.03

Table 5.3 Average utility and latency satisfaction of RL Configuration 1 approach

	Avg. Utility (Expected Precision)	Latency Satisfaction (%)
Fixed input	0.9192	94.72
Variable input	0.7636	81.36
Full day input	0.6865	79.31
Random input	0.7221	80.89

Table 5.4 Average utility and latency satisfaction of RL Configuration 2 approach

		Adaptive Logic	Image Processing	Total	Impact of AL in total latency
Macbook Pro 2012	**Heuristics**	0.00029s	0.07023s	0.07052s	0.42%
	RL	0.00021s		0.07044s	0.30%
Raspberry Pi 3 B+	**Heuristics**	0.00131s*	0.31193s	0.31324s*	0.42%*
	RL	0.00093s*		0.31286s*	0.30%*

Table 5.5 Overhead of AL. The average execution time for the logic for simulations in a Macbook Pro 2012 (Intel Core i5-3210M) and compared against the average processing time of images with 6 faces using LBP for face detection and the low accuracy face recognizer (fastest adaptive configuration). (*) projected times.

Chapter 6

Discussion and Related Work

6.1 Discussion

The results in the previous section demonstrate the ability of the adaptive framework developed to adapt to different execution contexts. Even though the results obtained are satisfactory, better results can be achieved in different ways.

To start with, the current model puts the responsibility of evaluating and assigning utility values of the different adaptation configurations to the developer. This is not an easy task sometimes, especially when there are multiple adaptation mechanisms available. Different options are being considered to solve this problem, such as using a developer-provided dataset for offline profiling of the adaptive configurations or using a "golden" combination that is expected to perform the best as ground truth for evaluating the performance of the other combinations.

To continue, a characteristic of Edge and Fog computing is the migration of tasks that a processing node can perform. In this work, it has been assumed that this task allocation has already been performed by a higher layer. However, taking into account task migration as an adaptation mechanism can provide further improvement to the results of the adaptation. This increases the adaptive configuration space greatly and thus represents a challenge for efficient adaptive logic implementations.

Moreover, it has been said that in order for tabular Q-learning RL to work, a limited number of states must be taken into account and because of this developers should carefully choose the adaptation mechanisms that are defined. This again can be solved by

performing offline profiling of the adaptation configurations and choosing the best ones to take into account during runtime while disregarding the other ones.

Finally, the results presented in this work have been realized by using the Adaptive Applications Simulator. Even though the simulations were made using real data to produce realistic results, real world tests are yet to be implemented.

6.2 Related Work

There is extensive research in the related fields of this thesis such as programming abstractions and systems adaptive behavior.

Senergy[30] is a framework for programming mobile applications. It enables developers to automate common latency, power, accuracy trade-offs by letting them specify priorities among them. ENT [31] provides a type-based proactive and adaptive energy management at the application level, where developers characterize energy behavior of different program fragments with modes that are later used in runtime depending on the execution context of the application. Nonetheless, these approaches do not permit the inclusion of constraints for enforcing applications' requirements.

Natural Adaptive Video Streaming with Pensieve[32] presents a system that generates adaptive bit rate (ABR) algorithms using Reinforcement Learning. These algorithms are used for video streaming and must balance a variety of QoE goals This work successfully applies a variant of deep RL, A3C, to create algorithms that adapt to a wide range of environments and QoE. The NN model runs on the server in order to avoid the overhead on the client.

In Chameleon[33], the performance of video analytics applications is optimized by performing automatic adaptation of its configurations. The application's behavior is customized to the execution context by selecting different parameter configurations; the best parameter configuration is selected by a logic inspired by greedy hill climbing combined with periodical online profiling. However, this research is centered around applications that use deep convolutional neural networks for video analytics, while the adaptive framework developed in this thesis aims to offer a flexible solution that can be applied to any kind of application.

Chapter 7

Conclusions

Developing applications and services for Edge and Fog computing is a complex task due to the characteristics of these paradigms such as the hardware heterogeneity and the limited scalability of the processing nodes. By using the adaptive framework hereby presented, developers are able to develop efficient applications and services in a simpler way.

Firstly, defining the application's or service's requirements and objective – as described in section 4.1.1 – lets developers have a clearer view of what should be optimized during running time as well as what are the constraints that should be respected while doing so. This approach has been published as part of Nandu, a system developed with the IoT Group at NEC Laboratories that adapts and migrates tasks dynamically[20].

Secondly, specifying the adaptation mechanisms the application or service can use – as explained in sections 3.4 and 4.2.2 – enables developers to take into account several ways in which the behavior of the software can be modified to provide better performance throughout different execution environments and conditions. This has been further developed and applied to IoT services as part of another research conducted with the IoT Group at NEC Laboratories and will be published at the International Conference on Network and Service Management 2018 in Rome.

To continue, the software developed in order to profile and simulate adaptive logics and adaptation mechanisms – as laid out in section 4.2 – provides a simple way to define applications or services and experiment with different adaptation schemes to evaluate them. This software has been made publicly available[26] in order to give the community

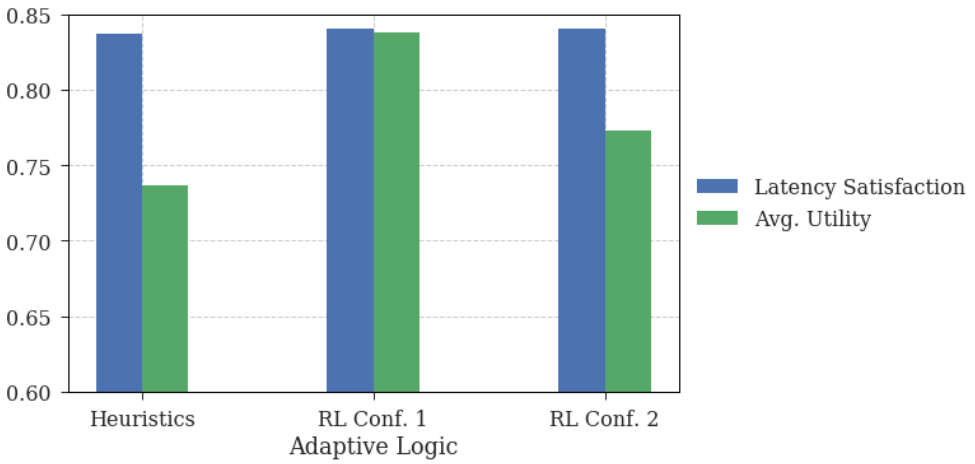

Figure 7.1 Comparison of Adaptive Logic approaches. The mean latency satisfaction and average utility of each approach for all datasets is shown. The best logic in terms of average utility and latency satisfaction is RL Configuration 1.

the opportunity of testing it and continuing the research conducted in this work.

Finally, by using RL – as conceived in section 4.1.2 –, the best configuration of adaptation methods can be found automatically during runtime without the necessity of implementing a pre-programmed logic. To sum up the results obtained in this work, figure 7.1 shows how the average utility, which is equal to the expected precision, and the end-to-end latency requirement satisfaction for the use case application, is improved by using the adaptive framework developed.

7.1 Hypotheses Evaluation

It is now possible to evaluate the hypotheses presented at the beginning of this thesis:

- The adaptive framework to implement applications and services has been developed and tested by implementing an application. The adaptive framework made its development easier and it has shown during several simulations of different execution conditions, that is able to adapt the applications behavior to deliver a high requirement satisfaction rate and performance. In addition, the AL proved to be efficient by generating a minimal overhead during runtime.

- The RL based AL has proved to be better than the pre-programmed logic in several aspects: it has achieved a higher applications' objective performance, a faster

reaction to conditions variations and a lower overhead for the application.

7.2 Future Work

The development of an adaptive framework that implements the concepts presented in this thesis is still ongoing research. As it has been discussed in section 6.1, there are still several opportunities to improve the results of the framework, while at the same time making the developers' work easier.

The implementation of methods to automatically determine the utility of each adaptive configuration takes off the responsibility of developers from assigning these values. In order to do so, two different options are being taken into account. The first one is to let developers specify a dataset (and its groundtruth) that will be used to profile the adaptive configurations in an offline manner.However, developers might not have access to real data that the application will process, and because of this the utility calculated will be an approximation. The other option, is to let developers select a "golden" adaptive configuration, which results will be used as the groundtruth during runtime to evaluate the performance of other configurations. Nonetheless, this approach needs more resources since the two configurations must be run in parallel at least a few times, and this might be very difficult to do in practice.

The inclusion of task allocation as another decision of the AL is also an opportunity to improve the performance of applications and services. This decision increases the size of adaptive configurations to take into account exponentially according to the number of processing nodes available. Because of this, deep Q-learning is being taken into consideration to implement the AL. Although deep Q-learning increases the demand of processing resources greatly, which poses a challenge to its implementation, a number of techniques have been developed to "compress" deep neural networks such as [34].

A next step is the development of multiple applications or services for Edge and Fog computing by using this adaptive framework; especially those that have different objectives and constraints such as precision, battery efficiency and cost optimization. Moreover, the implementation of these applications in the real world will give further insights about this approach's performance and hint on how to optimize it.

Bibliography

[1] C. V. N. Index, "The zettabyte era – trends and analysis, cisco," *Cisco company, June*, 2017.

[2] Y. Cao, S. Chen, P. Hou, and D. Brown, "Fast: A fog computing assisted distributed analytics system to monitor fall for stroke mitigation," in *Networking, Architecture and Storage (NAS), 2015 IEEE International Conference on*, IEEE, 2015, pp. 2–11.

[3] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, and A. Kitazawa, "Fogflow: Easy programming of iot services over cloud and edges for smart cities," *IEEE Internet of Things Journal*, 2017.

[4] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[5] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[6] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder, "Incremental deployment and migration of geo-distributed situation awareness applications in the fog," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, ACM, 2016, pp. 258–269.

[7] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic computing: A new paradigm for edge/cloud integration," *IEEE Cloud Computing*, vol. 3, no. 6, pp. 76–83, 2016.

[8] T. S. Guzella and W. M. Caminhas, "A review of machine learning approaches to spam filtering,"
Expert Systems with Applications, vol. 36, no. 7, pp. 10 206–10 222, 2009.

[9] P. J. Denning, "Acm president's letter: What is experimental computer science?"
Communications of the ACM, vol. 23, no. 10, pp. 543–544, 1980.

[10] R. S. Sutton, A. G. Barto, *et al.*, *Reinforcement learning: An introduction*.
MIT press, 1998.

[11] A. A. Markov, "The theory of algorithms,"
Trudy Matematicheskogo Instituta Imeni VA Steklova, vol. 42, pp. 3–375, 1954.

[12] F. Mattern and C. Floerkemeier,
"From the internet of computers to the internet of things,"
in *From active data management to event-based systems and more*, Springer, 2010, pp. 242–259.

[13] P. G. Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges,"

[14] M. T. Beck, M. Werner, S. Feld, and S Schimper,
"Mobile edge computing: A taxonomy,"
in *Proc. of the Sixth International Conference on Advances in Future Internet*, Citeseer, 2014, pp. 48–55.

[15] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli,
"Fog computing and its role in the internet of things,"
in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, ACM, 2012, pp. 13–16.

[16] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*,
"In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ACM, 2017, pp. 1–12.

[17] F.-J. Wu and G. Solmaz, "Crowdestimator: Approximating crowd sizes with multi-modal data for internet-of-things services," in *MobiSys'18*, 2018.

[18] T. Zhang, A. Chowdhery, P. V. Bahl, K. Jamieson, and S. Banerjee, "The design and implementation of a wireless video surveillance system," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, ACM, 2015, pp. 426–438.

[19] E. Poormohammady, J. H. Reelfs, M. Stoffers, K. Wehrle, and A. Papageorgiou, "Dynamic algorithm selection for the logic of tasks in iot stream processing systems," in *Network and Service Management (CNSM), 2017 13th International Conference on*, IEEE, 2017, pp. 1–5.

[20] J. Fürst, M. Fadel Argerich, K. Chen, and E. Kovacs, "Towards adaptive actors for scalable iot applications at the edge," *Open Journal of Internet Of Things (OJIOT)*, vol. 4, no. 1, pp. 70–86, 2018.

[21] F. Rothlauf, *Design of modern heuristics: principles and application*. Springer Science & Business Media, 2011.

[22] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[23] A. D. Bull, "Convergence rates of efficient global optimization algorithms," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2879–2904, 2011.

[24] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," *arXiv preprint arXiv:0912.3995*, 2009.

[25] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[26] M. Fadel Argerich, *Adas: Adaptive applications simulator*, `https://github.com/maufadel/AdAS`, 2018.

[27] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[28] G. Bradski and A. Kaehler, "Opencv,"
Dr. Dobb's journal of software tools, vol. 3, 2000.

[29] L. Spacek, "Collection of facial images: Faces94,"
Computer Vision Science and Research Projects, University of Essex, United Kingdom, http://cswww. essex. ac. uk/mv/allfaces/faces94. html, 2007.

[30] A. Kansal, S. Saponas, A. Brush, K. S. McKinley, T. Mytkowicz, and R. Ziola, "The latency, accuracy, and battery (lab) abstraction: Programmer productivity and energy efficiency for continuous mobile context sensing,"
ACM SIGPLAN Notices, vol. 48, no. 10, pp. 661–676, 2013.

[31] A. Canino and Y. D. Liu, "Proactive and adaptive energy-aware programming with mixed typechecking,"
ACM SIGPLAN Notices, vol. 52, no. 6, pp. 217–232, 2017.

[32] H. Mao, R. Netravali, and M. Alizadeh,
"Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ACM, 2017, pp. 197–210.

[33] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica,
"Chameleon: Scalable adaptation of video analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ACM, 2018, pp. 253–266.

[34] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,"
arXiv preprint arXiv:1510.00149, 2015.