ACIT5900

MASTER THESIS

in

Applied Computer and Information Technology (ACIT)

August 2021

Cloud-based Services and Operations

Investigate AI-based learning for cloud services for adaptive autonomous behavior

Eman Moustafa Azab

Department of Computer Science

Faculty of Technology, Art and Design

OSLOMET

Investigate AI-based learning for cloud services for adaptive autonomous behavior

Eman Moustafa Azab

Investigate AI-based learning for cloud services for adaptive autonomous behavior

http://www.oslomet.no/

Printed: Oslo Metropolitan University

Abstract

Cloud computing provides more reliable web services due to its flexibility for accessing resources on-demand and self-managed services. However, cloud computing faces new challenges when managing a massive amount of services in an environment full of uncertainties. Moreover, the customer´s variations of services requirements make them more complex. All of these factors increase the difficulty of managing the services on the cloud.

The two keys for providing a reliable web service are service performance and resources utilization. Service performance grantees providing reliable service in terms of response time, but that should be achieved without wasting the limited amount of available resources. Therefore, providing an optimized service without under- or over-provisioning is a requirement. There is a need to develop an autonomous service to adapt itself based on the surrounding environment.

This thesis explores introducing a learning automaton algorithm for developing an autonomous web service. The result is aSpace machine that has been built and developed through a set of phases and shows a progression from an autonomous scalable service towards the aSapce machine. The aSapce machine optimizes the service performance in terms of self-provision.

The results indicate the ability of the aSapce machine to manage the different workloads effectively by providing an elastic web service. Further exploration for this can pave the way towards handling more complex situations.

Acknowledgments

I want to thank my supervisor Kyrre Begnum for invaluable advice, help, and motivation during the work with this thesis. This year has been a challenge with the online studying and with kids at home, but continuous support from my supervisor helped me keep my motivation throughout the semester. His support did not limit the technical advice and ideas, but it expanded to encouraging words that kept me motivated while working on the thesis. Moreover, I would like to thank Stefano Nichele for his guidance through the process to understand the learning automata algorithm.

My appreciation would be extended to OsloMet, for providing me with the necessary skills to undergo this work and facilitating learning and academic progress during this challenging last year of the Covid pandemic.

Last but not least, I thank my family, my loving mother, and my father for everything they have done for me because I would not do that without their encouragement. Also, I would like to thank my husband, who encouraged me to study and support me in my difficult moments. Also, my two lovely kids were always my inspiration during this journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

During the last decades, the internet has become unavoidable in our daily life. It makes our life easy, fast, and straightforward. Thus, it has become the preferred medium for daily communications. The continuous growth of internet services has led to expanding the number of resources.

For responding to these growing demands on internet services, servers, networks, storage, development tools, and data centers have been developed and provided. Furthermore, the companies had to hire many IT professionals to run, develop, and maintain these resources. Consequently, the cost and the complexity of running and maintaining these resources have been increased.

As a result, the term *"cloud computing"* was created. Cloud computing comes as the answer to the question: "why do I need to purchase, set up, and maintain infrastructure if I can work with a shared resource pool?". Cloud computing technologies have become a way to provide on-demand services for whatever the customer needs and whenever he/she demands it. Therefore, cloud computing leads to a new paradigm for modern computing.

The embrace of cloud-based services not only guarantees to get reliable services but also promises them to be cost-effective. Consequently, the adoption of cloud-based services has not been limited to companies but has been attracted governments, individuals, and different sectors.

With the continuous rise of cloud computing and its services, managing the tasks associated with managing these environments increases and becomes more complex. Manually deploying and operating functions such as provisioning, setting up virtual machines, configuring resources, scaling, and monitoring is repetitive, inefficient, and often prone to errors—consequently, the need for automation appeared.

Cloud automation is essential to successfully managing and optimizing

system performance, and reduce resources consumption. Further, it improves efficiency by reducing repetitive tasks such as deployment and configuration.

However, automating the repetitive manual work was not enough to manage the challenges brought by the increased complexity and scale. Therefore, there was a need for more sophisticated solutions.

After the advent of autonomic computing, cloud computing has used it to develop self-manageable systems. The self-managed ability of autonomic systems makes the system administrators free to focus on high-level goals and reduces the complexity of managing provided services.

Developing a strategy that deals with the dynamic nature of services in a cloud that is adaptive to real-life cases and local context is a research area that has attracted the attention of many different research disciplines. However, the complex nature of these models, combined with their resilience on training data, has prevented any more breakthroughs.

Reinforcement learning is a sub-field of AI that does not rely on training data and promises to be adaptive to changing conditions. However, this approach is still evolving, and more research is therefore needed.

1.1 Motivation

This thesis approaches the topic of web service optimization from service perspective. Our goal is to let an intelligent web service acts as an autonomous agent. This agent should be able to understand and interact with the surrounded environment. Introducing such a perspective represents an essential step moving towards emerging concepts such as fog computing, edge computing, mobile cloud computing, and IoT. In such a world, the services should have the ability to make many decisions on their own. For example: the web service can decide to grow or shrink (i.e., change its size by increasing local resources) based on the number of current requests. Moreover, it can determine the optimal place for the execution based on the customer's geographical location.

1.2 Problem Statement

The goal of the thesis is to apply exploratory research on investigating introducing Learning Automata (LA) with cloud operations to achieve autonomous web service.

The problem statements that this project is focused on are as follow:

1. P1: Explore the introduction of AI to service management by designing a learning automata-based model.
2. P2: Develop and implement a prototype based on the designed model.

The proposed design of learning automata aims to develop autonomous web service in a stochastic environment with high uncertainty and a lack of information, then exploring the automaton´s ability to learn and adapt to the surrounded environment. Different models and different learning schemas will be used to design and develop the prototype based on it.

1.3 Outline

The rest of the thesis is organized as follows:

- Chapter 2 explains background and related works to this project.
- Chapter 3 illustrates the methodology of the research
- Chapter 4 explore a learning automata-based algorithm for developing an autonomous web service running on a stochastic environment.
- Chapter 5 and 6 present the proposed system architecture and implementation of the prototype.
- Chapter 7 describes the result, analysis and evaluation of the system.
- Chapter 7 contains discussion and future works.
- Chapter 8 concludes the thesis.

Chapter 2

Background

This chapter provides the reader with an in-depth view of cloud computing technologies and how they positively reduce the complexity of managing and delivering web services through different service models. Further, we introduce and discuss how the adoption of autonomic computing into cloud computing opens the door to shift the concentration from control service to self-management service. Next, we introduce reinforcement learning and its power on autonomic computing to develop self-adaptive services. Finally, we review some research efforts of achieving autonomic computing in terms of resource management.

2.1 The need for rental computing services

The increased demands on internet services over the years is remarkable. Due to the high demand, many companies have to invest more in IT facilities by building data centers, purchasing equipment, and hiring IT staff to run and maintain it. However, the increased demands and customer satisfaction were the main reasons to increase the complexity of running and maintaining, consequently increasing the service cost. Meanwhile, the great revolution in virtualization technology and the improvements in Internet bandwidth created on the economic argument for "rental" computing (Voas et al., 2013).

The concept of having service providers that rent out computing resources based on customer demands was considered a lifeline for IT enterprises and businesses organizations, governments, and people. It offered a solution for decades-long challenges represented in raising IT costs and investments for IT enterprises. In addition, it also helped to face the problems of increasing the system complexity and hiring

talented specialists for maintenance and support, expanding demands for business-process simplification, and fluctuating resources-usage. The cloud computing model was build based on the idea of rental computing.

National Institute of Standards and Technology (NIST) defined the cloud-computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" (Mell and Grance, 2011).

Cloud computing can be divided into into three major categories based on provided service: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS). IaaS is a model that provides the infrastructure services such as virtual servers, storage, backup, and other computing resources to enable the users to deploy and run their operating system, applications in the cloud (Mateescu et al., 2011).

2.2 Web services in cloud computing

Web services in cloud computing are a set of services that composed together to achieve a customer´s request. It can be managed through a web interface by humans or software can use API to automate some of the management. These services are stored and distributed in different data centers around the world.

To run any application, one or a combination of services can be used. In that sense, the services in IaaS can be a VM or a set of VMs, and the VM represents here a certain component. Based on that, the web service can be a set of components. For example, a web application-based service may consist of a database-component (MySQL), and a presentation layer component (e.g., Apache server).

The cloud provider offers different infrastructure services in terms of service deployment and service configuration such as installing packages and modules, defining authorizations, and updating configurations. Although the cloud vendors provide tools, significant manual effort is required once a service grows in scale. Therefore management of cloud services deployments is a crucial feature and needs to be automated (Karakostas, 2014).

2.2.1 Automation from API perspective

The deployment process is considered a core function of service management in cloud computing (Li et al., 2012). It implies a set of configuration processes for a particular need. For example, in the configuration process of a web server, the administrator specifies the size of the virtual machine (number of CPUs, storage, memory, etc), also specifies how long the server needs to be running. Further, it can be specified whether the server will be shut down or re-deployed. Imagine if one has to do the same task a hundred times during the day when deploying a new web server. Therefore, the idea of automating service deployment and configuration will reduce the workload and allow one to focus more time on higher-value goals. Automation can be done by writing scripts, and administrators can specify the processes based on local service requirements. The key element is that clouds offer APIs which automate efforts can use. But with the machines in place, the question of "What" actions are to be taken and "When" still remains.

There are some trials to provide automation tools such as VMware ESX and XenEnterprise. These tools abstracts processor, memory, storage, and networking resources into multiple virtual machines (Marshall et al., 2008). However, they are limited to certain virtual machine technology without offering a way to expand the tool's capabilities to local needs. (Begnum, 2006) proposed the MLN tool, it is an open sources tool that supported the design of virtual machines and their internal configurations.

The cloud offers the opportunity for utilizing local automated scripts; however, this can be done only on small deployment actions. So the cloud offered us the actions but not the strategy to do that. In that sense, we can say that

The table was set for the automation, and the only thing missing was the strategy."

2.2.2 The birth of automation of service behavior

Even though MLN provided a way to automate the deployment of the virtual machines with its local need, i.e., local configurations, the complex nature of cloud computing and services models motivated the researchers to go further beyond. Therefore, the attention had been shifted from *system's processes to system's internal state*.

The story began with the rise of "DevOps" (Development and Operations). DevOps supports the continuous collaboration between developers and operations by optimizing the software delivery cycle and time to pro-

duction. Further, using agile methodologies in cloud computing have a significant impact on software delivery. The idea behind DevOps involved writing code in deploying and updating infrastructure. Based on that, Infrastructure as Code (IaC) became the most popular approach for software development and supported code-centric tools.

The emergence of IaC contributed to developing tools for "configuration management" such as Puppet and Chef. Both Puppet and Chef are configuration management and deployment tools that use the IaC approach in their configuration process. The basic idea of these systems depends on the internal state of the system. Internal state means the system should be maintained based on predefined configurations. Therefore, the primary role for these systems is deploying, managing, and maintaining the system based on predefined configurations. These tools provided massive support for engineers at that time because working in fluctuations environments full of uncertainty is too hard (i.e cloud computing). Such tools can write, test, and launch applications without interference from Ops staff to *provision an entire environment with a single click* instead of it taking *a week to build a new environment.*

Based on the above, developing tools that consider the system's internal state and force it to return if it steers from the path was a great revolution. Even though that was done inside the physical machines, but this was *the simplest form of system's behavior*. However, systems in the cloud become larger and more decentralized, and the cloud face challenges with managing and maintaining services such as resources utilization, fault tolerance, monitoring (Tomar et al., 2018). Hence, the need for systems to manage themselves rather than just forcing systems to be maintained based on predefined configuration becomes necessary.

The birth of the idea *"I do not need to check this by myself, I could make it take care of itself"* was in the mid of '90s. When the idea was born, it focused on the behavior concept. However, the story's roots started from "physical machines can take care of themselves" by introducing the "computer immune system" concept. The godfather of "computer immunology" was *Mark Burgess* (Burgess et al., 1998). In 1998, he provided a paradigm capable of detecting the problem, investigating the causes, and mobilizing resources for automatic use. The biology system in the human body inspired the idea of computer immunology by self-healing without interference from the brain. Later, IBM released a document discussing system management complexity and suggested a solution to this problem by developing self-managed systems. As a result, the term "Autonomic

computing" was firstly coined in 2001 (Horn, 2001).

2.2.3 Autonomic computing in a nutshell

Self-management concept is the core of autonomic computing technology. It consists of four properties which represent its cornerstones: *Self-configuration* the computing system can configure itself automatically based on the high-level policy drawn by the system administrator, *Self-healing* the ability of computing systems to detect, diagnose, and repair local problems caused either by software or by hardware failures, *Self-optimization* the ability of the computing system to optimize itself by changing its parameters based on the surrounded environment circumstances, *Self-protection* the ability of the computing system to protect itself against malicious attacks and failures. Moreover, the ability to predicting possible attacks or failures and prepared against them. Shortly, they are called Self-CHOP systems. However, autonomic computing systems do not imply only these properties, but they extend to more. Hence, academic research adopt new terms like self-organization, self-inspection, self-repairing, and more. Thus, it becomes common to refer to them as self-* systems (Tesauro et al., 2004 , White et al., 2004 and De Wolf and Holvoet, 2006). For example, Sterritt et al., 2005 defined autonomic computing systems as self-CHOP, self-governing, self-adapting, self-managing, self-recovery, and self-diagnosis of faults. While Tianfield, 2003 defined it as self-planning, self-learning, self-scheduling, self-evolution, and more.

The emergence of autonomic computing into cloud computing technology changes the way of looking at systems. Each system consists of a set of components, and each component is not working isolated from the others. All components are aware of each other; moreover, they can change themselves based on the feedback they received from each other and the surrounded environment. Based on that, each component is considered a service that has its behavior:

Shifting from a system with an internal state to services that have behavior

IBM introduced the vision of autonomic computing with the main components of a self-management system; however, it did not provide a recipe for building such systems. In the Academic field, researchers have adopted autonomic computing and implemented it using different methods and perspectives for developing autonomic services. Next, we are going to review some of these research works.

2.2.4 Software engineering efforts towards autonomic computing

The emergence of autonomic computing into cloud computing technology made the researchers pay more attention to designing autonomic computing from a system admin perspective and focused on collaboration and coordination, rehearsal and planning, maintaining situation awareness and managing multitasking, interruptions, and diversions.

We will review some of the research efforts that have been done from 2001 to 2020. That will give an overview of the research to adopt autonomic computing into cloud computing systems.

The research efforts in autonomic web services had different approaches. It started from developing models and algorithms for adoption to developing systems to reach developing systems for verifying the outcomes from these autonomic systems. Houben et al., 2005 and Kapoor, 2005 are examples developing algorithms for self-manageability systems. Carzaniga et al., 2008 developed an autonomic repair management service using self-healing properties. Pastrana et al., 2008 developing a dynamic system in terms of configuration and adaptation. Maurer et al., 2012 proposed a self-adaptive management system. The system considered workload variations by re-configuring the VM autonomically based on the incoming workloads. Research also gave attention to verifying the outcomes of self-adaptive systems: Kaddoum et al., 2010 proposing a set of criteria for evaluating self-* properties on system performance. Cámara et al., 2013 introduced an architecture-based approach for evaluating a self-adaptive system.

2.3 AI as *panacea* for autonomic computing?

The last ten years have witnessed significant adoption of autonomic computing into cloud computing systems. The motivation behind the adoption is the high complexity, dynamically, and uncertain nature of cloud computing, making it too hard to manage directly by engineers. The positive result that autonomic computing showed was the motivation of the emerging AI-based algorithms called *"Reinforcement Learning."* One of the reinforcement algorithms is *Learning Automata*.

Learning automata depends on learning by a trial-and-error process until the optimal action is learned. One of its features is interacting with the surrounding environment. Based on the feedback from the environment, the automaton chooses its next action.

Learning through possibilities based on feedback from the environment

is similar to the conditions and the current needs of the cloud services. In that sense, the academic field introduced the learning automata to develop *self-adaptive services*. Another motivation of introducing learning automata is its apparent simple structure and implementation. It does not require large amounts of memory or complicated computations for its reward system because it depends only on the last state of the automaton.

The self-adaptive service has been introduced in cloud computing as a service can optimize itself based on the surrounding environment. It has been provided more adaptive solutions for dynamic resource provisioning (Qavami et al., 2017).

Ranjbari and Torkestani, 2018 proposes a learning automata algorithm for improving resource utilization and reduces energy consumption without violating the quality of service constraints . At the same time, Misra et al., 2014 introduces the learning automata framework for improving the performance in terms of response time without under- or over-resource utilization. Learning automata has been introduced recently for auto-scaling in Kubernetes Toka et al., 2020.

In the next section, we aim to provide the reader with introduction of learning automata algorithms.

2.3.1 Learning Automata algorithm

Learning Process

Learning is a continuous process that aims to evolve the agent´s behavior based on previous experience. Computer science defines the learning process as a process that seeks to train a program or machine to do a set of prescribed tasks. For example: identifying spam emails or pattern recognition, or even computer games (Lakshmivarahan, 2012).

The learning process can be achieved by the interaction between the learner, i.e., agent or the piece of software, and the surrounding environment. In that sense, there are a set of learning methods classified based on *the way of the interaction:*

1. Supervised learning: there is a supervised interaction between the environment and the agents—the environment guides the agent by mapping between the inputs and outputs using labels. Through these labels, the agents gain knowledge about the data features of each class. When the agents are provided with different data sets, they predict the data class based on the previous experience—for example, neural networks.

2. Unsupervised learning: refers to an unguided process where the agent learns from the detected similarities in data features. Here no supervised labels are provided.

3. Reinforcement learning: the agent learns mainly from interacting with the surrounded environment, so the agent receives a reward if it selects the right action. In such a way, the agent learns from the consequences of its previous actions by following a trial-and-error approach. Thus, at each time, the agent seeks maximum rewards by selecting the proper actions. This way of learning enables the agent to interact with the environment, gain more information about it through trials and improve its performance (Thathachar, 1990). Reinforcement learning uses different learning methods such as Q-learning, SARSA, temporal difference learning, and learning automata such as P-model and S-model.

The motivation behind selecting learning automata is that it does not require training data and can be used online without any prior information. Based on that, We found that using learning automata suits the uncertainty, unpredictable, and fluctuating nature of the cloud computing environment and resources management.

Introduction to learning automata

Learning automaton is a model operating in the framework of reinforcement learning (Ranjbari and Torkestani, 2018). The automaton´s learning approach is a process of determining an optimal action from a set of actions. Then the environment evaluated the selected action and send a reinforcement signal to the learning automaton.

The automaton can be classified based on the mapping between the inputs and the outputs to: *deterministic automaton* and *stochastic automaton*.

Deterministic automaton refers to dependency between the input and the output of the automaton. In other words, the outcome of moving from one state to another is based on the input, while if it is not, then it is *stochastic*.

The stochastic automaton suits more with the topic of research due to its uncertainty and dynamical nature of the problem. Through the following subsections, we will try to cover essential parts of the learning automata algorithm in the following paragraphs. It is too hard to cover all the parts of this algorithm due too many technical details related to the AI field. Instead, the focus will be on the main parts that help us understand how

this algorithm works and insight into the different learning schemas and the different models.

Automaton structure

The automaton is an abstract agent with a finite number of actions, and it selects from these actions and applies it to the environment (output). Based on the environment response towards this action which represents the input to the automaton, the automaton transforms from one state to another state. Finally, based on the current state and the environmental response, the next action is determined.

The above process can be expressed formally as following: the automaton is described by a quintuple (Φ, $\propto$, β, F, G) where $\propto$ refers to a set of the available actions $\propto = (\propto_1, \propto_2, ..., \propto_r)$, and the automaton must choose from. The $\Phi = (\Phi_1, \Phi_2, ..., \Phi_s)$ is a set of the states, while $\beta = (\beta_1, \beta_2, ..., \beta_q)$ is a set of inputs. The $G : \Phi \rightarrow \propto$ denotes the output function, and $F : \Phi \times \beta \rightarrow \Phi$ refers to transition map that computes the state $\Phi(t+1)$ from state $\Phi(t)$ upon receiving the input from the environment. The selected action at time instance (n) is denoted by $\propto(n)$ and represents the input to the environment, while $\beta(n)$ at time instance (n) is considered the environmental response.

Environment

Figure 2.1: *Environment* Narendra and Thathachar, 1974

The environment is a random medium that the learning automaton wants to maximize its rewards from by selecting the optimal action. Based on that, the environment (figure 2.1) receives and input from the automaton $\beta(n) \in \{\beta_1, \beta_2, ..., \beta_r\}$ and sends output (response) belonging to a set X $\in \{0,1\}$. The probability of emitting a particular output is depending on the input. At every instance, each action will have a certain probability $c_i \in \{c_1, c_2, ..., c_r\}$, known as penalty probability. Then the reward probability is obtained by $d_i = 1 - c_i$ (Narendra and Thathachar, 1974). Finally, if the c_i and d_i do not depend on the n, the environment is said to be *stationary*. Otherwise, it is *non-stationary*.

Stochastic learning automaton

The automaton represents the learning automata model's learning unit, while the environment represents the medium in which the automaton or group of automata can operate. In that sense, the interaction between the environment and the automaton in a way that helps to improve its performance is referred to *"learning automaton"*, figure 2.2.

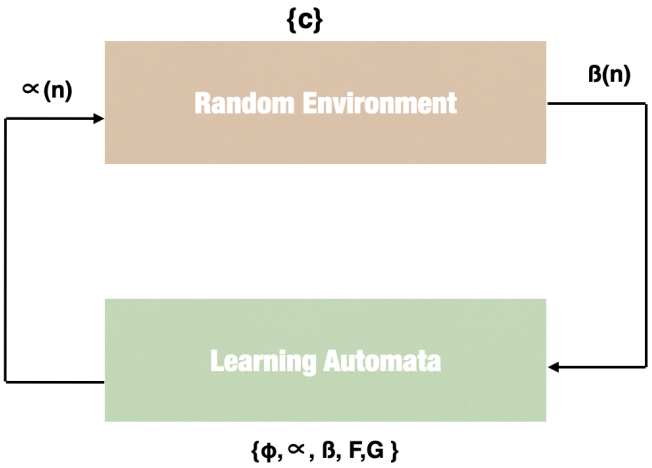

Figure 2.2: *Feedback connection of automaton and environment* Narendra and Thathachar, 2012

The environment response can be considered as an indicator of the automaton's behavior during the learning process because the environment determines whether the action is favorable or not. If the action taken is favorable, then the automaton will be rewarded, and the selection probability of that action will be increased, and the rest of the probabilities will be decreased. In contrast, if the action taken is unfavorable, then the automaton will be penalized, and the selection probability of that action will be decreased, and the rest of the probabilities will be increased. The automaton has a greedy nature of maximizing its rewards. In that sense, the automaton, during its learning process, will learn to select the action(s) that maximized its values.

The classifications of stochastic learning automata

1. The stochastic automaton can be classified based on the action probabilities values into: Fixed Structure Stochastic Automata (FSSA) and Variable Structure Stochastic Automata (VSSA).

    - In FSSA, the action probabilities are fixed.

- While in VSSA, the action probabilities are updated along with time (Johnoommen, 1986). The automaton is described by a six-tuple { Φ, $\propto$, β, F, G, P } where, P is the set of selection probabilities of the actions at an instance t.

2. The environment can be classified based on the reinforcement signal into three classes: P-model, S-model, and Q-model (Ranjbari and Torkestani, 2018).

    - P-model uses binary as an environmental response. Usually, in P-model, the failure or unfavorable response is denoted by one, while the successes or favorable response is denoted by 0.
    - In Q-model, the value of β is a finite number of values in the interval unit [0,1].
    - In S-model, the value of β is a continuous values in the interval unit [0,1].

Basic learning automata schemas

Learning automata´s actions have a probability distribution at each point in time. Furthermore, this probability distribution is adjusted at every time (t) based on the reinforcement signal it receives from the environment. The probability distribution is denoted by p(t) = (p_1(t),.....,p_l(t)) where p_i(t) represents the probability for selecting i^{th} action at time (t) and l denotes the number of actions. We introduce the most common schemas based on the reinforcement signal type:

1. **Binary signal (P-model):**

    There two cases if the action taken was favorable (i.e success) or unfavorable (i.e fail):

    - In case of success:

$$p_i(t+1) = p_i(t) + a(1 - p_i(t)) \tag{2.1}$$

if action i was taken at time step p

$$p_j(t+1) = (1-a)p_j(t) \tag{2.2}$$

$\forall$ j$\neq$i

    - In case of failure this becomes:

$$p_i(t+1) = p_i(t) - bp_i(t) \tag{2.3}$$

if action i was taken at time step p

$$p_{\mathrm{j}}(t+1) = p_{\mathrm{j}}(t) + b[(l-1)^{-1} - p_{\mathrm{j}}(t)] \tag{2.4}$$

$$\forall\, \mathrm{j} \neq \mathrm{i}$$

The constants a and b represent the reward and penalty parameters respectively with $a \in [0,1]$ and $b \in [0,1]$, and l is the number of actions in the action set of the automaton.

2. **Continuous feedback (S-model):**

$$p_{\mathrm{i}}(t+1) = p_{\mathrm{i}}(t) + ar(t)(1 - p_{\mathrm{i}}(t)) - b(1 - r(t))p_{\mathrm{i}}(t) \tag{2.5}$$

if action i was taken at time step p

$$p_{\mathrm{j}}(t+1) = p_{\mathrm{j}}(t) - ar(t)p_{\mathrm{i}}(t) + b(1 - r(t))[(l-1)^{-1} - p_{\mathrm{j}}(t)] \tag{2.6}$$

$$\forall\, \mathrm{j} \neq \mathrm{i}$$

where r is the environmental response at time (t) and r $\in$ [0,1]

Learning schemes:

There are three learning schemas. These schemas are classified depending on the ratio of the constant a over the constant b as following:

- Linear reward-penalty ($L_{R\text{-}P}$): if we set a = b.
- Linear reward-inaction ($L_{R\text{-}I}$): if we set b = 0 so the reward only is taken into account.
- Linear reward-$\in$-penalty ($L_{R\text{-}\in P}$): when a is small compared to b.

2.4 Alternative approaches for achieving autonomic resource management

Many types of research have been conducted to embrace autonomic computing in cloud computing environments to cope with the complexity of today's cloud computing environments. As a result, autonomic computing has been introduced from different approaches—for example; some focus on developing a holistic approach to the resource management system. In comparison, others focus on only one resource management process, such as self-provisioning. At the same time, others introduced autonomic computing to enhance some existing policies such as load

balancing, energy optimization, and quality of service combing with some of the mechanisms like machine learning to add more power to autonomic computing. Moreover, develop autonomic systems that can adapt based on the surrounding environments or even predict the next workload and provide it in advance. Through the following paragraphs, some of these efforts are presented.

Gill et al., 2019 introduced a *holistic approach* to developing a complete autonomic resource management system, including *provisioning, scheduling, and monitoring* processes using the self-CHOP properties of the self-management system. In addition, the developed approach used QoS policy to enhance the provided service and provide self-optimization for energy consumption by clustering the homogeneity workloads and executing them to minimize the energy consumption. However, they did not consider the heterogeneity workloads in the cloud computing. While others focused on one component of the resources management system, such as Dewangan et al., 2019 presented an autonomic resource management technique using self-CHOP for providing an efficient resource scheduling algorithm.

Some research works aimed to add more to the autonomic computing model by adopting AI/ML algorithms while building autonomic systems. Mateen et al., 2020 focused on *autonomic resource provisioning* at the application level on a cloud. They proposed a self-adaptive approach using fuzzy logic to develop a dynamic solution for irregular requests during run-time. While Moreno-Vozmediano et al., 2019 used the machine learning mechanism to predict the distributed server's processing load and estimated the fair number of resources to optimize resource utilization without violating QoS constraints.

The adoption of autonomic computing shows promising results in reducing the complexity of managing resources in the cloud computing environment, further providing the proper amount of resources without violating the QoS constraints. These results encourages the researchers to adopt autonomic computing in other cloud computing paradigms such as fog computing (Tadakamalla and Menasce, 2021).

Chapter 3

Approach

This chapter aims to provide the reader with the entire plan of how the research will be conducted and implemented. However, before further going into details, it is better to recall the problem statement as defined in the introductory chapter: *" investigating introducing Learning Automata (LA) with cloud operations to achieve autonomous resource allocation web service."*

Understanding the nature of the research project sheds light on the path taken and necessary steps to achieve the goal of the research. This project falls into the category of exploratory research.

Exploratory research is a method used for investigating a problem that is not obviously specified (Bhat, 2019). Exploratory research is used in types of topics that need to be investigated more in-depth or not have not been attempted before. It is not always a way which develops a complete solution for the proposed problem or derives conclusive results. However, the primary goal is to bring up new research paths to be explored or establish a strong foundation for exploring ideas (Bhat, 2019). In that sense, it requires the researcher to be flexible and not be restricted to one idea or path because everything depends on the findings and outcomes that come along the way (Bhat, 2019). These findings could be positive or negative, and based on that, new research can use these findings to build upon or select other approaches.

3.1 Desired outcomes

As illustrated above, the nature of this research is exploratory. It opens the door to introduce exploring questions and follows the path of: "what will happen if I do this instead of that?". Thus, trying different ways and asking more questions which helps in exploring more areas.

The path chosen will explore the usage of learning automata to develop an autonomous web service that can be self-adaptive based on the surrounded environment. It is a comprehensive objective; therefore, we will focus our attention first on the resource allocation side. We can conduct more investigation later, including other aspects such as the price of the service or the customer's geographical location. Also, we do not know yet whether the findings will pave the way to add more aspects to be investigated or not. Further, the concept of autonomous service is not a new one, as shown in the background chapter. However, the point of view that we are introduced our research from is relatively new. Based on that, the nature of our research project, i.e., exploratory research, no fixed and conclusive results are to be expected from this project.

It is unnecessary to develop a fully functional agent but provides the technology and the idea to be explored. We are unsure whether it will work or fail; however, we provided a guide for researchers who will continue the same research path. The value here is the journey itself towards the findings, not to develop a model or tool. Our project looks like an excavation mission. Some missions end with successful results and finding the treasure, while others end with nothing. However, the real treasure here is inside the journey, and the gained knowledge also not wasting future others' resources and time by following the same path.

3.2 Objectives

In our research, we are looking to introduce new perspectives to be investigated. Our motivation towards that is the new technologies that have been emerged into cloud computing technologies such as edge-computing, mobile-clouds, and IoT. These technologies create a need for new innovative ideas to be handled. As a result, new concepts and technologies have been developed, such as mobile agents, autonomous services, and decentralized control systems.

In that sense, we are looking at resource allocation not from the resource management system perspective but from the service perspective. Looking from the service perspective gives opportunity to think of the *"service as an agent"*. Furthermore, integrating autonomic computing technology and agent technology with cloud-based service provides an *"autonomous agent."* The idea of the autonomous agent elaborates the concept of *"decentralized control,"* which now suits edge-computing, IoT, and mobile cloud technologies.

Developing an autonomous agent using learning automata algorithm represents the core of our research study. We will divide our study into a set of phases. Each phase will explore a new aspect or introduce a solution for the problem that appeared in the previous phase. The connection between the different phases likes a chain of assembly of connected pieces. Each piece connects to the next one.

Each phase is considered a complete piece of work. It has its own goals to be addressed at the beginning of the phase, then the design part where the plan of the experiment and the other details regarding the environment and other elements will be provided.

As mentioned above, the study is divided into phases; therefore, the design, implementation, and analysis will follow the same way of processing through the different phases of the study.For the implementation part will explain the experiment steps and the code in detail, and the results part will provide the results of the experiment. Finally, the observation part where the analysis of results will be provided and new questions will be addressed for the next phase.

3.3 Design

As mentioned before, our research journey will be divided into a set of phases. We will explore a new thing at each phase or try to solve a problem that could be addressed from the previous phase. Based on that, the first phase will be the foundation of the research journey because exploring the learning automata will be done during this phase, and based on the results, the rest of the phases will be developed.

During the first phase, the design model of the autonomous service will be provided using the learning automata algorithm. Moreover, for a deep understanding of the behavior of learning automata, the three learning schemas of learning automata will be used. Furthermore, a comparison between the different learning schemas will be made to understand the differences between the three learning schemas through the results. Finally, based on this, one of these learning schemas will be selected for subsequent phases.

In the second phase, the developed model of the autonomous service will be exposed to a real scenario by using actual traffic workloads and simulating an actual web server response to investigate the behavior of the autonomous service. However, before implementing the actual scenario, we may enhance the designed service model or develop a new feature or

a solution to tackle the finding obstacles based on the findings from phase one.

In addition to exploring results in what can be considered normal operations, the service model must be tested for robustness under more unusual conditions by looking at edge cases. Thus, the final phase will investigate the robustness via asking: How will the autonomous service react if the web server was under anonymous attack? Or will it be able to learn and converge under this attack, or won't it adapt and protect itself by stopping the web service? Things like this do happen, so testing for such cases is essential to make an algorithm that could perform under real-life conditions.

3.4 Implementation and Simulation

The implementation phase represents converting the design to an initial prototype. First, the conceptual design of the autonomous service structure will be used to build up the prototype. Moreover, the three learning schemas of the learning automata will be implemented. Those schemas are considered as the fuel for the autonomous service used for the learning process of the autonomous agent. Further, the environment where the autonomous service should live will be simulated. This simulation aims to provide close insights to study the behavior of the automaton under specific circumstances.

As mentioned in the previous section, that each phase leads to the next one. Therefore, the final prototype will be done through a process composed of different stages. Each stage introduces a different level of autonomic sophistication.

Each stage moves towards enhancing the learning process of the autonomous service and improving its self-adapting towards the surrounded environment. The feasibility of developing an autonomous service for resource allocation will be proved through the different scenarios towards building this prototype, which opens the space for creating a line of logic. This logic line shows whether this autonomous service is just for show or provides a solid understanding that allows access to much more profound and much more advanced scenarios that enable us to move this research forward in the future. That means even the different cases might change perspectives from a technical to a more architectural discussion, if necessary.

The way of implementing the autonomous service in a real-life scenario

is not clear yet. Therefore, we will dedicate the first phase to implementing the initial prototype. Then, based on the findings, we will add more features or enhance the initial model of the autonomous service to be suitable to our environmental conditions. We have to point out that the simulated environment will be known from before. For example, the traffic capacity of used VMs in terms of responding in time will be known. Therefore, it will be used as an indicator of service behavior. That will give way to understanding the automaton behavior and accurately evaluating the performance of the developed autonomous service.

In the first phase, the P-model will be adopted due to its simplicity of implementation. Then, after gaining a comprehensive understanding of the behavior of learning automata, the S-model will be implemented to add more complexity to our research and dive more in-depth.

Our crowning achievement from this phase would be a completely functioning prototype and a demonstration of its capabilities. Finally, it is essential to mention that the design accuracy of our autonomous service will be investigated at the implementation stage through the different phases.

3.5 Analysis plan

Analysis phase aims to verify the completeness of the designed model and the implemented prototype. This analysis will be done at each phase after the implementation stage. Based on the analysis of the results, the direction of our compass and the focus of the next phase will be determined.

Therefore, we can expect the following questions to be answered in this phase:

1. What are the findings after each phase?
2. How does the implemented prototype work?
3. Did the autonomous service achieve the desired results?

Chapter 4

Phase I: Autonomous web service algorithm based on A Single Learning Automaton

In the following chapter, we introduce the autonomous agents from a service perspective. The chapter starts with an explanation of a web service as an agent and its learning unit. It will then followed by a description of the simulation process and the assumptions. The implementation section presents explanations of different operations and functions that have been performed. Finally, The results and observations are provided. The focus will be on creating and observing a single learning automaton. Hopefully, this first step will provide us with a component that can be used in more advanced learning systems later.

4.1 Introduction

Nowadays, web services are developing very fast, and on the other side, the cloud computing environment has a dynamic and stochastic nature besides the dynamic changes in the customer's requirements. Besides the limited resources on cloud computing, all of these factors form a real challenge for service providers to provide a reliable web service.

Since humans can no longer react and adjust to changing conditions in large and complex services. There is a need for a service to scale itself to accommodate the changing nature of requests, have high resiliency to failures for interrupted service, and reply within acceptable time bounds.

The load balancer is one of the innovative solutions that has been developed. While the cloud has a set of heterogeneous servers, the

load balancer aims to distribute the workloads across those servers (i.e., resources) to optimize the provided service performance regarding replying within an acceptable time. Therefore, the load balancer helps provide optimal performance from the resources management point of view.

However, this thesis introduces the web service optimization topic from a different perspective; it is the service perspective. Our goal is to let an intelligent web service acts as an autonomous agent. This agent should be able to understand and interact with the surrounded environment. Introducing such a perspective represents an essential step in a world moving towards emerging concepts such as fog computing, edge computing, mobile cloud computing, and IoT. In such a world, the services should have the ability to make many decisions on their own. Here is a set of examples for such choices:

1. Decide to grow or shrink (i.e., change its size by increasing local resources) based on the number of current requests.

2. Select where to run among available service providers based on the price.

3. Determine the optimal place for the execution based on the customer's geographical location.

Based on the preceding, we organized the first phase to introduce an agent-based architecture with learning possibilities based-on learning automaton algorithm. Our main contribution in this chapter is:

1. Introduce learning automaton-based autonomus web service service using P-model method.

2. Observe the autonomous service behavior based on surrounding environment.

3. Use the findings to design the next phase.

4.2 Web service-based learning automaton

Figure 4.1 depicts the interaction of web service-based learning automaton agent with the surrounding environment. The environment here is the cloud where the service is running. There is a learning unit attached to the web service. The learning unit here represents our learning automaton

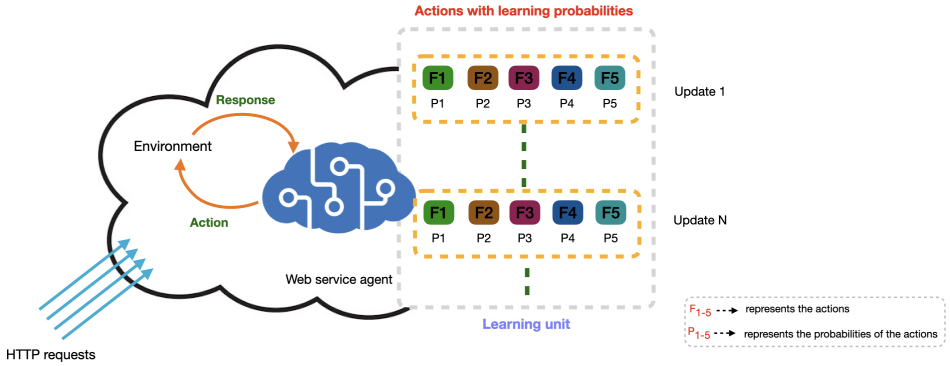

Figure 4.1: *The web service agent architecture has its learning unit. The agent interacts with the environment and receives feedback. Based on feedback, it updates its learning unit.*

algorithm using the P-model method. This method is a simple Reward-Penalty binary feedback. Based on that, the environment response will be "0" if the taken action was unfavorable and "1" if favorable.

The agent learns from trial-and-error experience. Therefore, the agents' actions are set up on the learning probabilities. Initially, there is no information for any optimal action. Thus, we define equal probabilities for all actions, and the first selection will be random. (r) represents the number of the actions and the initial configuration of the automaton is: The Reward and penalty of the learning automaton will be determined

Actions	F1	F2	F3	F4	F5
Initial probabilities	1/r	1/r	1/r	1/r	1/r

Table 4.1: *All of the actions have initial equal probabilities and the selection between these actions will be random.*

according to the environmental response. The automaton selects one of its actions in each iteration based on its probability vector. In the next iteration and before the selection, the automaton updates its actions' probabilities. If the automaton has chosen the right action, it will be rewarded by (see equation 2.1). Otherwise, it will be penalized by (see equation 2.2).

This experiment uses a single learning automaton to adapt the VM size based on the intensity of incoming requests. The aim is to optimize the service performance in terms of scaling up/down the service size based on

the incoming traffic.

4.3 Simulating the behavior of a single automaton

This section is divided into two parts. The first part presents the drawn scenario for a running web service with a learning unit. This scenario has been set to study an autonomous web service behavior using a single learning automaton. The second part is simulating the drawn scenario. We did not implement the drawn scenario; however, we simulate the experiment to study the behavior of our autonomous Web service.

4.3.1 Scenario

The proposed scenario for exploring the behavior of web service agent based-learning automata begins from assiging a one virtual machine (VM) as a web service. This web service is running on the ALTO cloud and managed by the Oslomet administration. There is a web server running on it, and it serves a single static web page. The web service is supposed to receive a set of HTTP requests, and it responds (Figure 4.2). From this point, the learning unit of the autonomous agent, i.e., the web service, will interact. It represents the brain of the autonomous agent. The brain is setting and monitoring the situation, and it should decide to scale up or down or keep on same size based on the incoming requests. ALTO uses the

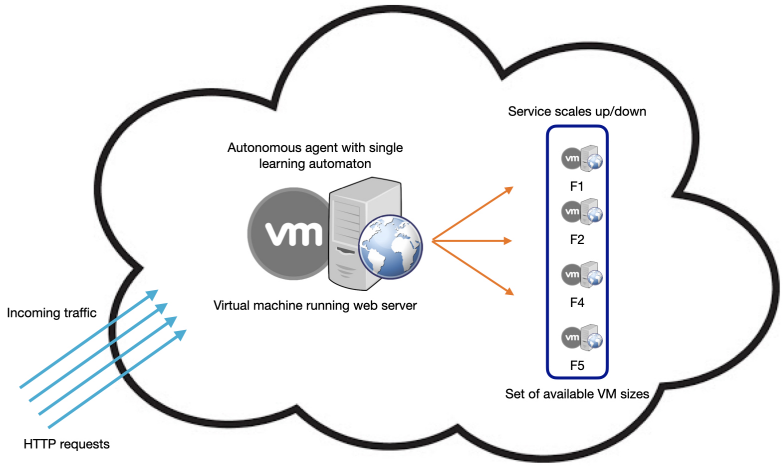

Figure 4.2: *Virtual machine runs web server with custom plug-in that implements a learning automaton. There are a number of "VM sizes" the service can scale up or down.*

term "flavors" to manage the sizing for the instance's compute, memory, and storage capacity. Therefore, the VM size will be represented by five flavors. We refer to these flavors from F1, which is m1.small, to F5, which is m2.xlarge, as illustrated in the table 4.2. Furthermore, the autonomous web service should choose from these flavors based on the incoming traffic.The autonomous agent supposes to learn from the surrounded environment via trial and error.

Flavor	Name	VCPUs	VM Disk (GB)	RAM (GB)
F1	m1.small	1	20	2
F2	m1.medium	2	20	4
F3	m1.large	4	20	8
F4	m1.xlarge	8	20	16
F5	m2.xlarge	8	20	32

Table 4.2: *presents the available flavors that the web service will spin up based-on.*

4.3.2 Simulation

The simulation part is depicting how we did simulate the above scenario. We set a set of assumptions to simulation an autonomous web service based on a single learning automaton. In the simulation, we have three main elements: the web service, the traffic, and the single automaton.

1. Web service.

   We will not run a real web server; instead, there is a python script running on ALTO. This script represents the web server that receives a set of HTTP requests and responds to these requests by exposing the suitable flavor based on the traffic-heavy. Furthermore, the script contains a set of functions. One of these functions is an evaluation function which represents the environmental response. It aims to evaluate the automaton actions based on the current action (flavor) comparing to the current traffic (discuss in the next section). It is worth mentioning that the web service expresses the surrounded environment where the single automaton should observe and interact with.

2. Single automaton.

   It is responsible for the learning process. It learns from monitoring the surrounded environment and behaves based on it. The automaton chooses to scale the service up/down or keep on the current flavor. There is no absolute scale; however, we represented these flavors

by using the names of these flavors as a set of actions (F1, ..., F5). Furthermore, we assumed each flavor has a specific capacity to serve by doing a simple calculation where each flavor can handle ten times its type number. For example, the flavor "F1" would only handle a maximum rate of 10, while "F5" would continuously handle 50. This simplification is necessary, at this stage, to control the environmental response sufficiently to observe the behavior of the automaton properly.

3. Traffic.

   in other words, the HTTP requests. There is no real traffic; instead, it is simulated by having a function that generates a random value after a normal distribution with an average of 25 (will be described in the next section).

To Summarize:

1. Environment: consists of a set of actions, outputs, and rewards/penalties.
2. List of actions: consists of VM flavors that the web service can scale up/down to.
3. Input to Environment: consists of the selected flavor as a response to the incoming HTTP requests to optimize the service performance.
4. output from the environment: consists of the responses of serving the request by selecting the suitable flavor.
5. Reward/Penalty probabilities: the probability for getting favorable/unfavorable responses from each flavor. These probabilities will vary with respect to the capacity of the selected flavor comparing to the traffic intensity.

4.4 Implementation

This section presents the python scripts and points out the main functions that have been used to run the experiment. We used the following python scripts:

1. **Automaton**

   The Automaton script is for structuring the automaton, i.e, the learning unit section 4.2. It contains a set of functions responsible of receiving the environmental response, updates the learning probabilities,

and sending the action. We are going to present the main functions of this script (Detailed script is provided at section 9.1).

(a) Constants a and b:

The constants a and b are the reward and penalty parameters, respectively. In the Automaton script, these constants are provided as inputs for the class constructor, which enables us to change these constants' values and see the effects on the automaton's behavior.

```
    def __init__(self, a, b, name, verbose=1, history
=10):
    # The reward parameter
        self.a = a
    # The penalized parameter
        self.b = b

```

(b) Update Policy function:

This function takes the environmental response as input and updates the learning probabilities based on that response. We have three states of response as following:

- "None" is no response from the environment. This happens only in the first interaction where the automaton is selecting its first action. Here all available actions are assigned to equal probabilities values.
- "0" means unfavorable environmental feedback/response on the previously selected action. Based on that, that action will be punished.
- "1" means favorable environmental feedback/response on the previously selected action. Based on that, that action will be rewarded.

```
    def update_policy(self, environmental_response):
    # first iteration all actions have equal
possibilities
        if environmental_response == "None":
            number_of_actions = len(self.
available_actions)
            initial_prob = 1 / number_of_actions
            for policy_key in self.prob_policy.keys():
                self.prob_policy[policy_key] =
initial_prob
     # Reward selected_action & penalized rest actions
        elif environmental_response == 1:
```

```
            for key,value in self.prob_policy.items():
                if key == self.selected_action:
                    self.prob_policy[key] = value +
    self.a * (1 - value)
                else:
                    self.prob_policy[key] = (1 - self.
    a) * value
    # Penalize selected_action & reward rest actions
        else:
            for key,value in self.prob_policy.items():
                if key == self.selected_action:
                    self.prob_policy[key]= value-(self
    .b * value)
                else:
                    self.prob_policy[key]= value+self.
    b*abs(1/(len(self.available_actions)-1)-value)

```

(c) Select action function:

This function selects the action to be taken in the next step. First, the function checks if all actions have equal probabilities. Then, it randomly selects one of these actions. Otherwise, the action will be selected randomly based on its probability using "NumPy.random.choice()" function.

```
    def select_action(self):
    #check if all actions have same probabilities
        prob_equal = len(list(set(list(self.
    prob_policy.values())))) == 1
        if prob_equal:
            random_action_index = random.randint(1,
    len(self.available_actions))
            self.selected_action = self.
    available_actions[random_action_index -1]
        else:
            self.selected_action = np.random.choice(
    list(self.prob_policy.keys()), p=list(self.
    prob_policy.values()), size=1)[0]
        return self.selected_action

```

2. **Plot Results**

The Plot_results_phase1 scrip is plotting the results of the automaton behavior when applying different schemas. The schemas have variations on the values of the constants of (a) and (b). To plot the automaton behavior, we created a buffer memory to store the

probabilities' values of the actions during the learning process. Therefore, the generated plots will help to study and understand the automaton behavior when changing the values of the constants (Detailed script is provided at section 9.2).

3. **Run Simulation**

   The run_experiment_phase1 script is for the learning process of the automaton. It contains a set of functions and one loop. The loop represents the number of iterations that the automaton has to go through and thus representing the length of the simulation. We are introducing two main functions (Detailed script is provided at section 9.3):

   (a) Generate rate function:

   This function is simulating the number of incoming HTTP requests to the service to be handled. We used the *NumPy.random.normal(loc=25) function* to generate a random value after a normal distribution with an average of 25.

```
    def generate_rate():
        return np.random.normal(loc=25)

```

   (b) Evaluate action function: This function generates the environmental response. It has two inputs: the taken action and the current rate. Then it evaluates the taken action by comparing it with the current rate (generated by the previous function). The output is binary, "1" is an favorable response, and "0" is a unfavorable response.

```
    def evaluate_action(action, current_rate):
    #calculate "capacity" of chosen flavor
    capacity = int(re.search(r'\d+', action).group())
    * 10
    if capacity > current_rate:
        return 1
    else:
        return 0

```

4.5 Results

In this section, we are going to present the results of executing the script run_experiment_phase1 (section 9.3). To explore, understand, and gain

a good insight into the automaton's behavior and its interaction with the environment, we decided to run different simulations. Each type intends to explore a particular side of the automaton behavior.

4.5.1 Explore the automaton behavior by changing the proportion of (a) to (b) using the learning automaton's schemas

We are using the three schemas of learning automaton. In the three schemas, the proportion of (a) to (b) is changing. Therefore, it enables us to explore the automaton behavior when changing a to b. We will fix a value of a =0.1 and set 250 as the number of updates for the learning process to compare the automaton behavior between different schemas.As mentioned earlier, a and b represent the parameters of reward and penalty, respectively.

- **The Linear Reward In Action Schema (L_{R-I}):** This algorithm can be obtained by setting b = 0 where no updating under a penalty input.

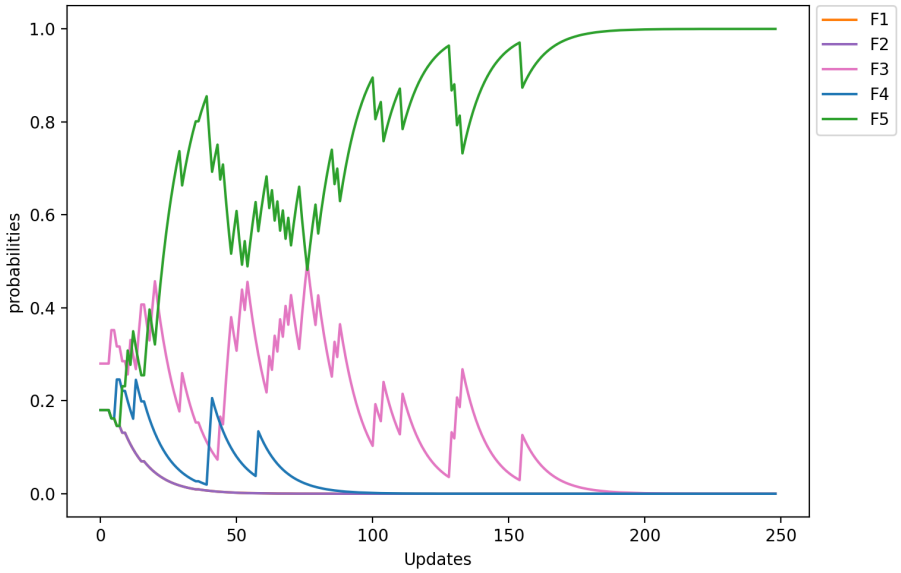

Figure 4.3: *depicts the behavior of linear reward in action schema (L_{R-I}) with a = 0.1 and b = 0.*

Figure 4.3 shows that the flavor F5 has the highest probability for converging after 190 updates to one. On the other side, it does not seem that the flavor F1 has a chance to be selected. While the flavors F2 and F4, after 50 updates, their chances to be selected reached 0.

Flavors F3 and F5 are the continuous competitors in the automaton learning process. It is worth observing; both actions have a similar pattern of oscillations but in opposite directions. The figure shows that after 30 updates steps, both actions have increasing probabilities to be selected. However, after that point, this a dramatic change in selection probabilities as the action F5 takes the up curve while F3 takes the down curve. Both actions meet again after 80 updates steps, and after this point, both actions take a fixed direction to continue. F5 keeps high on reaching the coverage to 1 after 190 update steps. On the other side, F3 keeps low on reaching the coverage to 0 after 190 update steps.
After repeating the simulation several times, we got the following results:

- F5 and F3 have got 40% of chances to converge to 1, while F4 has got 20% of chances for convergence.
- Having F2 getting out early and almost not selecting flavor F1 was continued in the same vein.
- The competition of converging to one was between F3, F4, and F5 flavors during all the trails. The roles of being the winner, competitor, and leaving were happening between the flavors interchangeably. F5 has got a 40% chance of being the winner and 50% of the competitor. At the same time, F4 got a 20% chance of being the winner and 50% of the competitor. Finally, F3 got a 40% chance of being the winner and 20% of the competitor.
- The learning process for the automaton is ranging between 80 to 220 update steps.
- It worth observing, in this schema and with a=0.1, there are two competitors, and one action leaves early. One action did not get a chance to be selected F1, while the other action F2 has a very low chance to be selected.

- **The Linear Reward Penalty Schema (L_{R-P}):** This algorithm can be obtained where $0 < a < 1$ and $b = a$.

  In the Reward-Penalty schema, there will be a punishment to receive unfavorable responses from the environment, and the value of b will be the same as the value of the reward (a). This schema is the opposite of the previous one (Reward-In-Action), where the reward

takes action and no punishment for negative actions. We are going to explore the behavior of the automaton based on using $L_{R\text{-}P}$ schema.

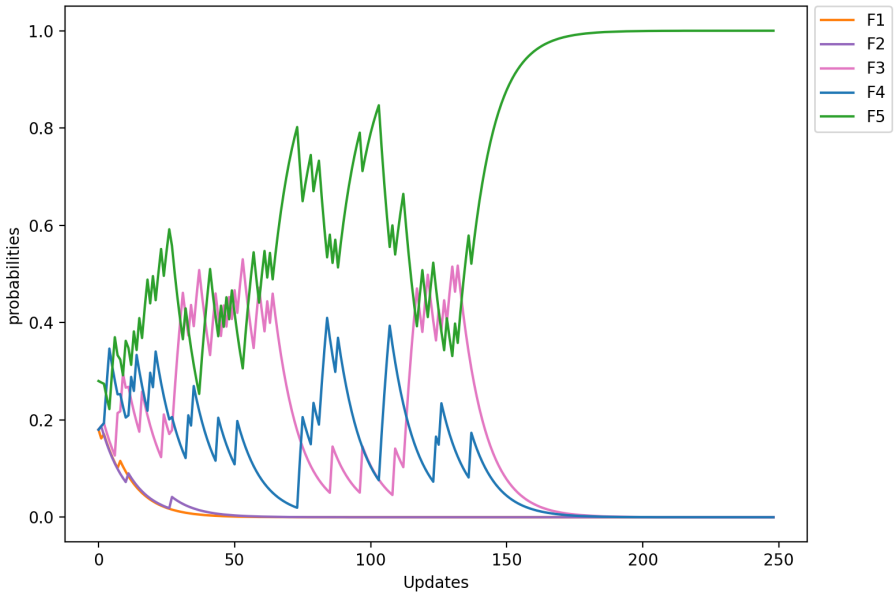

Figure 4.4: *depicts the behavior of linear reward penalty schema ($L_{R\text{-}P}$) with a = 0.1 and b =0.1.*

Figure 4.5 shows the results of simulating with a=0.1 and b =0.1 with 250 update steps. It is worth observing that flavor F1 is getting chances to be selected during the learning process of the automaton, which is the opposite behavior of the same flavor in ($L_{R\text{-}P}$) schema. Action F1 starts with a probability of 0.2, and it slows down until reaching 0 after 45 learning update steps. While the oscillations in the learning curve of flavor F2 have become much smaller and slows down significantly at the same number of update steps as F1. Flavors F3, F4, and F5 show learning curves with sharp oscillations until step 50, where F4 has a dramatic change towards down and F3 and F5 continuing up. Between steps 70 and 120, there was a dramatic change in the learning curves of F3 and F5. In comparison, we observe F4 from 70 to 150 update steps the oscillations become small, and on the other hand, the learning process slows down significantly. At the 180 update step, the action F5 was able to converge to 1, and the F3 and F4 reached 0. F5 was declared as the optimal action after 170 updating steps. After introducing b = 0.1 as a punishment parameter, the three flavors, F3, F4, and F5, continue in

the automaton's learning process. This phenomenon is the opposite of the actions' behaviors in the($L_{R\text{-}P}$) schema, as only two actions were competing to converge to 1.

After repeating the simulation several times, we got the following results:

- 80% of trials could cover one, while 20% could not get coverage through 250 update steps. While in the ($L_{R\text{-}I}$) schema, all the trials coverage to 1.
- F4 has got 40% of chances to converge to 1, F3 and F5 have got 20% of chances for convergence, and 20% could not converge to one.
- After introducing a punishment parameter with a value equal to the reward parameter, it becomes possible to have flavor F1 through the learning process of the automaton. Also, 50% of trials have three actions F3, F4, F5 are competing to converge to one instead of two actions like the ($L_{R\text{-}I}$) schema.
- The learning process of the automaton is between 100 to 160 update steps. While in the ($L_{R\text{-}I}$) schema, it was between 80 to 220 update steps.

- **The Linear Reward-$\mathcal{E}$-Penalty($L_{R\text{-}\mathcal{E}P}$):** This algorithm can be obtained by setting the parameter b to a small positive number (b « a < 1). Figure 4.5 shows the results after running the simulation with values a =0.05 and b=0.1. The oscillation of the actions is much higher in this schema comparing with other schemas. The probabilities of flavors F1 and F2 are much higher comparing to schema $L_{R\text{-}P}$, and the period of their learning processes are more extended than in the $L_{R\text{-}P}$ schema. We observe on the one hand that the oscillations for flavors F1 and F2 become much smaller, and on the other hand, the learning process slows down significantly after 140 update steps. At step 160, the flavor F4 has a flat learning curve going down to reach value 0, while F4 at the same step has fluctuated learning curve. For flavor F3, the situation was different as it reached at 200 update step to one, then it goes down to a probability of 0.8, and it did not reach at one again through the provided number of the update steps 250. Finally, flavor F5 shows similar behavior with flavor F4; however, it displays a high probability of being selected than flavor f4, and its learning curve continues after 250 update steps.

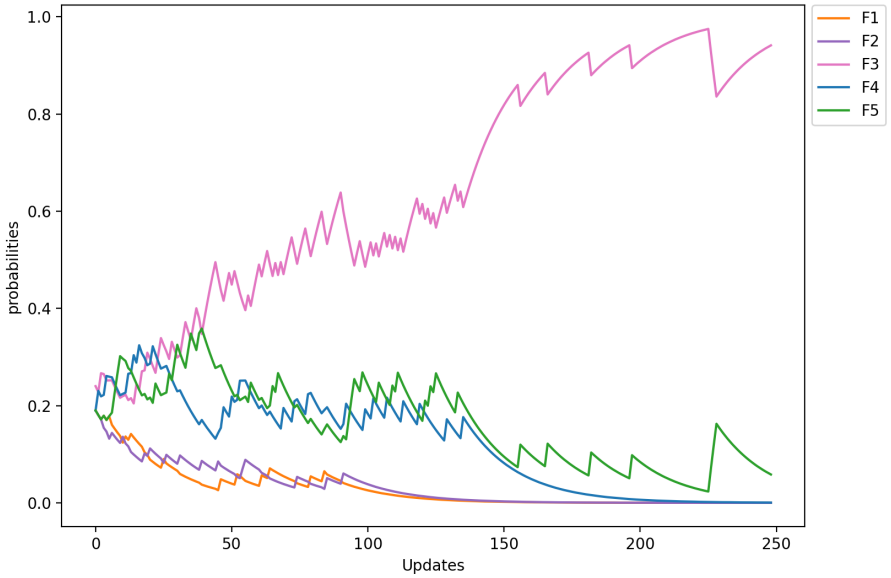

Figure 4.5: *depicts the behavior of Linear Reward-$\mathcal{E}$-Penalty schema ($L_{R\text{-}\mathcal{E}P}$) with a = 0.05 and b =0.1.*

After repeating the simulation several times and increasing the number of update steps to be 1000, we got the following results:

- 50% of trials could cover one, while 50% could not get coverage through 1000 update steps.
- F5 and F3 have equal chances of 20% converging to 1, F4 has got 10% of chances for convergence, and 0% chances for F1 and F2 for converging to one.
- After introducing a punishment parameter with a value higher than the reward parameter, 50% of trials have three actions F3, F4, F5 are competing to converge to one same like ($L_{R\text{-}P}$) schema.
- The oscillation in the behavior of the different flavors was very high comparing with the other schemas. It worth observing, in some trials, there was a high competition between three actions. Furthermore, we found that one of the actions was advanced at some learning steps than the others. However, the other actions could compete with the advanced action and become the optimal actions during the learning process of the automaton, and so on (figure 4.6). This way of learning and the high oscillation of the automaton's behavior was very high in this schema compared with the other schemas. The rest of the actions

were competing for 200 learning steps before reaching 0 probability. The learning process for these flavors was long comparing with other schemas, which gives the automaton a chance for better learning and good performance.

- The learning process of the automaton of 50% of the trials was higher than 500 update steps, while the rest did not provide coverage to 1 through 1000 update steps.

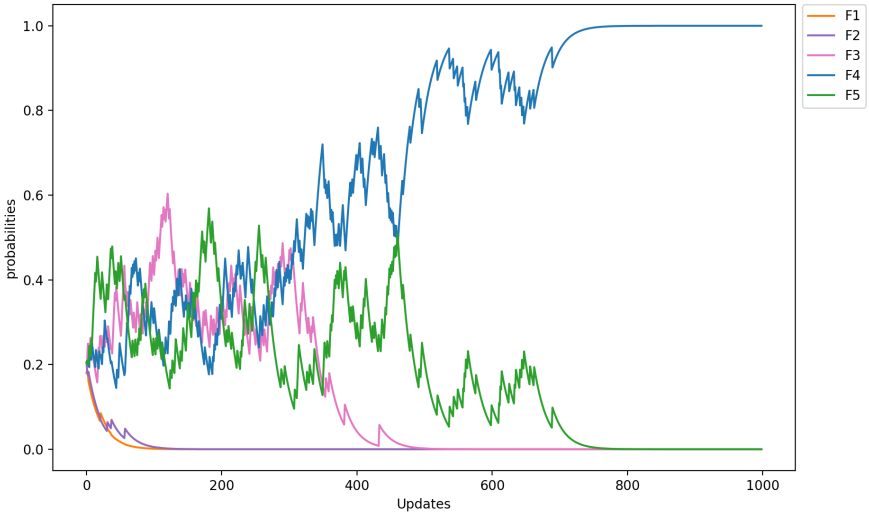

Figure 4.6: *depicts the behavior of Linear Reward-$\mathcal{E}$-Penalty schema ($L_{R\text{-}\mathcal{E}P}$) with a = 0.05 and b =0.1.*

4.5.2 Explore the effects of changing the values of (a) and (b) on the automaton's behavior

From one side, we observed the behavior of automaton using the different learning methods. Furthermore, we found that the ($L_{R\text{-}I}$) and ($L_{R\text{-}P}$) schemes converge to pure policies, while the ($L_{R\text{-}\mathcal{E}P}$) schema no longer converges to a pure policy but a mixed policy.

From the other side, a vital aspect of achieving an autonomous service is choosing an appropriate learning method to reinforce action probabilities for a learning automaton. The learning method directly impacts the adaptation ability of the automaton to deal with changes in the service´s environmental conditions, for example, the traffic and network conditions, and determines its overall efficacy and optimality.

Therefore, we will be interested in plotting the behavior of the automaton using ($L_{R-\varepsilon P}$) schema.

In the sequel of the discussion, we will, in particular, be interested in the convergence to mixed policies due to the dynamic nature of the environment and the dynamic behavior of the automaton using the ($L_{R-\varepsilon P}$) schema. Therefore, we simulated the behavior of the automaton using $L_{R-\varepsilon P}$ schema with different settings of parameters of a and b (Table 4.3).

a	b
0.075	0.15
0.125	0.25
0.25	0.5
0.375	0.75
0.5	1

Table 4.3: Values of a & b

After running the simulation several times using the values provided in table 4.3, we got the following results:

1. Automaton's behavior has high oscillations using the first and second values of a and b, illustrated in table 4.3. However, the oscillations continued to decrease using the last three values of a's and b's until it gained a flat curve with a dramatic change towards up or down.

2. Since increasing values of a and b parameters, the automaton reached the converge of 1. These results are opposite to the previous one with a = 0.05 and b =0.1 (figure 4.6). That concludes, the high value of parameter a, the more chance to converge to 1 the automaton.

3. Automaton's learning process period becomes shorter while increasing the values of a and b. Whereas the learning period was between 180 to 400 steps with a= 0.075 and b= 0.15 to be between 10 to 30 steps with a= 0.5 and b= 1.

4. All available automaton's actions have got chances to be selected during the learning process while running the simulation using the first and second values of a and b in the table 4.3. In comparison, this chance has been decreased by increasing the parameters of reward and punishment. By doing the simulation using the last values in table 4.3, we got only two actions during the automaton's learning

process. The explanation is the high value of the reward, then the action gains a high probability once it is rewarded. Consequently, it increases its chance to be selected next step comparing with the other actions. Furthermore, the same happens with punishment. Once the action is punished, it could vanish from the learning process at all.

4.6 Observation

For developing an autonomous web service using the learning automata algorithm, it would be essential to explore and study the behavior of the learning automaton.

Phase one has provided the first steps of exploring and studying the behavior of a single learning automaton. In this phase, we simulated a web service with a single automaton to scale up/down based on incoming traffic. The simulation provided us with results regarding the different automaton behaviors based on the learning schema and learning parameters. The results provided insights about different sides of the learning automata properties: the nature of the automaton learning curve, the period of the learning process, the availability of the actions during the learning process, and the probability of converging to 1.

We observed that $L_{R\text{-}I}$ and $L_{R\text{-}P}$ schemas had high chances for converging to 1 when a=b=0.1 and a= 0.1 and b=0, respectively. In comparison, this chance decreased with the $L_{R\text{-}\varepsilon P}$ schema when a =0.05 and b =0.1. On another side, the oscillation of the automaton´s learning curve was very high in the $L_{R\text{-}\varepsilon P}$ schema comparing to $L_{R\text{-}P}$ and $L_{R\text{-}I}$, which was very low and had a flat curve. Furthermore, the learning process of the automaton was very long with the $L_{R\text{-}\varepsilon P}$ scheme comparing to the $L_{R\text{-}P}$ schema, while the $L_{R\text{-}I}$ had the lowest period. In the $L_{R\text{-}\varepsilon P}$ schema, all available actions had chances to be selected during the automaton learning process, while these chances decreased with the $L_{R\text{-}P}$ scheme. In the $L_{R\text{-}I}$ scheme, some of the available actions did not have a chance at all. Figure 4.7 shows a comparison between the results of the three schemas.

During this phase, we observed how the constant (a) affected the automaton behavior, particularly with running the second simulation using $L_{R\text{-}\varepsilon P}$ schema and changing the values of a and b. The second phase of the simulation shows how the increment of the reward value using the $L_{R\text{-}\varepsilon P}$ scheme increases the chances of convergence to 1, reducing the period of the learning process and reducing the chances of less rewarded actions to be continued during the automaton learning process.

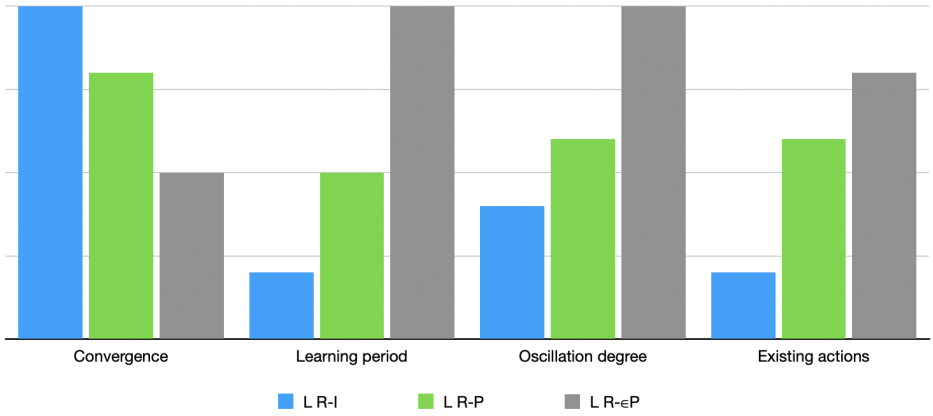

Figure 4.7: *shows comparison between the behavior of Learning Automata using different learning schema*

It worth observing the similarities between the automaton behavior results using $L_{R\text{-}I}$ and $L_{R\text{-}\varepsilon P}$ schemes after increasing the reward value of the latter. These similarities did not exist when we used $L_{R\text{-}\varepsilon P}$ with values a =0.05 and b=0.1 for the first time. However, these similarities arose when we did the second simulation phase since increasing the reward value (a). It shows changes in the learning process period, convergence, oscillation, and the availability of the actions.

Based on the previous observations, we can conclude that the reward value (a) can be considered as the learning parameter of the automaton that affects its performance. Furthermore, the actions F5 and F3 got high chances for convergence comparing to F4.

In phase two, we will introduce aSpace-machine of learning automata. In phase one, we were exploring a single automaton behavior based on the current rate. While in phase two, we will consider the previous rate, current rate, and the time. We will add more dimensions to our environment by having a set of time intervals. At each interval, there will explore the automaton's behavior based on the current and previous rate. Furthermore, we will use a network of single automaton to see the effects of the performance of the autonomous web service.

Chapter 5

Phase II: Autonomous web service using multiple learning automata

In the following chapter, we introduce a new concept, an aSpace machine. The chapter starts with a description of an aSpace machine and the description of the working environment. Then the implementation section that present explanations of different operations and functions that have been performed. Finally, the results and observations are provided. The focus will be on creating the aSpace machine and exploring if the machine can recognize the traffic pattern and manage it.

5.1 Introduction

In phase I, we have run different simulations to explore and study the automaton's behavior. We were able to investigate the features of the automaton's behavior: the period of the learning process, the oscillation of the behavior, and the convergence. Furthermore, the results revealed the relationship between the automaton's behavior and the used learning schema, and the values of the learning parameters.

The environment in phase I considered only one dimension, the current rate to explore the automaton's behavior using one single automaton. However, this scenario cannot handle a real-life scenario. In real life, any web server receives a set of incoming requests during the day. The number of incoming requests can vary during the day as well as during the week. For example, a web server for a web store can receive high traffic during the weekends compared to the traffic during the regular days. Also, the

amount of traffic can vary during the day; for example, the afternoon period can have more traffic than the morning and evening periods.

Based on the above, in this phase, we will introduce a new concept to be investigated. This concept introduces an aSpace machine, and the learning automaton represents the fundamental 'learning unit' of this machine. Proposing the aSpace machine aims to address the challenges of learning automaton at phase I, develop a machine that can configure the web server's traffic pattern, and present a suitable learning automaton based on this traffic pattern.

In the following sections, we will introduce the aSpace machine that consists of multi-automata, and these multi-automata are built upon the previous learning automaton from phase I.

5.2 The aSpace machine

In environments with a highly complex nature, it is wise to decompose the environment into a set of dimensions and find out a relationship between those dimensions. Those dimensions can be metadata or inputs of the environment's characteristics used as a classifier. In this way, it is possible to understand and have a level of control over these complex environments. Based on that, the idea of aSpace machine arose. Consequently, building a machine uses the environment's dimensions as its pillars and the learning automata as its learning unit. The aSpace machine consists of inputs, patterns, and outputs. Furthermore, the aSpace machine introduces for the global optima, not the local optima.

5.2.1 Environment

For a reliable web service, two factors should be considered: service availability and response time. In this context, traffic intensity is the essential factor for avoiding latency for a reliable web server. However, traffic intensity, i.e., the number of incoming HTTP requests, is unpredictable; therefore, it will be challenging to handle. Moreover, we discussed the convergence time and how it might affect the response time. To handle this point, we assumed it might be a relationship between the previous rate and the current rate at a specific point. Based on that assumption, we can draw an estimated pattern for the traffic intensity of a web server. We are introducing two dimensions, "time interval and previous rate," to the environment as well as the previous one, " current rate."

1. **Time interval** represents the amount of time between two points. The aim of introducing time intervals is to divide our space into a set of intervals and deal with each separately. We assume the traffic intensity does not show a dramatic change suddenly. So if we can have a set of intervals and each interval has an average of traffic intensity, i.e., pattern intensity. Furthermore, each pattern has a border of two values: the previous rate, which represents the beginning of the interval, while the current rate represents the ending of the interval. The interval period can be represented by a set of hours per day, or it can be regular days and the weekends, or even by season.

2. **The Previous rate and Current rate** represent the second and third dimensions, respectively. The aim of introducing both rates is a belief that there is a relationship between them. Some types of data traffic have a fairly non-stochastic nature, like the data traffic we will use here in phase II. We want to test our hypothesis of whether there is a relationship between the previous and current rate and how we can use it to categorize the traffic.

5.2.2 The aSpace3D machine

After introducing the three dimensions into the environment, we have to think about employing these dimensions to reduce convergence time and deal with the variation of traffic intensity.

From this point, the idea of introducing multiple automata in a single machine. A 3D array represents this suggested machine. The three dimensions of the array are the time interval, the previous rate, and the current rate. Furthermore, the learning automaton from phase 1 is representing *"the learning unit"* of this machine, *"aSpace3D machine"*. Figure 5.1 illustrates the conceptual model of aSpace3D machine.

The proposed mechanism of the aSpace3D machine starts by receiving a request. Then it generates a cell if it does not exist—the position of the cell is based on the values of the three dimensions: time interval, previous rate, and current rate. The generated cell launches a new learning automaton. The convergence here is filling the time intervals.

5.2.3 Scenario

To simulate the behavior of the aSpace3D machine, we used the file *"tf2_50k.dat"*. The *"tf2_50k"* file represents the number of active users for

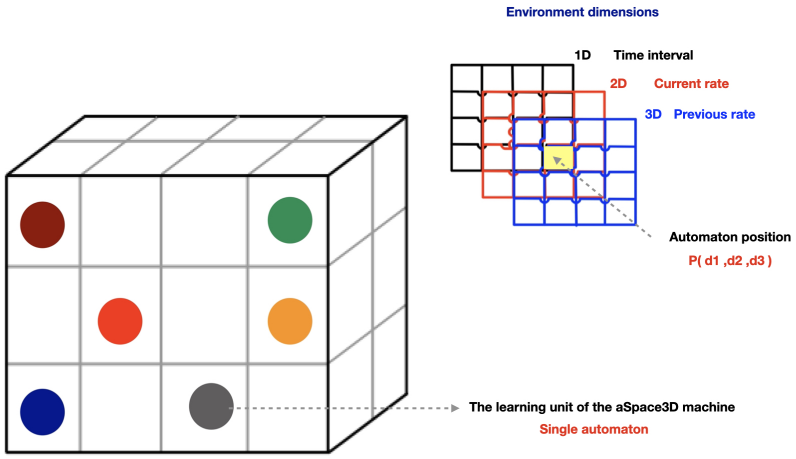

Figure 5.1: *The conceptual model of aSpace3D machine.*

an online game *"team fortress2"*, measured from *"steam.com"*. The data was collected by *Kyrre Begnum* and has been used in other projects such as *Jon-Erik Tyvand* Tyvand, 2011. The data was recorded every five minutes for 173 days. The minimum number of players is less than 1000 players, and the maximum number is up to 45K players. Based on the file characteristics, the values of the three dimensions are:

- Time interval: every five minutes.
- Previous rate: (n) value.
- Current rate: (n+1) value.

The machine convergence should happen once it has a cell for every possible traffic pattern, but it is not necessary for all automaton to be created. The expected number of cells for one day for every five-minute time interval is 24h * (6o min. / 5 min.) = 288. As mentioned above, the minimum number of players during 173 days is less than 1000 players, and the maximum number is up to 45K players. We divided the traffic intensity into a set of categories based on the capacity of the flavors. Therefore, we have five categories for both current and previous rates, as illustrated at table 5.1.

5.3 Implementation

This section presents the python scripts and points out the main functions that have been used to run the experiment. We used the tree python scripts

Flavor	Traffic intensity
F1	handle up to 10k request
F2	up to 20k request
F3	up to 30k request
F4	up to 40k request
F5	more than 40k request

Table 5.1: *The available flavors with the number of HTTP requests it can handle.*

to build the aSpace machine, run the machine, and plot the results.

1. aSpace3D:

   The aSpace3D script is for building the aSpace machine using a set of inputs, selecting the actions, and updating the policy for each automaton. The automaton's location is determined by the time_interval, previous_rate, and current_rate. We are presenting the main function(s), and the detailed script is provided at section 9.4.

   (a) **d1length, d2length, d3length:** represent the length of the three dimensions of our environment time_interval, previous_rate, and current_rate respectively. This function initializes the machine based on these three inputs. These inputs could be any number based on the environment.

```
    def __init__(self, d1length, d2length, d3length,
    name="", a=0.1, b=0.05, verbose=1, history=10):
    #specify the length of the three dimensions
        self.d1length = d1length
        self.d2length = d2length
        self.d3length = d3length
    #loop to build up the machine using the length of
    3 dimensions
        self.aspace = [[[0 for k in range(1, d3length
    + 2)] for j in range(1, d2length + 2)] for i in
                        range(1, d1length + 2)]
```

   (b) **Select_action & update_policy:** These functions use the same code in phase one to select the action and update the policy based on the environmental response. However, the only change is specifying the location of the automaton based on the value of the three dimensions. Also, if the automaton does not exist, it will be created.

```
def select_action(self, p1, p2, p3):
# find the right automata ( and create it if it doesnt
    exist )
```

```
3       if self.aspace[p1][p2][p3] == 0:
4       # automata does not exist. We need to create it
        and update the policy once in order to initialize
        it
5       # the name will mimic it's position in the space,
        so 1,2,4 will be named 1-2-4
6           self.aspace[p1][p2][p3] = Automaton(self.a,
        self.b, str(p1) + "-" + str(p2) + "-" + str(p3))
7           self.aspace[p1][p2][p3].update_policy("None")
8           self.aspace[p1][p2][p3].setVerbose(0)
9           self.automata_count += 1
10          return self.aspace[p1][p2][p3].select_action()
11
12  def update_policy(self, p1, p2, p3,
        environment_response):
13      self.aspace[p1][p2][p3].update_policy(
        environment_response)
14      self.usage_count += 1
15      self.history.append(environment_response)
```

(c) **Print_statistics:** this function is for calculating and printing the count of automata usage and accuracy for the whole machine. We present only how the mean usage and the median accuracy are calculated (see for full code at section 9.4).

```
1   def print_automata_statistics(self):
2       self.out("Automata count: " + str(self.
        automata_count))
3       self.automata_saturation = self.automata_count / (
        self.d1length * self.d2length * self.d3length)
4       self.out("saturation of possible number of
        automata: " + str(self.automata_saturation))
5       automata_usage_count = []
6       automata_accuracy_count = []
7       for p1 in range(1, self.d1length + 1):
8           for p2 in range(1, self.d2length + 1):
9               for p3 in range(1, self.d3length + 1):
10                  if self.aspace[p1][p2][p3] != 0:
            automata_usage_count.append(self.aspace[p1][p2
        ][p3].usage_count)
11                  automata_accuracy_count.append(self.
        aspace[p1][p2][p3].calculate_accuracy())
12          self.out("**Count statistics**")
13          self.out("Median usage: " + str(np.median(
        automata_usage_count)))
14          self.out("Mean usage: " + str(np.mean(
        automata_usage_count)))
15          self.out("**Accuracy statistics**")
16          self.out("15% percentile accuracy: " + str(np.
```

```
    percentile(automata_accuracy_count, 15)))
17          self.out("Median accuracy: " + str(np.median(
    automata_accuracy_count)))
```

2. aSpace-3D-simulation:

   The aSpace3D-simulation script runs the aSpace machine by providing the machine with the traffic data as input, then the machine uses the data to apply the pattern and print out the results. The input data comes from *"tf2_50k.dat"* file (for detailed code see section 9.5).

```
# read data from file
file1 = open('tf2_50k.dat', 'r')
Lines = file1.readlines()

count = 0
last_rate = 0
current_rate = 0
# Strips the newline character
for line in Lines:
    count += 1
    if count > 288:
        count = 1
    if last_rate == 0:
        last_rate = translate_rate(line.strip())
        continue

    current_rate = translate_rate(line.strip())

    action = machine.select_action(count, current_rate,
    last_rate)

    outcome = evaluate_action(action, current_rate)
    machine.update_policy(count, current_rate, last_rate,
    outcome)
#print outputs
machine.print_state()
machine.print_automata_statistics()
```

3. Plot_traffic_data:

   This script aims to plot the players' numbers from the "tf2_50k.dat" file. Plotting the players' numbers helps to understand the nature of the used data. The script contains a set of functions to load data and prepare it, then plotting. We present three functions for the rest functions see section 9.6.

   (a) **load_data_set:** this function reads the data, i.e., the players'

numbers, from the "tf2_50k.dat" file and save it into the data_set variable to be used for plotting.

```
def load_data_set(file):
    data_set = np.loadtxt(file)
    return data_set
```

(b) **get_statistics(data):** The function uses the data_set variable to calculate the mean, median, and standard deviation for players' numbers at each time interval over the 173 days. It returns three arrays of mean, median, and standard deviation.

```
def get_statistics(data):
    mean = []
    std = []
    median =[]
    data = np.array(data)
    for col in data.T:
        mean.append(np.mean(col))
        std.append(np.std(col))
        median.append(np.median(col))
    return mean, std, median
```

(c) **plot_data(data_set, mean, std, median):** the function takes four inputs. The data_set input is used to plot the players' numbers for each day individually. While the rest of inputs to plot the mean, standard deviation, and median for each time interval.

```
def plot_data(data_set, mean, std, median):
    x = np.arange(1, 289, 1)
    colors = []
    for i in range(288):
        if i == 0:
            colors.append('b')
        elif (i % 2) == 0:
            colors.append('b')
        else:
            colors.append('r')
    for i in range(len(data_set)):
        y = data_set[i]
        plt.plot(x, y,'--',linewidth=0.7)
    plt.errorbar(x, mean, yerr=std,fmt="k--",
    errorevery=72,label="mean")

    plt.plot(median,'r-', linewidth=3, markersize=3,
    label="median")
    plt.title('Traffic Intensity')
    plt.xlabel('24H')
    plt.ylabel('Players')
```

```
20      plt.legend(loc="upper right")
21      plt.show()
```

5.4 Results

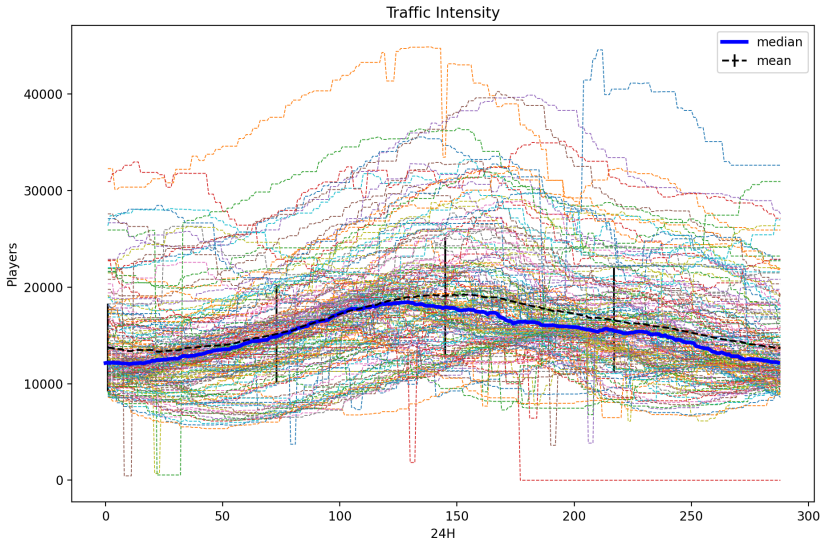

Figure 5.2: *The traffic intensity of an online game website for 288-time intervals per day over 173 days with the mean and median values of the players of each time interval*

Figure 5.2 illustrates the non-stochastic nature of the supported data traffic. It seems that this data set has a symmetric pattern behind this chaos. For example, the intervals from 75 to 120 have a seemingly symmetric distribution, while the intervals before 75 and after 120 have a seemingly asymmetric distributions. Based on these observations and the data nature, we assumed that: the aSpace3D machine might use its total capacity (288 intervals * 173 days)= 49 824 or fewer automata to handle the variation of the traffic intensity. On another side, these variations between the different intervals might affect the accuracy and the usage amount of the automaton. We think that at some intervals, it might end up with the same automaton usage. To investigate these assumptions, we are interested in exploring the status of the aSpace3D machine after simulating. There are two interesting aspects to be studying: (1) the usage count for each automaton and (2) the accuracy of the aSpace3D machine.

After running the aSpace3D machine simulation, we got the following results:

Automata count: 1266							
Usage Statistics				Accuracy Statistics			
mean	median	max	85% usage	1^{th}quart	mean	median	3^{rd}quart
39	20	134	121	0.2	0.67	0.95	1

Table 5.2: *Statistical results of aSpace3D machine using P-model*

Table 5.2 presents the statistical results of the automaton usage and the accuracy of the aSpace3D machine using P-model.

1. **Usage results:**

   The statistical results show the total amount of used automata which was only 1266 automata out of 49824. Hence, it is less than 10% of the total capacity of the aSpace3D machine. This result is inconsistent with our previous assumption of one automaton for each interval over 173 days. Consequently, it reflects the fact of using one automaton more than one time, which is evident from the mean usage value of 40. On the other side, the aSpace3D machine recorded maximum usage for an automaton 134 times and 40 times as an average. Therefore, it proves the second assumption of using one automaton more than one time—this due to having some time intervals with a homogeneous traffic intensity. Furthermore, the other time intervals with heterogeneous intensity have an average lying between 10K to 20K, reflecting the lowness of the data spread. Therefore, we can see that 85% of automaton got used 121 times or less, while 15% of automaton got used 121 times or more.

2. **Accuracy results:**

   The environmental response represents the accuracy indicator of the aSpcae3D machine. By looking at the first and third quartiles, 25% and 75%, respectively, as outliers of usage accuracy, the aSpace3D machine developed its learning process to move from 0.2 at 25% usage accuracy to 0.95 as median accuracy. It then reaches complete accuracy with the total positive environmental responses at the third quartile. Due to the heterogeneity nature of supported data, it took time for the aSpace3D machine to reach a reasonable accuracy level with proves our claim.

   The above results intrigued us to ask what is happening inside the aSpace3D machine, in other words, the experience of each automaton during the learning process of the aSpace3D machine. Therefore, for the rest of the results section, we will answer the following questions,

which will give us more insights into the automata status during the machine learning process.

(a) **What is the likelihood of an automaton getting a "typical" amount of usage?**

Automaton	Usage amount
1-2-2	131
1-3-2	16
240-2-2	127
240-3-2	30

Table 5.3: Automaton usage at different time intervals with different traffic intensity.

Suppose one automaton would be selected every day to handle the traffic at a specific time interval. In that case, the total usage count is 173 times regardless of the traffic intensity. However, this is not the situation because we have variation in the traffic intensity during the different time intervals or even at one interval over the 173 days. For example, table 5.3 illustrates two selected automata abruptly from different time intervals, midnight and 8:00 pm, respectively. The results show that the first interval with the highest automaton usage 131 times out of 173 times, where the current and previous rates were 20K players. However, it was only two days at midnight where there was traffic intensity between 20K and 30K players. On the other side, comparing two automata at two different time intervals, 1 and 240, serving the same traffic capacity (2,2), we can observe the total usage amount. It is almost the same amount of usage 131 and 127 times. Therefore, there is no absolute answer for the provided question; however, the traffic intensity mostly between 10k to 20K. Consequently, there are not many variations between other automata usage amount values at (2,2) in the different time intervals. However, usage amounts vary between different time intervals at the same position, as illustrated in table 5.3.

(b) **How much automaton usage is sufficient in terms of convergence?**

It is not easy to answer this question because more than one factor affects convergence. Table 5.4 shows the learning process

Automaton	Usage count	Action	Action Probability
1-3-2	16	F5	0.5
240-3-2	30	F4	0.48

Table 5.4: The relationship between the amount of usage and the chances of convergence.

of both automata (1-3-2) and (240-3-2). Even though the number of updates for the first automaton is less than the second one, it got F5 with a probability of selection 0.5, which is slightly higher than the second automaton with the probability of selection 0.48. Consequently, it is not the number of updates only. It is also the homogeneity of the data at a time interval over the 173 days. Moreover, whether the automaton was lucky in its selections reduces the number of penalties and increases the chances of a quick convergence.

(c) **What is the likelihood of the optimal action that each automaton will end up with?**

It was clear from phase I that the nature of the automaton is greedy. It was always looking to increase the chances of being rewarded. Therefore, it ended up selecting one of these flavors F3, F4, or F5. Phase two showed similar results, where the selected action at each automaton was F3, F4, or F5, regardless of the intensity of the traffic. It seems strange because the aSpace3D machine was built on instantiating an automaton based on the previous and current traffic. Nevertheless, the supported data has a pattern of traffic intensity between 10K to 20K with some variations above/below these boundaries. Therefore, the possible interpretation is the selected action(s) during the first trials of the automaton. Automaton might select F5, for example, and be rewarded, which increases the probability of F5 being selected again. Nevertheless, some automata selected either F1 or F2 as optimal actions. For example, automaton (3-1-2) selected F1 with accuracy 1 after a total of 25 days of updates.

(d) **What will be the results after introducing noisy and stochastic data to the aSpace3D machine?**

The aSpace3D machine was provided with random data generated using the function *NumPy.random.uniform*. The generated data set had the exact length of the previous data set to mimic the previous simulation but with noisy and stochastic data. At

this step, we have to recall our previous claim, "Data variations between the different intervals might affect the accuracy and the usage amount of the automaton." The results show an increment of automata count =1440 comparing with the previous simulation automata count = 1266. That proves our claim of affecting the amount of used automaton. Therefore, the aSpace3D machine needs more automata to handle the variations in time intervals. Therefore, the higher the variation data is the more amount of automaton used. The accuracy results show that at the first quartal, the accuracy was 0.6, and at 40%, the accuracy was one. Consequently, the machine showed significant accuracy with heterogeneity data comparing with the homogeneity data.

5.5 Observations

The main goal of phase two was to introduce a set of automata to serve the incoming traffic at different time intervals. These automata have been provided via the aSpace machine. After running the simulation and inspecting the results, there were a set of observations:

- Before running the simulation, there was a belief that the aSpace3D machine would assign one automaton for every time interval over the 173 days. Therefore, the total expected number of automata was 49824 automata(288 interval/day * 173 days); however, we got 1266 automata after running the simulation. Furthermore, after introducing a data set with high noise, we got a higher number of used automata comparing with the symmetric data set. Consequently, we can conclude that *"the lower number of automata, the higher the homogeneity data."*

- After investigating the aSpace automata status during the learning process, we found most of the automata got a high accuracy, although they did not converge. It comes as a fact of having homogeneous data, making it easy to settle the automaton quickly. It also reflects the ability of the aSpace3D machine to generate different automata to manage the different traffic intensity, which serves our primary goal of phase II.

- After plotting the raw data, players' average was between 10K to 20K during the 24 hours over 173 days (figure 5.2). Besides, based on our

assumptions (table 5.1), we expected the automaton, which handles the traffic intensity between 10K to 20K, to select F2 or even F1. However, most automata selected F3, F4, or F5. Only a few automata selected F1 and F2 as optimal action. For example, automaton (286-2-2) with the highest automaton usage 134 out of 137 selected F2 as optimal action. The automaton's greedy nature opens the door to re-think whether the automaton selects the optimal action not only in terms of quality of service but also the amount of exhausted resources to achieve that quality.

Therefore, phase three aims to introduce the idea of wastefulness without violating the quality of service constraints.

Chapter 6

Phase III: Introducing an elastic autonomous self-provisioning web-service

In the following chapter, we present the problem addressed in the previous chapter: over-provisioning. We are aiming to develop a policy to overcome the over-provisioning issue and provide an elastic autonomous webs service. After that, the implementation section presents explanations of different operations and functions that have been performed. Finally, we provide the results and observations section.

6.1 Introduction

In phase II, the idea of introducing the aSapce machine, which uses the *environment dimensions* as its "pillars" and the *learning automaton* as its "learning unit," showed promising results in terms of reducing the convergence time and providing accurate results. Therefore, after determining the working environment dimensions, we developed the aSpace machine using those dimensions and called it the "aSpace3D machine," referring to the environment's dimensions numbers. The aSapce3D machine aimed to recognize the traffic pattern and provide multiple-automata to manage the variations in time intervals.

The provided results in phase II were evidence of the aSpace3D machine's ability to enhance the performance by recognizing the traffic pattern of the webserver over 173 days and generate multiple automata to handle the different traffic intensities at different time intervals. Thus, the aim of phase II has been achieved successfully. However, after

investigating the automatas' learning history, we found that most automata selected F3, F4, or F5 as optimal action even though the traffic intensity was either under 10K or between 10K to 20K. That means there is over-provisioning which leads to wasting and exhausting the limited available resources. This observation directed our attention to investigate the automaton's greedy nature and how it affects its decision during the learning process.

Therefore, phase III aims to investigate the amount of exhausted resources by the automaton to serve the incoming traffic. Through phase III, we are aiming to develop a policy to avoid under-or over-provisioning. The policy will consider the number of exhausted resources compared to the suitable amount to serve the traffic intensity on one side. The other side deals with not violating the quality of the service constraints.

6.2 Proposed method for efficient resource provisioning

Successful adoption of cloud computing systems requires rational usage for the limited available resources. Therefore, elasticity is one of the vital features of the cloud computing system. Elasticity refers to growing and shrinking the virtual resources dynamically according to users' demands Bharanidharan and Jayalakshmi, 2021.

In phase II, the automaton behavior did not show the elasticity feature; however, the automaton could grow with the high workloads but not return, i.e., shrink, with low workloads. In other words, the automaton has a scalable feature, not an elastic feature. It is worth mentioning that there is a difference between scalability and elasticity. Scalability refers to growing the virtual resources with a high workload but not shrinking during the low workloads. That results in exhausting the available resources even though it grantees not violating the quality of services constraints.

Al-Dhuraibi et al., 2017 proposed the following equation, which illustrating the elasticity concept (see figure 6.1):

$$Elasticity = \underbrace{scalability + automation}_{\text{auto-scaling}} + optimization$$

Figure 6.1: *Elasticity concept* Al-Dhuraibi et al., 2017

Based on this equation, complicated algorithms are developed to

optimize resources by predicting the workload and providing appropriate resources in advance, such as machine learning algorithms. However, by using learning automata, we will explore the ability of learning automata to optimize its behavior regarding over-provisioning and investigate whether the automaton will be able to develop its policy about resource provisioning optimally. Based on that, we will present the proposed method for developing an elastic autonomous service and explore the automaton behavior and obtained policy.

Punishing model using S-model:

Our proposed method depends on developing a punishment model using S-model. Here there two essential pillars that the proposed model stands on. The first is using the S-model instead of the P-model. The reason lies behind the ways of representing the environmental response in both models. As illustrated in the two previous phases, in the P-model, two values represented whether the input to the environment was favorable or unfavorable. However, there is one question needs to be introduced: *If we have more than one action that is correct, but one is better than the other, how can we represent that and vise versa for the wrong action?*.

P-model can not answer this question because it provides two absolute values for favorable and unfavorable actions. Therefore, there is a need for another model to answer our question. While the S-model's environmental response is an input set of an interval [0,1], it could be the answer of the question by providing continuous input responses within the interval [0,1]. Accordingly, We want to evaluate the different selected actions in each update based on the optimal action that should be taken based on the current workload and the actual taken action. Consequently *the environmental response will be a relative value based on the taken action and the optimized action*.

The second pillar is that the environmental response will follow the punishment system. So our message for choosing to work with a punishment system, not a rewarding system, is: *"there is one right choice; however, there could be more than one wrong choice, and it is better to distinguish"*. Hence, the proposed model will punish based on two things. The first is: the selected actions with lower flavor will be punished harder than the higher one. The second is: the punishment value will be based on the distance between the optimized flavor and the selected one.

The working mechanism of the proposed punishing-model:

In the proposed model, we distinguish between three situations based on the selected flavor comparing with the optimal one. The available

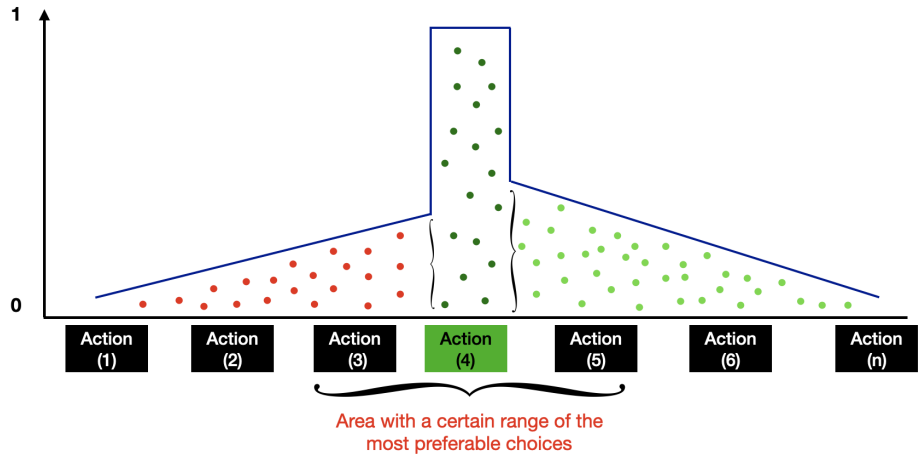

Figure 6.2: *The mechanism of the proposed rewarded model where F4 is optimal choice for the case*

actions vector is divided into three areas: the higher flavor area, the lower flavor area, and the optimum area. Assume action(4) is the optimum action based on the current workload. Based on that, figure 6.2 displays three areas where the left area represents the lower flavors area, while the right area represents the higher flavors area, and the middle is the optimum area. Although the left area is slightly greener, there is a considerable increase on both sides. Further,the punishment system does not go from the bottom to the top dramatically, but it goes from a certain point. The aim lies behind this is to gain some convergence by stabilizing the convergence to an area to be a center range of choices. By doing that, the automaton will not gravitate too much towards either side and stabilize towards the optimal area; consequently, it can be converge rapidly. This previous punishment system that allows this gravitation and distinguishes between both sides. In other words, we do not want symmetric values between both areas. The following paragraph presents an example to illustrate the mechanism of the proposed model (see figure 6.2).

Assuming the optimal action based on the current workload is action (4). To distinguish between the lower flavors area and the higher flavors area, *damage factor (f)* will be used. Moreover, the punishment model will be less forgiving with the lower flavors while more forgiving with high ones. Based on that, the damage factor f1 denotes *the higher flavors area* while f2 denotes *the lower flavors area*. Based on that, f1 $>$ f2 and f $\in$[0,1]. Moreover, to force the automaton to gravitate towards the areas near the optimum area, *a distance factor (d) is used*. The (d) factor represents the distance between the optimum action and the selected action and the degree of

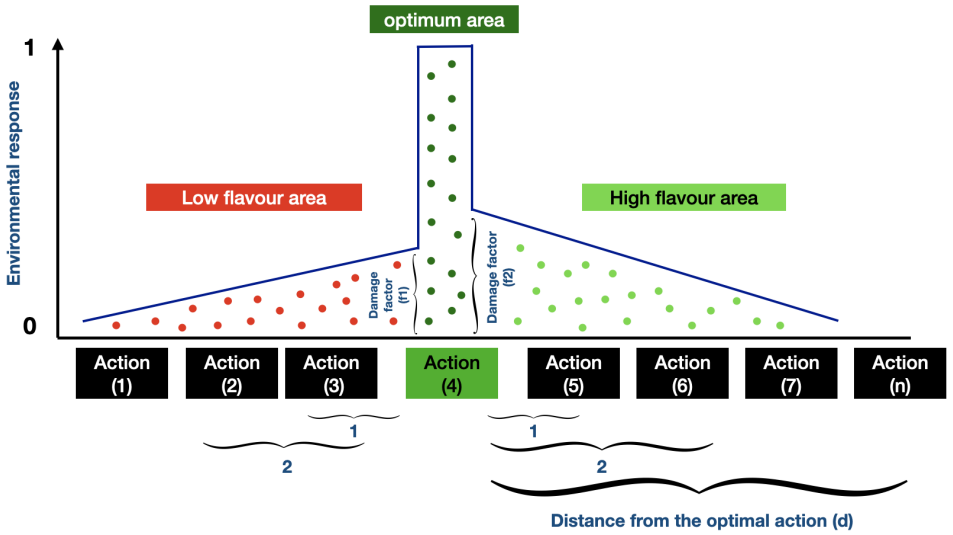

Figure 6.3: *Proposed punishment model*

the punishment will be based on this distance. Based on the above, the environmental response is represented by the equation:

$$r(t) = \frac{f}{d} \tag{6.1}$$

where r(t) denotes to the environmental response at time (t).

If a low value has been chosen as a start value for f, then the lower starter has been chosen, and the difference between the two areas of flavors will say something about the relative seriousness and damage of each side.

Finally, the probabilities of the available actions will be changed, and the new probabilities will be calculated using the equation 2.5 of P-model.

For more illustration of how the environment response will be calculated; we will use the same above example.First we have to determine the damage factor values, the lower flavor area f1= 0.2 and for the higher area f2= 0.1 (as mentioned before, the lower area will be punished harder than higher area). Then assume the first action was action 5 (F5) then the environment response should be = 0.1/(5-4) = 0.1 /1 = 0.1. If the second action was action 7 then the environment response should be = 0.1/(7-4) = 0.03. So the automaton get less rewarded when it selects actions far from the optimal one. Same will happen with other side (higher flavor side). Finally if the action was right then the response will be 1.

In the following section, the implementation of the model is presented.

6.3 Implementation

This section presents the python scripts and points out the main functions that have been used to run the experiment. We used the same python scripts to build the aSpace machine, run the machine, and plot the results.

1. **update_policy function**: we did minor changes on the automaton script, specifically the update_policy function to apply P-model formula inside of S-model formulas:

```
def update_policy(self, environmental_response):
        # if it is first iteration then the selection will
     be random and all actions will have equal
    possibilities
        if environmental_response is "None":
            number_of_actions = len(self.available_actions
    )
            initial_prob = 1 / number_of_actions
            for policy_key in self.prob_policy.keys():
                self.prob_policy[policy_key] =
    initial_prob
            return
        for key, value in self.prob_policy.items():
            if key == self.selected_action:
                # apply p_i formula on the chosen action
                self.prob_policy[key] = value + a *
    environmental_response * ( 1 - value) - b * ( 1 -
    environmental_response )*value
                #self.prob_policy[key] = value + self.a *
    (1 - value)
            else:
                # apply p_j formula on all other actions
                self.prob_policy[key] = value - a *
    environmental_response * value + b * ( 1 -
    environmental_response ) * ( 1/(len(self.available_a$
#                self.prob_policy[key] = (1 - self.a) *
    value
        self.prob_policy = self.trim_prob_policy(self.
    prob_policy)
        self.history.append(environmental_response)
        self.usage_count += 1
```

2. **environment_response.py**: we developed a new python script represents the environmental response to implement the developed punishment algorithm. We specified the values f_lower if taken action was lower that the optimal one. f_higher if taken action was higher that the optimal one. Also, we specified the values of both 0.1

and 0.2. The function "evaluate_action" to evaluate the taken action and generate the environmental response. The distance is calculated by "optimal action (current_rate) - taken action (capacity)" and if the taken action was the optimal one then the value will be 1.

```
    import re
class flavorEnvironment:
    def __init__(self,name,f_lower = 0.1, f_higher = 0.2):
        self.name = name
        self.f_higher = f_higher
        self.f_lower = f_lower
    def evaluate_action(self,action,current_rate):
        # lets figure out what flavor was chosen and
    calculate the "capacity" of this flavor
        capacity = int(re.search(r'\d+', action).group())

        if capacity > current_rate:
            # Too high, we are wasting resources. apply
    f_higher
            response = self.f_higher / ( capacity -
    current_rate )
        elif capacity < current_rate:
            # Too low, we are damaging the service. apply
    f_lower
            response = self.f_lower / ( current_rate -
    capacity )
        else:
            response = 1

        return response
```

6.4 Results

After applying the P-model and the punishment algorithm, we could ask, **"What level of degree did it affect on the statistical results of the aSapce machine?"**.

Automata count: 1266							
Usage Statistics				Accuracy Statistics			
mean	median	max	85% usage	1^{th}quart	mean	median	3^{rd}quart
39	20	134	121	0.103	0.49	0.45	1

Table 6.1: *Statistical results of aSpace3D machine using S-model*

Table 6.1 illustrates the statistical results, and there is no change in usage results comparing to the results of phase II (see table 5.2).

Applying different models does not affect the mechanism of instantiating the automatons on the aSapce machine because the instantiation depends on the existence of the automaton in a time interval that can handle a specific current based on a previous rate. However, when looking at the accuracy results, the of value at the 1st percentile and the median, we find the accuracy did not reach even 0.5, while it reached one in the 3rd percentile. That can be explained by how severe the aSapce machine becomes with selecting the automaton to the different flavors. Therefore, it needs time from the aSpace machine to reach complete accuracy with the total positive environmental responses. Also, another reason that could affect the accuracy results is the initial choice of the automatons. For example, assume an automaton has its position in aSapce machine <1-2-1>, which means an automaton manages the first interval, with current load 2, and with previous load 1 and the first selection was F5. The first choice was far from the optimal action (F2), and therefore, it will take some time to converge towards the optimal action and it depends on how much the automaton will be lucky in its first selection until gets the optimal one.

However, we argue that the developed punishment system that we proposed helps to gravitate the automaton towards the optimal action or at least towards the area with the preferable choices. If we look back to the previous example: we argue that most of the automaton choices will be around F1 (represent the lower flavors area) and F3 (represent the higher flavors area) based on the developed punishment system (see figure 4.2). Based on that, the automaton can gravitate towards the preferable range of actions, enabling the automaton to enhance the learning experience by reaching optimal action or at least the sub-optimal actions.

The above results intrigued us to investigate our claim whether the punishment algorithm achieved the planned goals of gravitation or not. Also, we need to investigate whether using S-model contributes to reducing wastefulness or not. Therefore, the rest of the results section will be dedicated to answering these questions, which will give us more insights into the automata status during the learning process. It is worth to state that the selected automatons are representative for similar automatons from the previous phase. Moreover, we reviewed the entire data set of the aSpace machine and not just the selected automatons before we reflect our observations.

1. **What is the likelihood of the automaton gravitating towards the optimal action and towards the area with the preferable choices?**

   To answer this question, we selected two automatons with two

different usage numbers. Table 6.2 illustrates the automatons position in aSapce machine and the number of usage.

Automaton	Usage amount
288-2-2	131
1-1-2	25

Table 6.2: Automaton usage at different time intervals with different traffic intensity using S-model.

We traced the history of those automatons through their learning process to figure out how many times each flavor was chosen and how far were the most chosen flavors from the optimal action. Figure

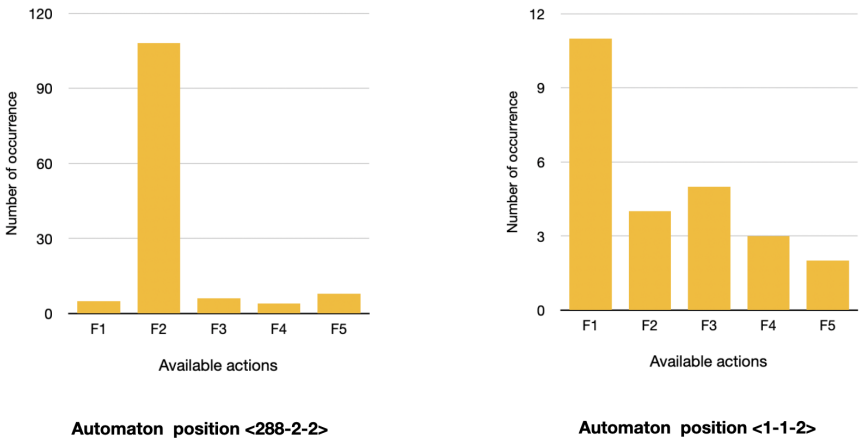

Figure 6.4: *The number of actions occurrence of each automaton*

6.4 illustrates the results of how many times each action has been selected. By taking a look at the first automaton <288-2-2>, we can see the optimal action should be F2 based on the interval's current rate (the middle number). F2 was selected 108 times out of 25 times (82%), while both of the nearest actions, F1 and F3, were selected five times and eight times, respectively. The farthest action is F5 and was selected two times. At the same time, the second automaton <1-1-2> the optimal action should be F1 based on the interval's current rate (the middle number). F1 was selected 11 times out of 25 times (44%), while both of the nearest actions, F2 and F3, were chosen four times and five times, respectively. The farthest action is F5 and was selected two times.

There is a set of observations we can state here. First, in both automatons, the optimal action was the most selected by the automaton. Second, in the automaton <288,2,2> the nearest neighbors are F1 and F3, and both showed a high number of selections comparing to F4, but still, F5 shows a high number of selections comparing to F3. Another thing is that the number of selections for lower flavor capacity F2 is less than higher flavor capacity F4 (gravitate more towards high flavor area, see figure 6.3). The same happened with the automaton <1,1,2>, the optimal action F1 was the most action was selected. Also, the nearest neighbor is F2 with four selection times, which is greater than F4, and F5 is the farthest neighbor. However, F3 was greater than F2,even though F2 is the most near to F1; but the difference is not that much (only one time).

Based on that, the automaton in both cases, regardless of the number of usages, selected the optimal action. Moreover, they reduced the wastefulness by selecting the nearest actions without being under- or over-provisioning.

2. **Is the number of occurrences of the optimal action enough to evaluating the service behavior towards optimization?**

   The number of occurrences may be a good indicator to see how many times the optimal action was selected. However, it is not enough to decide whether the service was able to optimize its behavior through its learning process or not. Therefore, we have to investigate two things. First, the first and the last actions to see the evolution of the learning process of the automaton. Second, see after 173 days the probability distribution of the automaton. We will use the same previous automatons.

Probability distribution									
Automaton <288-2-2>					Automaton <1-1-2>				
F1	F2	F3	F4	F5	F1	F2	F3	F4	F
3.27E-06	0.99	4.08E-06	3.28E-06	3.01E-06	0.67	0.10	0.08	0.08	0.08

Table 6.3: *The probability distribution of Automatons after 173 days*

Table 6.3 illustrates the probability distribution of the automatons after 173 days. The first automaton <288-2-2> shows that the probability of F2 is 0.99, the optimal action in this case; it almost reaches one. While the automaton <1-1-2> shows that the probability of F1 is 0.67, the optimal action in this case, and it is the highest

probability compared with others.

On the other side, when we take a close look at the automaton <288-2-2> behavior through its learning process (see table 6.4), we find the development in its behavior through the last ten updates where all selections were F2 comparing with the first ten updates where the selections were still random.

First occurrence	f3	f4	f5	f5	f5	f4	f1	f2	f2	f5
Last occurrence	f2	f2	f2	f2	f2	f2	f2	f2	f2	f2

Table 6.4:
The Automaton <288-2-2 > through learning process using the S-model

Also, the automaton <1-1-2> shows development in the behavior of the last five updates where all selections were F1 comparing with the first five updates where the selections were still random (see table 6.5).

Based on that, we can see the improvement of the automaton through its learning process. However, for the automaton <1-1-2>, there were F3 and F2 in the automaton´s selections. On the opposite, the automaton <288-2-2> all the last selections were the optimal actions. That leads to thinking about the convergence of each automaton. The next question will investigate that.

First occurrence	f4	f1	f2	f5	f3
Last occurrence	f3	f2	f1	f1	f1

Table 6.5: The behavior of automaton <1-1-2 >through the its learning process

3. **How much automaton usage is sufficient in terms of convergence?**

Automaton	Usage count	Action	Action Probability
288-2-2	131	F2	0.99
1-1-2	25	F1	0.67
1-3-2	16	F3	0.38

Table 6.6: The usage and the chances of choosing the optimal action using S-model

To answer this question, we selected the same two automatons. Then,

we selected one more with lower usage to see the convergence value compared to the number of usages. As illustrated in the table 6.6 the first automaton with 131 usages has F2 with a probability of selection 0.99, while the middle automaton with 25 usages has F1 with a probability of selection 0.67, and the automaton with 16 usages has F3 with a probability of selection 0.38. Based on that, the number of usages or updates of the automaton increases the learning experience, consequently increases the chances of convergence. The good part here is, the optimal action in all presented automatons has a higher probability of selection. That represents the achievement of using the S-model with the punishment system to reduce wastefulness and enhance the automatons learning experience.

4. **Does introducing the S-model and the punishment algorithm affect the aSapce machine in terms of accuracy comparing to P-model?**

   We plotted the environmental response for each time interval (from 1 to 288) through the 173 days to investigate the accuracy of all time intervals.

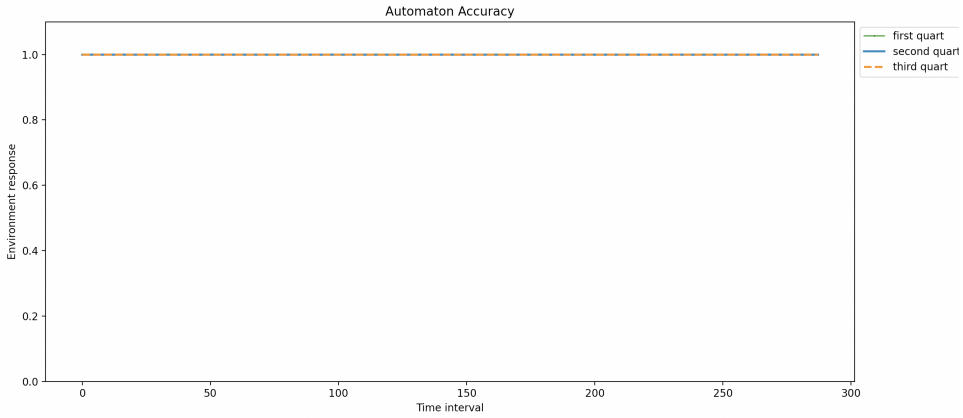

Figure 6.5: *Accuracy of aSpace machine using P-model*

Figure 6.5 shows the accuracy of the first, second, and third quartiles of each time interval over 173 days for aSapec machine using the P-model. The aSapces machine shows high accuracy from the first quartile.

While figure 6.5 shows the accuracy of the first, second, and third quartiles of each time interval over 173 days for aSapec machine using the S-model. The aSapces machine does not reach accuracy until the third quartile.

That means the aSpace machine using S-model needs more time

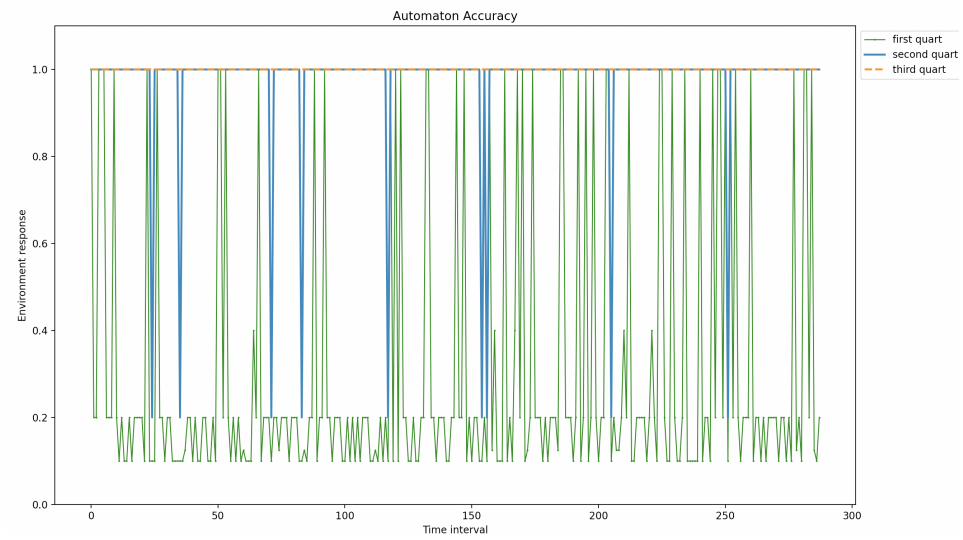

Figure 6.6: *Accuracy of aSpace machine using S-model*

for convergence than aSpace machine using S-model. That can be interpreted as the values of environmental response where the P-model uses only 0 or 1 while the S-model uses values from the interval [0,1]. Moreover, when the aSpace machine used the P-model, there is only one absolute answer, optimal or non-optimal, regardless of the taken action wasting resources or not. However, the situation is different in the S-model, where the selection of the actions is evaluated whether it is optimal or not and how far it is from the optimal action. Based on that, the aSapce machine using S-model may take some time to reach a high accuracy compared with the aSapce machine using the P-model.

However, the convergence of the aSpace machine is not the only essential factor to be considered; we have to consider the selected actions by each automaton for service optimization. When we introduced the service optimization, we considered the optimization from the customer perspective (cost, response time) and the provider perspective (not wasting the limited available resources). Based on that, we have to investigate the automatons' status in both models.

5. **Comparing the automaton´s convergence status in the P-model and the S- models.**

   We will use the same automatons examples that we used in the previous phase (see table 5.4) in this phase (see table 6.7).

   When comparing the automaton <1-3-2> in both models, we will

Automaton	Usage count	Action	Action Probability
1-3-2	16	F3	0.38
240-3-2	30	F3	0.28

Table 6.7: The amount of usage and the chances of convergence using S-model.

find the probability was 0.5 after 16 updates in P-model, which is higher than the 0.38 in the S-model. However, the action was F5 in the P-model while it was F3 in the S-model. At the same time, the automaton <240-3-2> in both models will find the convergence was 0.48 after 16 updates in P-model, which is higher than the 0.28 in the S-model. However, the action was F4 in the P-model while it was F4 in the S-model.

Based on the above results, the automaton shows higher values of convergence in the P-model comparing to the S-model. However, the selected action in the automaton <1-3-2> was the optimal action F3, while in the P-model, it was F5. Also, in the automaton <240-3-2>, the selected action in the automaton <1-3-2> was the optimal action F3, while in the P-model, it was F4.

Based on the above results, the automatons showed higher values in term of convergence when using P-model. That means using S-model needs more time to converge.

6. **Comparison between the learning evolution of the automaton in both models.**

   To investigate the effectiveness of introducing the S-model on the learning process of the automaton, we will select the most usage automaton <288-2-2> and compare the first and the ten updates with the same automaton using the P-model.

First occurrence	f2	f2	f1	f1	f1	f1	f2	f4	f1	f1
Last occurrence	f2	f2	f2	f2	f2	f2	f2	f2	f2	f2

Table 6.8: The behavior of automaton <288-2-2 >through the its learning process using P-model.

The table 6.8 represents the learning experience of the automaton using the P-model; we used the results that we got from the previous phase II. While table 6.4 reflects the learning experience of the

automaton using the S-model.

Using the P-model (table 6.8) shows the first ten updates where the most updates were F1 and F2, then one time F4; while the last ten updates were F1. The automaton was lucky in its first selections between F1 and F2; therefore, it can reach the optimal action during the last ten updates F1. Maybe that was not the case with others automatons, especially if the first selections were higher than the optimal one, as we saw earlier. On the other side, the table 6.4 shows the first ten updates where the most updates were shifting between the actions from F1 to F5. However, in the last ten updates, the automaton reached convergence to the optimal action.

6.5 Observations

The main goal of phase three was optimizing the service performance in terms of wastefulness. After running the simulation and inspecting the results, there were a set of observations:

1. Introducing S-model and the punishment algorithm enabled to reduce the wastefulness without affecting the response time. We observed that on the automaton´s actions during its learning process, the automaton has the highest probabilities centered in the favorable area that we specified in our model.

2. After investigating the aSpace machine and the status of its automatons, we figured out that the aSapce machine and the automatons using S-model need more time to convergence comparing with the aSapce machine using P-model.

3. Although the automatons need more time for convergence, the optimal action probabilities of the automatons were high, reflecting the enhancement of the automaton learning experience.

4. The ability of convergence depends not only on the time it needs to converge but also on how the automaton was lucky in its first selections. That helps the automaton to increase the optimal action probabilities.

Chapter 7

Discussion

7.1 Answering the problem statements

The first problem statement states introducing AI into service management by designing a learning automata-based model (P1). The design of the model was developed through a set of phases. In phase I, the initial model was the structure of the autonomous service using the learning automata algorithm as the learning unit of the service. Then, based on the observations from the first phase, the model was expanded to consider different elements, the dimensions of the environment, and how to overcome the convergence issue. Based on that, the design of the aSapce machine model was introduced in phase II. Finally, in phase III, some enhancements were introduced in the model; based on the observations from the previous phase. The enhanced model aimed to reduce the waste of resources by using S-model and the punishment algorithm.

Based on the above the service management was achieved by designing the learning automata-model for optimizing resource management.

If we returned to the problem statement (P1), we would rephrase "learning automata-based model" to "Reinforcement learning automata algorithm." When we developed the statement, it was unclear what we will end with; therefore, we chose the "model"; maybe at that time, it meant something but now no. Therefore, using "algorithm" describes what we ended up with much better.

The second problem statement represents the implementation of the designed model. The implementation process was done by simulating the behavior of the autonomous service through the different phases. Through the simulation, we explored the behavior of the automaton through its learning process and then exploring how the service behavior was changed

after implementing and simulating the aSapce machine. The results were surprising regarding the ability of the service to adapt its behavior based on the surrounding environment.

The aSapces represents the actual implementation of the designed model. It uses the learning automaton as its learning unit and the environment dimensions as its pillars. The aSpace machine has proved its ability to optimize the service performance autonomously by providing the optimal action based on environment circumstances.

By returning to the formulation of P2, we would remove one of "developing and implementing" cause both give the same meaning. Further, we may use "experiment" or "simulation" for the prototype and "evaluation" for that prototype because any prototype needs to be evaluated. Based on that, a better formulation would be "Implementing, Simulating, and Evaluating the prototype based on the designed model."

Even though we got promising results from the aSpace machine, it still needs more investigation under different circumstances. Also, it can be expanded more by adding new features to strengthen its functionality.

7.2 Planning the exploration process

In planning this project, we had to think about the project's nature and the most suitable approach. In the approach chapter, we described the project as an exploratory project and divided it into a set of phases. Each phase had a question to be answered or an objective to achieve. However, the nature of exploratory projects does not provide a clear line of achieving a complete end results. This kind of project implies a set of challenges and ideas.

What if the next step is not clear after the phase? What if one feels lost after the phase and needs to return and start over, or there is no connection, and the phases do not build top on each other? However, it did not happen in this project, but this does not mean we did not have these concerns while working.

What if one overestimates the size of each phase? The phases become more extensive than they need. OR Is this phase sufficiently covered, and are we ready to go to the next chapter? Ironically, everything one wants to happen, and every goal one wants to achieve, was also the source of the most frustration. Our initial thoughts were that we wanted new ideas, but we were not prepared for the frustration of saying no to these new ideas. So if one wants to do a new project and follow the same way, one needs to be mentally prepared

to look at these new openings as successes as we want to be open, and saying no to these openings is not necessarily negative. It is easy to be more frustrated because one is used to working systematically on everything that is opened in front of ones eyes.

Is the amount of work enough? The exploratory project is not effective in producing the results. One never sees the goal line that one has to cross and only sees long series of small tasks that need to be complete. It might be easier to have one big task and know what one has to do. It is straightforward, and there is an end.

So we can say that the exploratory project may fit more with the long thesis or a Ph.D., and is not fully compatible with the short thesis. However, the time in the short thesis is valuable comparing with the long thesis.

7.3 The road towards achieving results

The road towards achieving the final results was full of difficulties and uncertainties. In the following paragraphs, we will discuss the most important of these difficulties and efforts that have been exhausted. The project was divided into a set of phases, and each phase had its objective to help reach the final goal, "an autonomous web service." However, before going through these phases, it is important to address the difficulty of integrating a different discipline (AI) in our own problem domain. This step was crucial because the whole project was built upon this step.

The road before starting the phases:

Specific formal terms in AI need to be put in our field of systems operations. In that sense, we need to find local representations of those terms in our paradigm. For example, the main terms in learning automata that anyone needs to know before implementing this algorithm are environment, reinforcement signal, states, transitions, and actions. If we take *"the environment"*, we had to define the meaning of environment in our case. Does it mean the physical area (VM or something else) where to place the automaton, or does it mean the customer? Another example describing the connection between the automaton and its environment is *"the environment sends a reinforcement signal to the automaton."* What kind of signal?. It required spending time and effort to understand and model these terms into our paradigm. It was a challenging process to accomplish, and it needed one to understand the meaning of the original term then mapping it to a local meaning in the operation field. Despite these challenges, we can

say it is exciting and can it sometimes feels like it is completely uncharted territory that everything is open and needs more work to explore.

Reading, and understanding another field on ones own is challenge. One of the main reasons for these difficulties was the lack of examples of transforming the learning automata terms in the operations context in the relevant research papers. They could explain the developed algorithm without illustrating the concepts like environment or reinforcement signals explicitly. Therefore, it was not clear to us how they approached their algorithms and models. Further, each one translated these concepts to fit their own problem domain. Based on that, we had to read the original Ph.D. thesis Velusamy, 2018 to see how they explained the learning automata algorithm from the operation side in details. Moreover, we reviewed some technical papers in learning automata (Nowé et al., 2005) and (Narendra and Thathachar, 1974) and read the introduction chapter and pick some sections from other chapters of the book "Learning automata: an introduction" (Narendra and Thathachar, 2012) to get a better understanding of the algorithm and its different models. It was a challenge to understand and absorb these concepts in a limited time. However, it was the main support for us to read through the different technical stuff of the algorithm such as different classifications which helped us decide whether to use a stochastic or deterministic automaton. Also, another critical classification was the different environmental responses and their different models. Consequently, it helped us to think critically to select and implement the algorithms that suited our problem domain.

The processes towards the results:

The translation of an abstract model into a particular context was challenging. We read so many approaches that the others have done, and we could not directly apply them. The way other papers approach their project is different even though they are targeting the same or similar problems. For example, both (Ranjbari and Torkestani, 2018) and (Misra et al., 2014) proposed learning automata algorithms for optimizing service performance in terms of resource management. However, the context of introducing the learning automata was different. Ranjbari and Torkestani, 2018 used a learning automaton to predict whether the physical machine will be overloaded or not based on the average of the total CPUs usage of that machine. Then if the machine will be overloaded, one VM or more will be migrated to another physical machine. Misra et al., 2014 used a learning automaton as a learning system to select the suitable resource target to serve the user requests based on the maximum job success rate of each.

We were inspired by how they defined the automaton's actions, inputs, outputs, and how defining surrounded the environment. Both were using the learning automata in a deterministic way. Therefore, it needed to be transformed to fit our situation. We introduced service performance based on incoming workload; therefore, we had to figure out how to implement the automaton stochastically.

There is no standard way or a clear outline of using and implementing AI into the operations field. In that sense, we had two ways to implement the learning automata algorithm, either using an existing engine from a python library or develop the algorithm from the ground using our own code. We decided to develop the algorithm from the ground up. The first reason, we could not find a python library that fits our domain problem. We found just a few libraries that were developed deterministic automatons, not the stochastic ones. Moreover, there was one question on our mind at that time *"If we found a python library for agent-based learning automata and adapt it into our environment, would it give us accurate results?"*. As an exploration, we wanted to avoid a black-box situation where we could not investigate what happened along the way. Even though it was challenging to translate the automaton structure based on the AI terminology to code and the different models to represent the environment response, it was exciting to figure out how to do that. Moreover, developing the algorithm from scratch gives us more control over the problem domain.

What do we mean when we say that we are using AI? There is no standard when someone says, "I have used AI", it is not like the next one using AI. There is no standard way to implement it, at least in such a way that we can easily take their way and replicated it in our case. Further, there is perhaps a knowledge gap, and once this knowledge gap is closed it would be straightforward to understand what someone means when they say we are using AI in our field. In that sense, we could expect to have a new buzzword *AIOps* like *DevOps*. That represents the professions who can do the translation for us.

7.4 Proposing aSpace machine and related work

The aSpace machine has been developed through different phases due to the observations of the previous phase. However, there is more to be explored and investigated. Through the journey, some questions arosed and need to be answered:

What is the contribution of the aSpace machine in our field? After analyzing

the results and our observations, we can say it provides a simple and straight forward way to reuse it in other contexts. The aSapce machine works well on time-series data; however, it does not require it. It seems to be well suited to the type of data we used all the time. Moreover, it allows us to have arbitrary length of non-stochastic memory. Also, it offers a way to expand the environment dimensions based on the context of the problem to fit more than three dimensions. In theory, it can provide an arbitrary amount of combinations. It is effective when we use it with only three dimensions, and two of them are the same data type. It can be used with some configuration to other time-series data, so it is quite adaptive due to it is the simplicity of configuration. So, it can be reused in many situations, but that requires a much deeper understanding of the problem domain first.

What has been achieved by developing aSpace machine? The aSpace machine provides a new way to solve one of the fundamental issues related to embracing learning automata; *Convergence* and *Optimal policy*.

1. *Convergence:* it provided a set of automatons; each can handle a different workload. That was built based on recognizing the behavior of the time-series data. For example, is this data is homogeneous or heterogeneous? We adjusted the five minutes between each interval to reduce the noise level by current and previous rates, then assigned them as dimensions for our aSpace machine. In that way, we were able to provide a solution to the convergence problem.

2. *Optimal policy:* During the learning process, the auto automaton develops a policy that has been used until it reaches convergence. This opens up the question: is the obtained policy the optimal one? This question is crucial because we do not know the best solution, especially with the high complexity problems like resource allocation or VM capacity. Therefore, it is essential to think of this question while using the learning automaton algorithm. In the last phase, we tried to answer this question using the S-model and developing a punishment/reward system. The idea of that system is that "if we cannot reach optimality, at least we can reach sub-optimality." Using a P-model provided that. However, many types of research use the P-model for its simplicity, but it provides only an absolute yes or no, which does not fit with the nature of the cloud environment. P-model does not get that much attention even though it is a powerful way to be adapted to circumstances where a nuanced answer is required.

There may be some risks during the implementation. *What if our results are influenced by errors we did not discover?* If we built everything in one

goal without phases and the aSpace was everything, we would run the experiment and the sum of complexity would be significant, and the logs would likely not make any sense. It will be like a black box. But the way the project was organized as phases was beneficial because we dealt with the complexity by building every piece and continuously testing it with the same cases. That meant we had a better position to examine the things if something started to behave weirdly. Also, we saw improvements along the way because we still understood what was going on at each phase. That is happened because we started building the aSpace machine from the ground up. But what would happen if we used some python library that has everything. Then we just put the data in and get the answer out. So the phase design may have been a hidden way to prevent losing control of what the aSapce machine was doing along the way. Also, reviewing the results at every step and keeping a very close eye on every stage manually gave us a good position to believe the results.

We think the results are believable because we approached it piece by piece, and the results were manually inspected by time. So we can have some confidence in the produced results.

7.4.1 aSpace machine and related work

In our study, we used the learning automata algorithm to optimize the service performance by selecting the optimal action based on the incoming workload without under-or over-provisioning. While Maurer et al., 2012 proposed an algorithm using the rule-based knowledge management (KM) approach to avoid under- and over-utilization of available resources based on threat threshold (TT). Both studies achieved the same results that prove the effectiveness of the learning automata used in our study.

Ranjbari and Torkestani, 2018 proposed an algorithm using learning automata for resource utilization. Their study used the P-model to predict only the overloaded physical machine(s). In our study, we were able to provide solution for the under-or over-provisioning by using the S-model and the punishment/reward algorithm.

7.5 Future work

In this thesis, we have developed the learning automata-based algorithm for implementing the aSpace machine model. Our goal was to enable an autonomous service to be self-adaptive based on the workload. We achieved our goal; however, we believe many new stories can be told.

Therefore, there are potential future directions the project paves the way for.

One of these situations is: the individual automatons with little experience need a way to reach convergence. That could be achieved by developing an inter-communication way between the automaton and its nearest neighbors. Then, based on that communication, the neighbors can vote for an action based on their previous experiences. Based on that, the problem statement could be stated as:

"Developing an inter-communication system for enhancing the learning process of the automatons with little experience."

Chapter 8

Conclusion

The goal of the thesis was to explore introducing the learning automaton algorithm with cloud operations to achieve an autonomous web service. In order to address the problem statement both in theory and practice, a model has been designed, formal terms of AI have been modeled in operations terminology, a prototype has been implemented based on the designed model, and the proposal of aSpace machine as solution was discussed.

The model design has been developed through the different phases of the project. In the initial phase, the model was confined to a scalable automaton to handle the incoming requests. Then, the initial model was developed through the second and third phases based on the previous phase's results and observations. The developed model was found to be a good representation of the aim of each goal, and it paved the way for the next phase to develop upon it. Moreover, the developed model can be expanded to cover more explorations that are not uncovered yet.

The prototype was created in the same way as the design model was created. With each new exploration or a problem at each phase, the model was developed, and the prototype was developed as a consequence. These local solutions showcase how the learning automaton evolves from an autonomous scalable web service towards aSpace machine that can manage time-series data.

Future suggestions include exploring aSapce machine, both on a design level and a technical perspective. Another potential work could be implementing an inter-communication system between the automatons inside aSpace that could benefit more complex situations.

Chapter 9

Appendix

9.1 Automaton.py

```
import random
import numpy as np

class Automaton:
    def __init__(self, a, b, name, verbose=1, history=10):
        # The reward parameter can be a = 0.1
        self.a = a
        # The penalized parameter can be b = 0.05
        self.b = b
        # the name of this particular automata instance
        self.name = name
        self.verbose = verbose
        self.history_length = history
        self.history = []
        self.version = "v1"
        # A list of the available actions to choose between
        self.available_actions = ["f1", "f2", "f3", "f4", "f5"]
        # self.available_actions = ["f1", "f2", "f3", "f4", "f5
    " ]
        # A variable to store the selected action after each
    response from the environment
        self.selected_action = None
        # Dict for the policy mapping states to probabilities
        self.prob_policy = {'f1': 0, 'f2': 0, 'f3': 0, 'f4': 0,
     'f5': 0}

        self.usage_count = 0

    # function to get the environmental_response and update
    rule for changing the policy mapping states to
    probabilities
```

```
    # or preferences for actions.
    # if the selected_action is favorable then the environmental_response will be 1
    # or unfavorable then the environmental_response will be 0
    def update_policy(self, environmental_response):
        # if it is first iteration then the selection will be random and all actions will have equal possibilities
        if environmental_response == "None":
            number_of_actions = len(self.available_actions)
            initial_prob = 1 / number_of_actions
            for policy_key in self.prob_policy.keys():
                self.prob_policy[policy_key] = initial_prob
            return
        # if the environmental_response = 1 then reward the selected action and penalized the rest of actions
        elif environmental_response == 1:
            for key, value in self.prob_policy.items():
                if key == self.selected_action:
                    self.prob_policy[key] = value + self.a * (1 - value)
                else:
                    self.prob_policy[key] = (1 - self.a) * value

        # if the environmental_response = 0 then reward the selected action and penalized the rest of actions
        else:
            for key, value in self.prob_policy.items():
                if key == self.selected_action:
                    self.prob_policy[key] = value - (self.b * value)
                else:
                    self.prob_policy[key] = value + self.b * abs(1 / (len(self.available_actions) - 1) - value)

        self.prob_policy = self.trim_prob_policy(self.prob_policy)
        self.history.append(environmental_response)
        self.usage_count += 1

    # function to assign the action with highest probability to be the selected_action at time(t+1)
    def select_action(self):
        # check if all actions have same probabilities then choose one action randomly
        prob_equal = len(list(set(list(self.prob_policy.values()))))) == 1
        if prob_equal:
```

```
63              random_action_index = random.randint(1, len(self.
    available_actions))
64              self.selected_action = self.available_actions[
    random_action_index - 1]
65          else:
66              # change this one
67              self.selected_action = \
68              np.random.choice(list(self.prob_policy.keys()), p=
    list(self.prob_policy.values()), size=1)[0]
69          return self.selected_action
70
71      def trim_prob_policy(self, prob_policy):
72          if sum(list(prob_policy.values())) > 1:
73              difference = sum(list(prob_policy.values())) - 1
74              highest_prob = max(prob_policy, key=prob_policy.get
    )
75              prob_policy[highest_prob] -= difference
76              self.out(" Probability needed to be adjusted by " +
     str(difference) + " for action " + str(highest_prob))
77
78          return prob_policy
79
80      def out(self, text):
81          if self.verbose == 1:
82              print("    <" + self.name + ">: " + text)
83
84      def setVerbose(self, value):
85          self.verbose = value
86
87      def print_state(self):
88          self.out("Usage count for this automata: " + str(self.
    usage_count))
89          self.out("Probability policy:")
90          self.out(str(self.prob_policy))
91          self.out("Sum of probablilities: " + str(sum(list(self.
    prob_policy.values()))))
92          self.out("Accuracy so far: " + str(self.
    calculate_accuracy()))
93          self.out("history: " + str(np.array(self.history)))
94
95      def calculate_accuracy(self):
96          # this function uses a sliding window of the last X
    outcomes to determine it's accuracy
97          return sum(self.history) / self.history_length
```

9.2 plot_results_phase1.py

```
import matplotlib.pyplot as plt

# plot the learning probablities of automaton during x updates
def plot_automaton_behavior(f1, f2, f3, f4, f5):
    figure, axs = plt.subplots(3, 2)
    axs[0, 0].plot(f1)
    axs[0, 0].set_title('F1')
    axs[0, 1].plot(f2, 'tab:orange')
    axs[0, 1].set_title('F2')
    axs[1, 0].plot(f3, 'tab:green')
    axs[1, 0].set_title('F3')
    axs[1, 1].plot(f4, 'tab:red')
    axs[1, 1].set_title('F4')
    axs[2, 0].plot(f5, 'tab:gray')
    axs[2, 0].set_title('F5')
    axs[2, 1].axis('off')
    figure.tight_layout()
    figure.suptitle('Single Automaton with a=0.3, b=0.05')
    plt.rcParams['font.size'] = '5'
    for ax in axs.flat:
        ax.set(xlabel='Updates',ylabel='Probability of Action')
    plt.show()
```

9.3 run_experiment_phase1.py

```
import re
import numpy as np
from LA_v1 import Automaton
from plot_results_phase1 import plot_automaton_behavior

# lists save the history of the actions' learning probabilities
# of the automaton during x updates
f1 = []
f2 = []
f3 = []
f4 = []
f5 = []

def generate_rate():
    # this function generates a random value between 0 and 50
    after a normal distribution
    # average will be 25
    return np.random.normal(loc=25)

def evaluate_action(action, current_rate, ):
    # lets figure out what flavor was chosen and calculate the
    "capacity" of this flavor
```

```
    capacity = int(re.search(r'\d+', action).group()) * 10
    if capacity > current_rate:
        return 1
    else:
        return 0

# initiated object from Automaton class
new_service = Automaton(0.3, 0.05, "First")
action = new_service.select_action()

# we have initiated the probabilities ( but no action is taken
    yet )
new_service.print_state()
new_service.update_policy("None")
new_service.setVerbose(0)

for iteration in range(1, 200):
    new_service.setVerbose(1)

    # the state of the environment
    current_rate = generate_rate()

    # the service makes a move:
    action = new_service.select_action()

    # evaluate if we should reward or punish the action
    outcome = evaluate_action(action, current_rate)

    # update the actions' learning probabilities
    new_service.update_policy(outcome)
    # save the history of the actions' values of probabilities
    during the learning process(no. updates)
    x = list(new_service.prob_policy.values())
    f1.append(round(x[0], 4))
    f2.append(round(x[1], 4))
    f3.append(round(x[2], 4))
    f4.append(round(x[3], 4))
    f5.append(round(x[4], 4))

# draw the results of the iteration
plot_automaton_behavior(f1, f2, f3, f4, f5)
```

9.4 aSpace3D

```
import random
import numpy as np
from numpy_ringbuffer import RingBuffer
```

```
4  from LA_v2 import *
5
6  class aSpace3D:
7
8      def __init__(self, d1length, d2length, d3length, name="", a
       =0.1, b=0.05, verbose=1, history=10):
9
10         self.verbose = verbose
11
12         self.name = name
13
14         self.a = a
15
16         self.b = b
17
18         self.d1length = d1length
19         self.d2length = d2length
20         self.d3length = d3length
21
22         self.history_length = history
23
24         self.aspace = [[[0 for k in range(1, d3length + 2)] for
       j in range(1, d2length + 2)] for i in
25                        range(1, d1length + 2)]
26         #self.aspace = [[[0 for k in range(d3length)] for j in
       range(d2length+1)] for i in
27                        #range(d1length+1)]
28
29         self.usage_count = 0
30
31         self.history = RingBuffer(capacity=history)
32
33         self.automata_count = 0
34
35     def out(self, text):
36         if self.verbose == 1:
37             print("[" + self.name + "]: " + text)
38
39     def setVerbose(self, value):
40         self.verbose = value
41
42     def select_action(self, p1, p2, p3):
43         # for example 1,2,4
44
45         # find the right automata ( and create it if it doesnt
       exist )
46
47         if self.aspace[p1][p2][p3] == 0:
```

```
48              # automata does not exist. We need to create it and
     update the policy once in order to initialize it
49              # the name will mimic it's position in the space,
    so 1,2,4 will be named 1-2-4
50              self.aspace[p1][p2][p3] = Automaton(self.a, self.b,
     str(p1) + "-" + str(p2) + "-" + str(p3))
51              self.aspace[p1][p2][p3].update_policy("None")
52              self.aspace[p1][p2][p3].setVerbose(0)
53              self.automata_count += 1
54
55          return self.aspace[p1][p2][p3].select_action()
56
57      def update_policy(self, p1, p2, p3, environment_response):
58          self.aspace[p1][p2][p3].update_policy(
    environment_response)
59          self.usage_count += 1
60          self.history.append(environment_response)
61
62      def print_state(self):
63          #         print("This is the state of the automata")
64          #         print(version)
65          self.out("Usage count for this A-space machine: " + str
    (self.usage_count))
66
67          self.out("Accuracy so far for machine: " + str(self.
    calculate_accuracy()))
68          self.out("Content of ringbuffer (history): " + str(np.
    array(self.history)))
69          self.out("Automata space output: ")
70
71          for p1 in range(1, self.d1length + 1):
72              for p2 in range(1, self.d2length + 1):
73                  for p3 in range(1, self.d3length + 1):
74                      if self.aspace[p1][p2][p3] != 0:
75                          self.aspace[p1][p2][p3].setVerbose(1)
76                          self.aspace[p1][p2][p3].print_state()
77
78      def calculate_accuracy(self):
79          # this function uses a sliding window of the last X
    outcomes to determine it's accuracy
80          return sum(np.array(self.history)) / self.
    history_length
81
82      def print_automata_statistics(self):
83          self.out("Automata count: " + str(self.automata_count))
84          self.automata_saturation = self.automata_count / (self.
    d1length * self.d2length * self.d3length)
85          self.out("saturation of possible number of automata: "
```

```
    + str(self.automata_saturation))
86          automata_usage_count = []
87          automata_accuracy_count = []
88
89          for p1 in range(1, self.d1length + 1):
90              for p2 in range(1, self.d2length + 1):
91                  for p3 in range(1, self.d3length + 1):
92                      if self.aspace[p1][p2][p3] != 0:
93                          automata_usage_count.append(self.aspace
    [p1][p2][p3].usage_count)
94                          automata_accuracy_count.append(self.
    aspace[p1][p2][p3].calculate_accuracy())
95
96          self.out("************ Count statistics **********")
97
98          self.out("Median usage: " + str(np.median(
    automata_usage_count)))
99          self.out("Min usage: " + str(np.min(
    automata_usage_count)))
100         self.out("Max usage: " + str(np.max(
    automata_usage_count)))
101         self.out("Mean usage: " + str(np.mean(
    automata_usage_count)))
102         self.out("85% percentile usage: " + str(np.percentile(
    automata_usage_count, 85)))
103
104         self.out("************ Accuracy statistics **********")
105         self.out("Mean usage: " + str(np.mean(
    automata_accuracy_count)))
106
107         self.out("Min usage: " + str(np.min(
    automata_accuracy_count)))
108         self.out("15% percentile accuracy: " + str(np.
    percentile(automata_accuracy_count, 15)))
109         self.out("30% percentile accuracy: " + str(np.
    percentile(automata_accuracy_count, 30)))
110         self.out("40% percentile accuracy: " + str(np.
    percentile(automata_accuracy_count, 40)))
111         self.out("Median accuracy: " + str(np.median(
    automata_accuracy_count)))
112         self.out("70% percentile accuracy: " + str(np.
    percentile(automata_accuracy_count, 70)))
113         self.out("85% percentile accuracy: " + str(np.
    percentile(automata_accuracy_count, 85)))
114         self.out("Max usage: " + str(np.max(
    automata_accuracy_count)))
115         return automata_usage_count
```

9.5 aSpace-3D-simulation

```
from aSpace3D import *
import re

def evaluate_action(action, current_rate):
    # lets figure out what flavor was chosen and calculate the
    "capacity" of this flavor
    capacity = int(re.search(r'\d+', action).group())
    if capacity >= current_rate:
        return 1
    else:
        return 0

# translate the traffic into set of categories to be provided
    into machine as inputs
def translate_rate(ratestring):
    rate = int(ratestring)
    if rate <= 10000:
        return 1
    elif rate > 10000 and rate <= 20000:
        return 2
    elif rate > 20000 and rate <= 30000:
        return 3
    elif rate > 30000 and rate <= 40000:
        return 4
    elif rate > 40000 and rate <= 50000:
        return 5
#provide the machine with inputs
machine = aSpace3D(name="3Dtest", d1length=288, d2length=5,
    d3length=5)

# Using readlines()
file1 = open('tf2_50k.dat', 'r')
Lines = file1.readlines()

count = 0
last_rate = 0
current_rate = 0
# Strips the newline character
for line in Lines:

    count += 1
    if count > 288:
        count = 1
    if last_rate == 0:
        last_rate = translate_rate(line.strip())
        continue
```

```
44
45      current_rate = translate_rate(line.strip())
46      action = machine.select_action(count, current_rate,
        last_rate)
47      outcome = evaluate_action(action, current_rate)
48      machine.update_policy(count, current_rate, last_rate,
        outcome)
49
50  #print the outputs
51  machine.print_state()
52  machine.print_automata_statistics()
```

9.6 plot_traffic_data

```
1   import matplotlib.pyplot as plt
2   import numpy as np
3
4   def load_data_set(file):
5       data_set = np.loadtxt(file)
6       return data_set
7
8   def chunks(lst, n):
9       for i in range(0, len(lst), n):
10          yield lst[i:i + n]
11
12  def reshape_data(data):
13      for i in range(len(data)):
14          if i == len(data) - 1:
15              list_len = 288 - len(data[i])
16              convert_list = data[i].tolist()
17              for j in range(list_len):
18                  convert_list.append(0)
19              data[i] = np.array(convert_list)
20      return data
21
22  def get_statistics(data):
23      mean = []
24      std = []
25      median =[]
26      data = np.array(data)
27      for col in data.T:
28          mean.append(np.mean(col))
29          std.append(np.std(col))
30          median.append(np.median(col))
31      return mean, std, median
32
33  def plot_data(data_set, mean, std, median):
```

```
    x = np.arange(1, 289, 1)
    colors = []
    for i in range(288):
        if i == 0:
            colors.append('b')
        elif (i % 2) == 0:
            colors.append('b')
        else:
            colors.append('r')

    for i in range(len(data_set)):
        y = data_set[i]
        plt.plot(x, y,'--',linewidth=0.7)

    plt.errorbar(x, mean, yerr=std,fmt="k--", errorevery=72,
    label="mean")

    plt.plot(median,'r-', linewidth=3, markersize=3,label="
    median")

    plt.title('Traffic Intensity')
    plt.xlabel('24H')
    plt.ylabel('Players')
    plt.legend(loc="upper right")
    plt.show()

data_set = load_data_set('tf2_50k.dat')
file_data = list(chunks(data_set, 288))
data_to_plot = reshape_data(file_data)
mean, std, median = get_statistics(file_data)
plot_data(data_to_plot, mean, std, median)
```

Bibliography

Al-Dhuraibi, Y., Paraiso, F., Djarallah, N. & Merle, P. (2017). Elasticity in cloud computing: State of the art and research challenges. *IEEE Transactions on Services Computing*, *11*(2), 430–447.

Begnum, K. M. (2006). Managing large networks of virtual machines. *LISA*, *6*, 205–214.

Bharanidharan, G. & Jayalakshmi, S. (2021). Elastic resource allocation, provisioning and models classification on cloud computing a literature review. *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, *1*, 1909–1915.

Bhat, A. (2019). Exploratory research: Definition, methods, types and examples. *Questionpro. com*.

Burgess, M. et al. (1998). Computer immunology. *LISA*, *98*, 283–298.

Cámara, J., de Lemos, R., Vieira, M., Almeida, R. & Ventura, R. (2013). Architecture-based resilience evaluation for self-adaptive systems. *Computing*, *95*(8), 689–722.

Carzaniga, A., Gorla, A. & Pezzè, M. (2008). Self-healing by means of automatic workarounds. *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, 17–24.

De Wolf, T. & Holvoet, T. (2006). Evaluation and comparison of decentralised autonomic computing systems. *CW Reports*, 10–10.

Dewangan, B. K., Agarwal, A., Venkatadri, M. & Pasricha, A. (2019). Design of self-management aware autonomic resource scheduling scheme in cloud. *International Journal of Computer Information Systems and Industrial Management Applications*, *11*, 170–177.

Gill, S. S., Chana, I., Singh, M. & Buyya, R. (2019). Radar: Self-configuring and self-healing in resource management for enhancing quality of cloud services. *Concurrency and Computation: Practice and Experience*, *31*(1), e4834.

Horn, P. (2001). Autonomic computing: Ibm's perspective on the state of information technology.

Houben, G.-J., Fiala, Z., Van Der Sluijs, K. & Hinz, M. (2005). Building self-managing web information systems from generic components. *CAiSE Workshops (2)*, 53–67.

Johnoommen, B. (1986). Absorbing and ergodic discretized two-action learning automata. *IEEE transactions on systems, man, and cybernetics*, *16*(2), 282–293.

Kaddoum, E., Raibulet, C., Georgé, J.-P., Picard, G. & Gleizes, M.-P. (2010). Criteria for the evaluation of self-* systems. *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, 29–38.

Kapoor, V. (2005). Services and autonomic computing: A practical approach for designing manageability. *2005 IEEE International Conference on Services Computing (SCC'05) Vol-1*, *2*, 41–48.

Karakostas, B. (2014). Towards autonomic cloud configuration and deployment environments. *2014 International Conference on Cloud and Autonomic Computing*, 93–96.

Lakshmivarahan, S. (2012). *Learning algorithms theory and applications: Theory and applications*. Springer Science & Business Media.

Li, W., Svärd, P., Tordsson, J. & Elmroth, E. (2012). A general approach to service deployment in cloud environments. *2012 Second International Conference on Cloud and Green Computing*, 17–24.

Marshall, D., Beaver, S. S. & McCarty, J. W. (2008). *Vmware esx essentials in the virtual data center*. Auerbach Publications.

Mateen, M., Hayat, S., Tehreem, T., Akbar, M. A. et al. (2020). A self-adaptive resource provisioning approach using fuzzy logic for cloud-based applications. *International Journal of Computing and Digital Systems*, *9*(03).

Mateescu, G., Gentzsch, W. & Ribbens, C. J. (2011). Hybrid computing—where hpc meets grid and cloud computing. *Future Generation Computer Systems*, *27*(5), 440–453.

Maurer, M., Brandic, I. & Sakellariou, R. (2012). Self-adaptive and resource-efficient sla enactment for cloud computing infrastructures. *2012 IEEE Fifth International Conference on Cloud Computing*, 368–375.

Mell, P. & Grance, T. (2011). The nist definition of cloud computing.

Misra, S., Krishna, P. V., Kalaiselvan, K., Saritha, V. & Obaidat, M. S. (2014). Learning automata-based qos framework for cloud iaas. *IEEE Transactions on Network and Service Management*, *11*(1), 15–24.

Moreno-Vozmediano, R., Montero, R. S., Huedo, E. & Llorente, I. M. (2019). Efficient resource provisioning for elastic cloud services based on machine learning techniques. *Journal of Cloud Computing*, *8*(1), 1–18.

Narendra, K. S. & Thathachar, M. A. (1974). Learning automata-a survey. *IEEE Transactions on systems, man, and cybernetics*, (4), 323–334.

Narendra, K. S. & Thathachar, M. A. (2012). *Learning automata: An introduction*. Courier corporation.

Nowé, A., Verbeeck, K. & Peeters, M. (2005). Learning automata as a basis for multi agent reinforcement learning. *International Workshop on Learning and Adaption in Multi-Agent Systems*, 71–85.

Pastrana, J. L., Pimentel, E. & Katrib, M. (2008). Composition of self-adapting components for customizable systems. *The Computer Journal*, *51*(4), 481–496.

Qavami, H. R., Jamali, S., Akbari, M. K. & Javadi, B. (2017). A learning automata based dynamic resource provisioning in cloud computing environments. *2017 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, 502–509.

Ranjbari, M. & Torkestani, J. A. (2018). A learning automata-based algorithm for energy and sla efficient consolidation of virtual machines in cloud data centers. *Journal of Parallel and Distributed Computing*, *113*, 55–62.

Sterritt, R., Parashar, M., Tianfield, H. & Unland, R. (2005). A concise introduction to autonomic computing. *Advanced engineering informatics*, *19*(3), 181–187.

Tadakamalla, U. & Menasce, D. A. (2021). Autonomic resource management for fog computing. *IEEE Transactions on Cloud Computing*.

Tesauro, G., Chess, D. M., Walsh, W. E., Das, R., Segal, A., Whalley, I., Kephart, J. O. & White, S. R. (2004). A multi-agent systems approach to autonomic computing. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, 464–471.

Thathachar, M. (1990). Stochastic automata and learning systems. *Sadhana*, *15*(4-5), 263–281.

Tianfield, H. (2003). Multi-agent autonomic architecture and its application in e-medicine. *IEEE/WIC International Conference on Intelligent Agent Technology, 2003. IAT 2003.*, 601–604.

Toka, L., Dobreff, G., Fodor, B. & Sonkoly, B. (2020). Adaptive ai-based auto-scaling for kubernetes. *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, 599–608.

Tomar, R., Khanna, A., Bansal, A. & Fore, V. (2018). An architectural view towards autonomic cloud computing. *Data engineering and intelligent computing* (pp. 573–582). Springer.

Tyvand, J.-E. (2011). *On the predictability of server resources in online games, an investigative approach* (Master's thesis).

Velusamy, G. (2018). *Energy-delay aware web request routing using learning automata* (Doctoral dissertation). University of Houston.

Voas, J., Bojanova, I. & Zhang, J. (2013). Cloud computing. *IT Professional*, *15*(2), 0012–14.

White, S. R., Hanson, J. E., Whalley, I., Chess, D. M. & Kephart, J. O. (2004). An architectural approach to autonomic computing. *International Conference on Autonomic Computing, 2004. Proceedings.*, 2–9.

9.7 LAs.py

```
import random
import numpy as np
from numpy_ringbuffer import RingBuffer

#


# This is a LA of the running service in the cloud
class Automaton:

    def __init__(self, a, b, name,verbose=1,history=10):
        # The reward parameter can be a = 0.1
        self.a = a
        # The penalized parameter can be b = 0.05
        self.b = b
        # the name of this particular automata instance
        self.name = name

        self.verbose = verbose

        self.history_length = history

        self.version = "v1"
        # A list of the available actions to choose between
        self.available_actions = ["f1", "f2", "f3", "f4", "f5"
    ]
```

```
26          # self.available_actions = ["f1", "f2", "f3", "f4", "f5
    " ]
27          # A variable to store the selected action after each
    response from the environment
28          self.selected_action = None
29          # Dict for the policy mapping states to probabilities
30          self.prob_policy = {'f1': 0, 'f2': 0, 'f3': 0, 'f4': 0,
     'f5': 0}
31
32          self.usage_count = 0
33
34          self.history = RingBuffer(capacity=history)
35
36      # function to get the environmental_response and update
    rule for changing the policy mapping states to
    probabilities
37      # or preferences for actions.
38      # if the selected_action is favorable then the
    environmental_response will be 1
39      # or unfavorable then the environmental_response will be 0
40      def update_policy(self, environmental_response):
41          # if it is first iteration then the selection will be
    random and all actions will have equal possibilities
42          if environmental_response is "None":
43              number_of_actions = len(self.available_actions)
44              initial_prob = 1 / number_of_actions
45              for policy_key in self.prob_policy.keys():
46                  self.prob_policy[policy_key] = initial_prob
47              return
48          for key, value in self.prob_policy.items():
49              if key == self.selected_action:
50                  # apply p_i formula on the chosen action
51                  self.prob_policy[key] = value + a *
    environmental_response * ( 1 - value) - b * ( 1 -
    environmental_response )*value
52                  #self.prob_policy[key] = value + self.a * (1 -
    value)
53              else:
54                  # apply p_j formula on all other actions
55                  self.prob_policy[key] = value - a *
    environmental_response * value + b * ( 1 -
    environmental_response ) * ( 1/(len(self.available_actions)
     - 1) - value )
56  #                self.prob_policy[key] = (1 - self.a) * value
57
58
59          self.prob_policy = self.trim_prob_policy(self.
    prob_policy)
```

```
60          self.history.append(environmental_response)
61          self.usage_count += 1
62
63      # function to assign the action with highest probability to
         be the selected_action at time(t+1)
64      def select_action(self):
65          # check if all actions have same probabilities then
    choose one action randomly
66          prob_equal = len(list(set(list(self.prob_policy.values
    ()))))) == 1
67          if prob_equal:
68              random_action_index = random.randint(1, len(self.
    available_actions))
69              self.selected_action = self.available_actions[
    random_action_index - 1]
70          else:
71              # change this one
72              self.selected_action = np.random.choice(list(self.
    prob_policy.keys()),p=list(self.prob_policy.values()),size
    =1)[0]
73  #              self.selected_action = max(self.prob_policy, key=
    self.prob_policy.get)
74
75          self.out("selecting action: " + self.selected_action)
76          return self.selected_action
77
78
79      def trim_prob_policy(self,prob_policy):
80          if sum(list(prob_policy.values())) > 1:
81              difference = sum(list(prob_policy.values())) - 1
82              highest_prob = max(prob_policy, key=prob_policy.get
    )
83              prob_policy[highest_prob] -= difference
84              self.out(" Probability needed to be adjusted by " +
     str(difference) + " for action " + str(highest_prob))
85
86          return prob_policy
87
88      def out(self,text):
89          if self.verbose == 1:
90              print("   <" + self.name + ">: " + text)
91
92      def setVerbose(self,value):
93          self.verbose = value
94
95      def print_state(self):
96  #         print("This is the state of the automata")
97  #         print(version)
```

```
98          self.out("Usage count for this automata: " + str(self.
    usage_count))
99          self.out("Probability policy:")
100         self.out(str(self.prob_policy))
101         self.out("Sum of probablilities: " + str(sum(list(self.
    prob_policy.values()))))
102         self.out("Accuracy so far: " + str(self.
    calculate_accuracy()))
103         self.out("Content of ringbuffer (history): " + str(np.
    array(self.history)))
104
105     def calculate_accuracy(self):
106         # this function uses a sliding window of the last X
    outcomes to determine it's accuracy
107         return sum(np.array(self.history)) / self.
    history_length
```

9.8 environment_response

```
1     import re
2 class flavorEnvironment:
3     def __init__(self,name,f_lower = 0.1, f_higher = 0.2):
4        self.name = name
5        self.f_higher = f_higher
6        self.f_lower = f_lower
7     def evaluate_action(self,action,current_rate):
8         # lets figure out what flavor was chosen and calculate
    the "capacity" of this flavor
9         capacity = int(re.search(r'\d+', action).group())
10
11        if capacity > current_rate:
12            # Too high, we are wasting resources. apply
    f_higher
13            response = self.f_higher / ( capacity -
    current_rate )
14        elif capacity < current_rate:
15            # Too low, we are damaging the service. apply
    f_lower
16            response = self.f_lower / ( current_rate - capacity
     )
17        else:
18            response = 1
19
20        return response
```