ISSN 1688-2806

Universidad de la República
Facultad de Ingeniería

Optimizing Cloud Elasticity: A Deep Reinforcement Learning Approach Enhanced by Transfer Learning

Tesis presentada a la Facultad de Ingeniería de la Universidad de la República por

Santiago Serantes

en cumplimiento parcial de los requerimientos
para la obtención del título de
Magister en Ciencia de Datos y Aprendizaje Automático.

Directores de Tesis

Javier Baliosian ..... Universidad de la República
Matías Richart ..... Universidad de la República

Tribunal

Federico La Rocca ..... Universidad de la República
Jorge Visca ..... Universidad de la República
Matías Di Martino ..... Universidad Católica del Uruguay

Director Académico

Javier Baliosian ..... Universidad de la República

Montevideo
Tuesday 15th July, 2025

Optimizing Cloud Elasticity: A Deep Reinforcement Learning Approach Enhanced by Transfer Learning, Santiago Serantes.

ISSN 1688-2806

This thesis war prepared in LaTeX using the class iietesis (v1.1).
It contains a total of 103 pages.
Compiled Tuesday 15th July, 2025.

Sean los orientales tan ilustrados como valientes.

José Gervasio Artigas

This page intentionally left blank.

Acknowledgements

Quiero expresar mi mayor agradecimiento a mis tutores, Javier y Matías, quienes me brindaron un enorme apoyo a lo largo de estos más de dos años de trabajo en esta tesis. Su guía constante, las reuniones semanales y el compromiso tanto conmigo como con el proyecto fueron fundamentales para superar los desafíos y llevar adelante el desarrollo de la tesis. Gracias por su paciencia, dedicación y por compartir todo su conocimiento conmigo.

También quiero agradecer a mi familia, que estuvo siempre a mi lado durante todo el proceso de la maestría, que inicialmente se suponía que iba a durar dos o tres años, pero terminó extendiéndose a casi cinco. En algunos momentos, parecía que estaban más ansiosos que yo por que esto llegara a su fin, y siempre me dieron el empujón necesario para seguir adelante y lograr cumplir con el objetivo.

This page intentionally left blank.

Abstract

Cloud elasticity enables providers to dynamically scale application resources in response to fluctuating demand. Traditional scaling mechanisms often rely on simple heuristics, which can lead to suboptimal performance and resource utilization.

This work proposes a Deep Reinforcement Learning based controller designed to manage cloud resources more efficiently. Although RL-based controllers have been explored previously, they often suffer from poor initial performance, which limits their practical applicability in real-world scenarios. To address this issue, an investigation into transfer learning is performed, and two distinct transfer learning techniques are explored: Sim-to-Real Transfer and Learning from Demonstrations, which significantly enhance initial controller performance, making RL viable for cloud elasticity management from the outset. Sim-to-Real Transfer utilizes simulation-based training to embed the model with prior knowledge, while Learning from Demonstrations leverages expert behaviors to significantly improve early-stage performance, thereby reducing the time required for effective scaling.

The proposed model was evaluated using CloudSim Plus, a well established cloud simulation tool. The results demonstrate substantial performance improvements over traditional heuristic methods, with both transfer learning techniques notably improving the initial deployment phase of the RL controller. Specifically, these advancements render the use of RL in cloud elasticity scenarios not only viable but also highly advantageous. These findings open avenues for further exploration of RL-based cloud management strategies and demonstrate the potential of transfer learning to make RL models suitable in scenarios where it was previously unfeasible.

This page intentionally left blank.

Contents

This page intentionally left blank.

Chapter 1

Introduction

Cloud computing has gained significant popularity in recent times, allowing clients to focus exclusively on the development of applications without the need of managing the underlying infrastructure. A key functionality enabled by cloud computing is resource scaling, which is the ability to dynamically adjust the number of resources deployed to run an application. This capability is essential for adapting to the varying demands of users and applications efficiently.

This capability can significantly reduce operational costs by efficiently managing allocated resources in response to real-time demand fluctuations, using only the necessary resources to satisfy incoming demand and thereby lowering costs. Additionally, it can lead to an enhanced user experience, decreased energy consumption, and consequently a reduction in environmental impact, among other benefits [30].

However, achieving optimal resource management in cloud environments is an exceedingly complex issue due to the dynamic and unpredictable incoming demand, which is directly influenced by end-user interaction. A major challenge is that this demand can vary significantly, exhibiting both predictable patterns and elements of high randomness.

Traditional resource management approaches in cloud environments are usually based on simple heuristics, such as threshold-based policies, and frequently prove inadequate for efficiently handling unpredictable and dynamic workloads, leading to sub-optimal resource provisioning. These conventional approaches often result in either under-provisioning, which compromises system performance and user satisfaction, or over-provisioning, which incurs unnecessary costs.

It is critical for the resource manager to guarantee the fulfillment of Service Level Agreements (SLAs), which are contracts between providers and clients that define acceptable performance metrics, such as maximum response times. Failure to meet these metrics can result in penalties or a degraded user experience. The impact of SLAs is especially significant when it comes to response times. When the system does not have enough resources to meet demand, it experiences increased response times, and unprocessed requests start to accumulate. This delay can directly lead to SLA violations, leading to penalties. Conversely, over-allocating resources to prevent SLA violations results in higher costs, undermining the cost-efficiency of the system.

Resource scaling can be classified into two categories: horizontal scaling or vertical scaling. Horizontal scaling involves scaling the resources by increasing the number of instances allocated to the task (servers or VMs), distributing the load across a larger number of instances, therefore increasing the total system's capabilities. On the other hand, vertical scaling augments the resources allocated to a specific instance in order to augment its capacity. There are a number of resources that can be scaled. Most often it is the number of

CPU cores, but other possibilities include the amount of memory, data bandwidth, storage, and more.

However, cloud elasticity also has its drawbacks, including the potential for increased latency during scaling events, unpredictable cost fluctuations due to dynamic resource allocation, and the complexity of managing autoscaling policies, which can result in over- or under-provisioning if not carefully tuned.

1.1 Objectives

The objectives of this thesis are twofold. The first objective is to engineer a generic deep reinforcement learning-based controller capable of managing cloud resources dynamically by learning optimal provisioning strategies in response to demand fluctuations, which aims to improve upon the performance achievable with existing solutions. In addition to serving as a resource management solution, the controller provides a framework for evaluating the impact of transfer learning techniques on its performance.

The second objective is to study the use of transfer learning techniques in the context of reinforcement learning for cloud resource management. This analysis explores their potential to enhance the initial performance of the RL controller and mitigate the traditionally extensive training process required to learn effective strategies. Based on a review of transfer learning approaches, two techniques—Simulation to Reality (Sim-to-Real) [54] and Learning from Demonstrations [56]—were selected for further investigation.

1.2 Solution

The proposed solution is based on a Deep Reinforcement Learning (DRL) controller for managing cloud resources, a decision motivated by several key reasons. Its model-free approach eliminates the need to learn or model the complex, dynamic, and non-deterministic environment, as it can learn the transition dynamics and identify actions that optimize rewards. It is capable of handling a large state space, which is crucial for managing the extensive size of continuous input variables. DRL optimizes long-term decision-making, which is essential for balancing immediate and long-term effects. Moreover, it supports continuous learning, constantly adapting to evolving environmental dynamics, unlike static train-then-deploy models. Additionally, DRL can easily be adapted to different elasticity policies, allowing it to perform effectively across diverse operational scenarios.

Nevertheless, designing an RL controller to manage resources poses a number of complex challenges. One such challenge is the issue of delayed rewards, where the effects of actions taken are not immediately apparent. For instance, allocating additional CPU to reduce high response times will only show marginal improvements initially, making it difficult for the controller to associate actions with their long-term impacts on system performance. Additionally, the controller must adapt to dynamic workloads in cloud environments that can change rapidly, which is essential to maintain a positive user experience. Furthermore, balancing performance requirements with cost considerations is a crucial challenge, as the controller must optimize resource allocation for both performance and cost-effectiveness. Finally, Reinforcement Learning-based models require a significant period of time interacting with the environment to properly learn a highly performing policy. Although there are several techniques that promise to minimize the learning period, none can completely eliminate it, rendering the deployment of an RL controller in real environments unviable, given that poor performance during training is intolerable in production settings.

It is this last challenge that leads to the second objective of this thesis: investigating the use of transfer learning techniques to minimize or eliminate the RL controller's poor performance during the training phase.

Transfer learning facilitates knowledge transfer from a source domain to a target domain. By leveraging data and experience from similar or pre-existing systems, this approach can be used to pre-train an RL controller. This preliminary training allows the controller to achieve a satisfactory performance level more rapidly, potentially even from the outset.

The field of transfer learning is vast, offering a variety of techniques adaptable to different use cases. Two techniques are particularly suitable for the case study. The first technique, Sim-to-Real, involves pre-training the RL controller in a simulator. This method allows the controller to learn a policy based on the simulator's behavior, which should be similar to the real-world conditions, providing prior knowledge of system behavior. When transferred to the real environment, the controller already possesses a solid knowledge base. The second technique, learning from demonstrations, involves using examples of actions from an expert or a pre-existing controller, such as a threshold-based controller, to pre-train the model to imitate the demonstrations provided. Once pre-trained with these examples, the RL controller is deployed in the real environment, already embedded with some understanding of optimal actions.

This research specifically addresses the optimization of cloud resource management for a generic API serving as a video server, characterized by highly variable demand driven by user interactions.

To validate the efficacy of the proposed resource manager and the improvements in comparison to conventional techniques, experiments were conducted within a simulated environment, mirroring real-world cloud computing scenarios.

This investigation aims to demonstrate that a machine learning-based approach can significantly refine resource management efficiency in cloud computing, thereby minimizing operational costs, increasing user satisfaction, and potentially reducing the environmental footprint of data centers. Additionally, it seeks to demonstrate that applying transfer learning can significantly shorten the required training time, enabling new use cases that cannot tolerate sub-optimal initial performance.

1.3 Main Challenges

There are several challenges that arise when designing a controller for resource management.

1.3.1 State and Action Spaces

Defining appropriate state and action spaces for the Reinforcement Learning (RL) algorithm is crucial. The state should capture relevant information about the system, and the action space should be capable of adjusting resources appropriately. Choosing these spaces effectively impacts the learning process significantly.

1.3.2 Delayed Rewards

Actions take time to impact the system; that is, the state of the system begins to change only after an action is taken, and the full extent of the action's effects may only be fully appreciated after a prolonged period of time. For instance, if the system is in a state where more resources are needed (due to high response times) and additional CPU is allocated, the response time will only gradually decrease, remaining high for a period of time. The challenge lies in enabling the controller to associate actions with their long-term effects on system performance, a difficulty that is further exacerbated by the continuing nature of the task, where there is no clearly defined endpoint or *episode* after which the quality of all actions can be evaluated.

1.3.3 Scaling Delay

Another related difficulty arises once an action is decided upon; it may entail a lengthy period before a new resource can be effectively added, whether it involves setting up a new virtual machine and reconfiguring the load balancer, or adding a new core to an existing virtual machine.

1.3.4 Dynamic Workloads

Workloads in cloud environments can change rapidly, and the controller must adapt quickly to avoid negatively affecting the user experience of those utilizing the service.

1.3.5 Non Determinism

The complexity of managing dynamic workloads is further emphasized by their inherent randomness. For a given state and action, the reward obtained may vary depending on how the demand evolves. Consequently, the optimal action might differ based on the future of the workload, which is unknown to the controller. This uncertainty complicates the controller's learning of an optimal and stable policy.

1.3.6 Multi-Objective Problem

Resource management in cloud elasticity is a multi-objective problem. Balancing performance requirements with cost considerations presents a crucial challenge. The controller must optimize resource allocation not only for performance but also for cost-effectiveness.

1.3.7 Data Efficiency and Training Time

Training a model often requires a significant amount of data. In cloud environments, collecting and labeling data for training can be challenging. Techniques for efficient data learning must be explored. Deploying the controller without prior training can result in poor initial performance, to the extent that deploying it in reality is not viable.

Chapter 2

Background

This chapter provides an overview of key concepts and tools relevant to the development of a reinforcement learning controller for managing cloud elasticity, and the use of transfer learning to improve its training speed. It starts with an introduction to cloud elasticity, covering its advantages, challenges, and exploring different resolution strategies. The next section introduces the fundamentals of reinforcement learning, followed by a detailed investigation of transfer learning, covering its foundational concepts and classifications or categorizations. Finally, a review of cloud simulation tools is presented, with a focus on available platforms, their capabilities, and the selection of an appropriate simulator for this work.

2.1 Cloud Elasticity

2.1.1 Introduction

Resource elasticity is one of the most notable features of cloud computing [32]. Elasticity refers to the ability to dynamically add or release computing resources in a system. This capability allows application providers to automatically scale allocated resources without human intervention in response to dynamic workloads. This is done with the goal of minimizing resource costs while maintaining or improving compliance with Quality of Service (QoS) requirements. While it is most common for elastic systems to automatically adjust resources, a manual alternative does exist, where administrators can adjust resources manually in real time as the needs change.

There are two alternatives to resource scaling: horizontal scaling and vertical scaling [3].

Horizontal scaling involves adding or removing instances (servers or VMs) to handle changes in demand. This is typically done by distributing workloads across multiple instances to improve system capacity and ensure availability during peak demand periods. Horizontal scaling allows systems to dynamically increase their capacity by simply adding more instances, making it ideal for handling large, distributed applications.

Vertical scaling, on the other hand, involves increasing or decreasing the capacity of an existing instance. This typically includes increasing the amount of resources such as CPU, memory, or storage allocated to a single machine. Unlike horizontal scaling, where more instances are added, vertical scaling involves making a single instance more powerful. While vertical scaling can provide scalability, it has limitations due to the physical constraints of the hardware on which the instance is running. For example, increasing the memory or CPU of a server will eventually hit a limit, at which point horizontal scaling may be required to continue supporting growth [35].

Elasticity is closely related to, but distinct from, scalability. Scalability is the system's ability to handle increasing workloads by expanding its resource capacity, whether through vertical or horizontal scaling. The critical difference between the two is that scalability is a static property, concerned with how a system can be designed to grow to meet future demand, whereas elasticity is a dynamic property, concerned with how a system automatically adjusts to demand changes in real-time. Scalability is a system's potential for growth, while elasticity is how efficiently and quickly a system can adapt to fluctuations in demand.

2.1.2 Principles of Cloud Elasticity

In cloud systems, elasticity is performed through automated components called *scaling controllers*, which dynamically adjust resource allocations in response to real-time fluctuations in user demand. These controllers continuously monitor system metrics and make decisions aimed at balancing performance with operational costs.

An application providing video streaming via an API can be used as an example. Users access the application at various rates during the day, sometimes creating peaks in demand. Each user request consumes a portion of the application's processing resources—typically computational resources like CPU cycles, memory, and bandwidth. As demand increases, the amount of computational effort required by the application also grows. Eventually, the application's allocated resources will reach their maximum capacity. At this point, additional incoming requests cannot be adequately processed, resulting in increased response times, accumulation of queued requests, and ultimately, violations of performance guarantees (Service Level Agreements). To avoid this, the controller must take a *scaling action*, adding more resources to increase the application's processing capability. This action is known as scaling out.

Conversely, when demand decreases, the application might become over-provisioned, meaning it has more resources than necessary. In this case, continuing to maintain excess resources leads to unnecessary costs. To prevent this waste, the controller must again take a scaling action, this time reducing resources, an action known as scaling in.

The component responsible for managing these scaling actions is the reinforcement learning controller, which acts as a decision-making agent interacting with the cloud environment. The RL controller follows a specific workflow composed of three fundamental steps: monitoring, decision-making, and acting.

In the first step, the RL controller monitors the current state of the system. This state comprises various metrics that characterize system performance and workload, including but not limited to:

- Resource utilization (e.g., CPU, memory, or network usage percentage).
- Average response time for user requests.
- Number of requests currently queued, awaiting processing.
- Throughput, defined as the number of requests processed per unit of time.

This monitoring is not continuous but occurs at discrete, periodic intervals known as the scaling interval. The choice of this interval directly influences the responsiveness of the system. Short intervals allow more reactive adjustments but increase computational overhead, while longer intervals reduce overhead at the risk of slower adaptation to workload changes.

In the second step, the RL controller takes the collected metrics (the current state) and decides on an appropriate scaling action. To make this decision, the RL controller relies on a policy learned through interactions with the environment. This policy dictates whether

the application should scale out (add resources), scale in (remove resources), or maintain the current resource allocation (no action). For example:

- If resource utilization is persistently high and the number of queued requests is increasing, the controller recognizes a need to scale out by adding more instances to distribute the load and maintain performance standards.
- Conversely, if resource utilization is low and the system has spare capacity, the RL controller will likely decide to scale in, removing unused instances to reduce costs.

In the third step, after deciding the appropriate scaling action, the RL controller executes the action by interacting with the cloud platform. It issues commands to provision or de-provision resources. For instance, scaling out might involve launching new virtual machines (VMs) or containers, while scaling in involves shutting down or releasing existing ones.

An important aspect of the RL controller's decision-making is its attempt to balance two conflicting objectives:

- Minimizing costs, by reducing the number of allocated resources.
- Maximizing user experience, by ensuring low response times and high performance levels.

These objectives inherently pull the controller in opposite directions. Scaling out quickly can improve user experience significantly but increases operational costs. Conversely, scaling in aggressively lowers costs but risks violating performance guarantees if demand suddenly spikes. The RL controller must seek an optimal balance, learning from historical data and past decisions, progressively improving its scaling policy through repeated interactions with the environment.

Figure 2.1 provides a visual representation of the described workflow, illustrating clearly the interaction between the RL controller, the monitored system state, and the scaling actions taken.

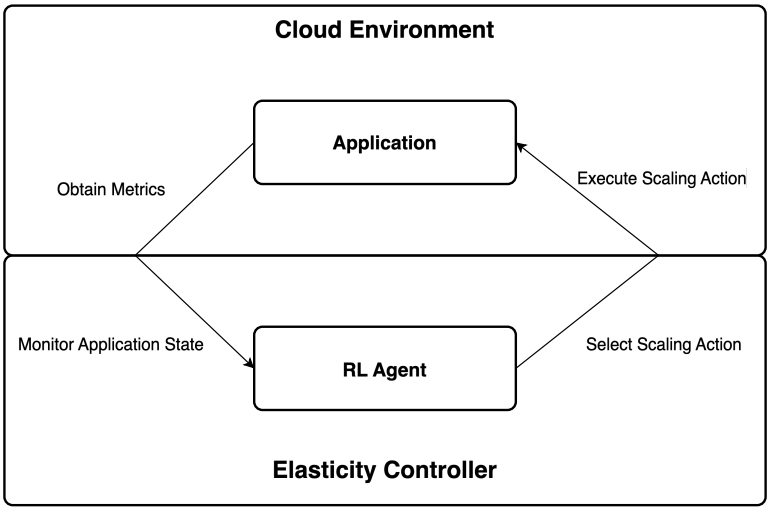

Figure 2.1: Overview of the RL Controller's Workflow in Cloud Elasticity

This continuous cycle of monitoring, decision-making, and action execution ensures that the system maintains optimal resource utilization in the face of dynamic and unpredictable workloads, efficiently managing resources to achieve a balance between performance and cost.

2.1.3 Advantages

The growth of cloud computing has created new possibilities for the management of applications or services. One of these possibilities is cloud elasticity, which focuses on its capacity to radically transform how applications manage and utilize computational resources in cloud environments.

This functionality allows for unprecedented cost optimization by dynamically adjusting resources to exactly match the needs of the workload in real time. By doing so, applications can avoid unnecessary expenses on unused resources while ensuring sufficient capacity to handle demand spikes, resulting in more cost-efficient operations.

Moreover, elasticity plays a crucial role in maintaining a high level of Quality of Service. By allowing resources to scale to meet demand, it ensures that application performance does not degrade during periods of high demand. This is vitally important for critical applications where adherence to Service Level Agreements (SLAs) is fundamental to the provider's success. Additionally, elastic cloud environments strengthens the reliability of services by providing high availability and fault tolerance due to their capacity to replicate VMs and containers, thereby preventing downtime.

The adaptability and scalability provided by elasticity are especially valuable in today's business environment, where agility and adaptability are critical factors for success. Cloud elasticity facilitates this adaptability without requiring significant investments in additional infrastructure or human resources for its management.

Cloud Elasticity additionally provides the ability to quickly meet capacity demands, which allows for a faster rollout of new services and applications, keeping pace with market dynamics.

Finally, the inherent automation in elasticity reduces the need for manual intervention and constant monitoring by IT staff. This allows teams to focus on more strategic and less operational tasks, enhancing overall efficiency and enabling more agile IT infrastructure management. This aspect of elasticity contributes to greater innovation by freeing up valuable resources that can be redirected towards the development of new functionalities and services.

2.1.4 Challenges

While Cloud Elasticity can have a number of advantages, it is not without its challenges.

One of the main challenges is to comply with SLAs while managing elastic resources. SLAs specify the expected performance and availability levels, and failing to meet these can result in penalties [23].

Another challenge in cloud elasticity is the delay in resource allocation, which can severely impact the performance of applications, particularly those requiring high responsiveness and real-time data processing. The process of starting up new virtual machines, configuring networks, and deploying applications can introduce unacceptable delays, contradicting the very principle of elasticity.

Additionally, the multi-tenant nature of cloud computing introduces the challenge of performance isolation, where activities from one tenant can negatively affect the performance of others sharing the same physical resources [39]. Ensuring elastic scaling without resource contention is complex and requires advanced resource management and scheduling algorithms to dynamically allocate resources while protecting individual tenants' performance.

Lastly, resource elasticity increases overall complexity, and for applications with predictable, steady demand, the cost and complexity of implementing an elastic solution may outweigh its benefits.

2.1.5 Resolution Strategies

Resource management in the cloud is a challenge of complex nature. Over time, various approaches have emerged to address this issue [41] [55]. The initial variants used simpler techniques, but more recently, there has been a trend towards the use of increasingly sophisticated techniques with the purpose of improving outcomes.

The different methods can be classified into two main categories [32]: reactive methods and predictive or proactive methods. Reactive methods adjust resources in response to observed changes in workload, whereas predictive methods attempt to anticipate such changes to make proactive adjustments. Reactive methods are simpler and more straightforward to implement, but they may not be as efficient in handling unexpected spikes in demand. Solely relying on reactive mechanisms can result in delayed responses to traffic spikes, causing temporary performance degradation. On the other hand, predictive methods can offer better resource management and cost efficiency by anticipating future needs, using historical data to forecast future demands and preparing the system in advance for expected load changes. These controllers are usually based on machine learning models. However, the effectiveness of predictive scaling is highly dependent on the accuracy of the forecasts, which can be compromised by abrupt and atypical workload patterns.

Reactive Methods

Reactive methods adjust computing resources in response to current changes in demand or performance, without attempting to predict future fluctuations.

Threshold-based rules are methods that operate by reacting to changes in specific system metrics, such as CPU utilization, memory, network bandwidth, or latency. These performance metrics are monitored in real-time or at regular intervals to assess the current state of the system. When these metrics cross a predefined threshold, a self-scaling action is triggered, such as adding or removing resources. This method is simple to implement but can lead to delayed or premature scaling decisions due to its reactive nature. One case can be found in [9].

Fuzzy Rules, unlike traditional techniques that operate under binary logic (all or nothing), evaluate input data on a continuous spectrum of truth values, allowing for a more robust and adaptable scaling decision to subtle variations in the system state. The process has three stages: Fuzzification, where input values are taken and converted into fuzzy values, for example, utilization value is translated into high, medium, low values. The second stage is the application of rules consisting of if-else statements, and the last stage is Defuzzification, which involves transforming the outcome of the rule application into a concrete value with the action to be taken. The method is recognized for its flexibility and its ability to handle the uncertainty and imprecision inherent in monitoring and analyzing the performance of web applications. However, its successful implementation depends on the ability to precisely define rules that best reflect the desired needs and behavior. A work based on this technique is [31].

Predictive Methods

Predictive methods aim to anticipate future resource demands by utilizing historical data and analytical models, enabling proactive resource adjustments.

Application Profiling is a technique that involves a thorough analysis of application behavior under different workloads, identifying how variations in demand affect resource usage and performance. Through this analysis, the goal is to establish a predictive model that can foresee resource needs before significant changes in demand occur, thus ensuring optimal performance and efficient resource management in the cloud. [38] bases its work on this method.

There are two variants: offline profiling and online profiling. Offline profiling is conducted before deploying the application in a production environment and involves running the application under various test scenarios to gather data on resource consumption and performance in response to varied workloads. From the collected data, it is possible to deduce the application's behavior in response to demand changes, and generate a basis for resource adjustment in actual deployment. Unlike offline profiling, online profiling occurs in real-time while the application is operational in a production environment. This phase involves continuously monitoring the application's performance and resource usage, dynamically adapting to demand fluctuations.

Control Theory is a technique that relies on mathematical models and algorithms to anticipate the system's future behavior and proactively adjust resources to meet preestablished performance and Quality of Service goals. Implementation is carried out using the MAPE cycle, which consists of four stages: Monitoring, Analysis, Planning, and Execution. One work based on this approach is [21].

Queueing Theory is based on mathematical models that describe the behavior of request queues within a system, allowing for the prediction of workload and optimization of performance and availability of applications. By predicting the buildup of requests in the queue and the necessary processing time, systems can anticipate when existing resources will be insufficient and require scaling. This is particularly valuable during unexpected demand peaks, where a swift scaling response can prevent performance degradation and violations of the Service Level Agreement. An implementation based on this technique can be found in [4].

Time Series Analysis employs statistical models to predict future demands based on historical usage data. This allows the auto-scaling system to anticipate demand spikes and adjust resources before they occur, improving efficiency and preventing performance degradation. [37] employs time series analysis to predict future workload demand and adjust resources accordingly.

Lately, various machine learning techniques have played a fundamental role in multiple areas of computing, and resource scaling is no exception. Within ML, different approaches can be found to address the problem of scaling in the cloud.

The most popular is **Reinforcement Learning** and its variant, Q-Learning. These techniques enable an agent controlling the auto-scaling to learn and adapt from interaction with the environment. In the context of auto-scaling, these methods focus on making optimal decisions for resource allocation without explicit guidance, based on rewards and penalties derived from previous actions. For instance, [44] uses this approach.

Another technique involves **Hidden Markov Models** (HMMs). HMMs are a statistical approach that models systems with hidden states. In auto-scaling, HMMs can be used to predict workload patterns or application behavior based on indirect observations. The hidden states represent internal conditions of the system or application that are not directly observable, while the observations are measurable indicators that reflect the system's behavior. The work found in [26] uses HMMs for resource scaling.

A less popular technique involves the use of **Support Vector Machines** (SVMs). This is a supervised learning technique, and it can be used for classifying the state of the system or load, or for predicting values such as future demand. Such a use case can be found in [22].

Although previously mentioned, machine learning and deep learning techniques can be used with the goal of taking time series analysis to the next level. Among the various ML techniques, the use of Recurrent Neural Networks, especially LSTM networks, stands out. LSTM networks' unique capability to remember information over long periods makes them particularly suitable for capturing complex patterns and temporal dependencies in time series data, thus enabling precise predictions about future resource demands, and thereby acting proactively in response to changes in demand.

2.2 Reinforcement Learning

Reinforcement learning is a subfield of machine learning that focuses on learning through interaction with an environment. Reinforcement learning has two elements, a RL agent, and an environment. The agent observes the state of the environment and selects an action based on its current policy, receiving a reward for the action and transitioning to a new state in the environment. The goal of the agent is to learn an optimal policy that maximizes the expected cumulative reward over time.

Reinforcement learning (RL) algorithms can be classified into two main categories: model-based and model-free.

Model-based reinforcement learning algorithms learn a model of the environment, which includes a transition function that captures the probabilities of transitioning from one state to another upon taking a specific action, and a reward function that specifies the reward obtained for each state-action pair. The agent then uses this model to plan its actions by simulating the consequences of different actions and selecting the action that leads to the highest expected reward.

On the other hand, model-free reinforcement learning algorithms directly learn a policy or value function without explicitly modeling the environment. Model-free algorithms estimate the value of a state or state-action pair based exclusively on the observed rewards and transitions from samples. This might require more samples to converge to an optimal policy or value function compared to a model-based method. The most common algorithm in model-free RL is Q-Learning, which learns the optimal action-value function (Q-function) by iteratively updating the Q-values for each state-action pair using Bellman's equation.

In recent years, deep reinforcement learning has grown in popularity. Traditional RL algorithms, such as Q-Learning, use a lookup table to store the Q-values for each state-action pair. However, in high-dimensional state and action spaces, this approach becomes infeasible as the number of state-action pairs grows exponentially as dimensionality increases. This is where deep reinforcement learning (DRL) comes in.

DRL, and specifically Deep Q-Learning, uses deep neural networks to approximate the value function or policy of the RL agent. Deep neural networks are powerful function approximators that can handle high-dimensional input spaces and capture complex relationships between inputs and outputs.

Reinforcement learning (RL) algorithms can also be categorized based on their optimization objectives. Finite Horizon RL [11] focuses on optimizing decisions within a fixed number of steps. This approach is relevant for tasks with a clear endpoint, with the objective of maximizing the rewards within this bounded timeframe. Infinite Horizon RL is designed for scenarios where decision-making extends indefinitely, with the aim of maximizing cumulative rewards across an infinite timeline. In this case, discount factors are often used to ensure the summation of rewards converges, allowing for a stationary optimal policy. Continuous Horizon RL [53] represents a subset of infinite horizon RL, where the environment continuously evolves without a set termination, requiring perpetual adaptation of strategies.

A variant of traditional Q-Learning, N-step Q-Learning [12], extends the traditional technique by incorporating a sequence of future rewards into the update process, rather than relying solely on the immediate reward. This method leverages a trajectory of actions and states over N number of steps to update the Q-values, which can provide a more accurate view of outcomes. Aggregating rewards over multiple steps allows for a smoother and often more stable learning process as it reduces the variance that comes with basing updates on a single next step, which might not fully represent the value of the state-action pair.

This technique is particularly advantageous in environments where rewards are sparse or

delayed, as it allows the agent to foresee and evaluate the outcomes of sequences of actions rather than individual moves.

2.3 Transfer Learning

2.3.1 Introduction

Transfer learning is a complex and highly varied topic that involves transferring knowledge learned in one domain to another. This approach is particularly valuable in machine learning, where it allows models trained on large datasets in one context or domain to be adapted for use in different but related contexts. The essence of transfer learning lies in its ability to leverage previously acquired knowledge to improve learning efficiency and performance in new tasks, reducing the need for extensive data collection and training from scratch. This makes it a valuable technique in fields where data is scarce or expensive to obtain.

2.3.2 Differences Between Domains

To implement transfer learning, it is necessary to have two distinct domains: a source domain, from which we want to extract knowledge, and a target domain, to which we want to transfer the knowledge. Depending on the type of transfer learning, there may or may not be one or more differences between these domains [56]. In the following, the generic differences typically found in transfer learning are outlined, with specific examples of how these differences manifest in reinforcement learning (RL).

- **Data Distribution**
  The state spaces in both domains may differ. This can manifest as variations in the distributions of states or in the observations across different Reinforcement Learning (RL) environments.
- **Feature Space**
  Differences in the feature space. In the case of RL, this is reflected in variations in how states are represented, such as differences in sensory inputs or in the visual representation of environments.
- **Action Space**
  Variations in the action space (the set of all possible actions that the agent can take) between tasks can affect the applicability of a learned policy or value function. This includes differences in the number, type, and consequences of actions.
- **Label Space**
  There may also be differences in the label space, between continuous vs. categorical variables. This translates to the reward structure in RL, where the rewards in the source task might not directly align with those in the target task, necessitating a reevaluation or reshaping of the reward function.
- **Objectives - Reward Function Values**
  The overall objectives or goals of the tasks could differ, leading to different optimal strategies. Even if the state and action spaces are similar, different objectives may require completely distinct strategies for optimal resolution. In the case of reinforcement learning, this is seen as differences in the reward function, specifically the values of the function. These define the task's goal by assigning rewards to state-action pairs, and differences between them can lead to different optimal policies.
- **Transition Dynamics**
  The transition dynamics, which describe the probability of moving from one state

to another given an action, may differ between tasks. This includes changes in the environment rules or the physics of the world.

- **Initial State** μ_0
  Variations in the distribution of initial states from which the agent begins in the environment can influence the learning process and the applicability of transferred knowledge.

Depending on the differences between the two domains, different transfer techniques can be applied to better adjust to such differences.

2.3.3 Categories of Transfer Learning

A primary classification of transfer learning includes Homogeneous Transfer Learning and Heterogeneous Transfer Learning [47]. Homogeneous Transfer Learning occurs when the feature space is the same in both the source and the target domains, and there is an intersection between the labels or, in the case of RL, the actions. The difference between the source and the target then lies in the data distributions, the objectives, or the transition dynamics. On the other hand, Heterogeneous Transfer Learning occurs when there is no intersection in the state spaces and/or no intersection in the labels. This latter case is much more complex.

Within Homogeneous Transfer Learning, a specific classification of transfer learning methods can be derived from [29] and [57]. It is classified into four categories:

- **Instance-Based Transfer Learning**
  This method involves reusing data instances from the source domain during training in the target domain, but by re-weighting the instances. This consists of assigning weights to instances in such a way that greater importance or weight is given to those that better approximate the distribution of the target domain, therefore ensuring that representative data is used to train in the target domain. These samples are used directly for learning in the target domain. This method is particularly effective when the two data domains have similar but slightly different probability distributions.

  In reinforcement learning, instance-based transfer learning can be applied by reusing and re-weighting trajectories or transitions collected in the source domain for training in the target domain. For example, in simulation-to-reality transfer, trajectories from a simulator can be re-weighted based on how well they match the dynamics of the real-world environment. This ensures that the agent prioritizes learning from trajectories that are more representative of the target domain.

- **Feature-Based Transfer Learning**
  This technique is applicable to both homogeneous and heterogeneous transfer learning. With homogeneous problems, these methods aim to reduce the gap between the probability distributions of the source and target domains, by transforming the feature space to a common space between both domains.

  There are two versions. The first approach, Asymmetric Feature Transformation, transforms the feature space from the source domain to that of the target domain. Conversely, Symmetric Feature Transformation modifies the feature space of both domains to a new feature space, generally of lower dimension. This is particularly useful in scenarios where the domains have similar features but distinct distributions.

  In the case of RL, these methods can be applied to adapt state representations between domains. For example, symmetric transformations can project state representations from related RL environments into a shared latent space, facilitating policy transfer. Conversely, asymmetric transformations can be used to map simpler state representations from a source domain into the more complex state space of a target domain.

- **Parameter-Based Transfer Learning**
  This technique involves sharing or transferring model parameters (such as the weights of a neural network) from the source domain to the target domain. This method is particularly useful when the domains are identical between the source and the target, and the tasks are related, allowing the target task to benefit from the pre-trained knowledge of the source task, leading to faster convergence and improved performance. It is commonly applied in deep learning, where transferring the weights of a deep neural network from one source to a target task can significantly increase learning efficiency. In reinforcement learning and DRL, this approach can be used to transfer learned policies or value functions from one environment to another, especially when the environments share similar dynamics or objectives.

  Broadly, there are two ways to perform parameter-based transfer learning: soft weight sharing, and hard weight sharing. In the former, the model is penalized if it significantly deviates from the original weights. In the latter, a simple copy of the model parameters is made.

- **Relational-Based Transfer Learning**
  This method involves transferring the learned knowledge, in the form of relationships or rules common to both domains, from the source domain to the target domain. It is useful when the data from the domains are not independent but are connected by relationships or rules.

  In the context of RL, relational-based transfer learning can be used to transfer knowledge of environmental dynamics or structural patterns. This is particularly applicable in domains where relationships, such as spatial layouts or causal interactions, are of great importance. For example, in a maze-navigation task, the agent could transfer relational knowledge such as "walls block movement" or "shorter paths lead to higher rewards". This relational understanding can then generalize to new mazes with different layouts.

2.4 Cloud Simulation Tools

2.4.1 Introduction

Cloud simulation tools enable developers and researchers to test a number of different aspects of cloud computing [45], including architecture, provisioning algorithms, and multiple performance metrics, in a repeatable manner, allowing for rapid and cost-effective evaluation of various configurations that may not be feasible to test in a real environment. No simulator is capable of fully replicating all aspects of reality [28]. Therefore, it is crucial to select the simulator that most closely matches the requirements to achieve the most accurate results.

In order to identify the most suitable simulator for this work, a comprehensive survey of available tools was conducted. Several key factors were prioritized during the selection process. Firstly, a robust and widely adopted simulation tool was desired. Furthermore, the simulator needed to have built-in resource scaling capabilities, as this would eliminate the need for the extensive time and effort required to develop such functionality from scratch. Lastly, the simulator had to be extensible, allowing for the necessary integration of custom features, such as the external connection to the resource controller. The following sections provide an analysis of the tools evaluated during this survey.

2.4.2 CloudSim

Introduction

CloudSim [8] is an open-source software tool designed for simulating cloud computing infrastructures and services [1]. It enables researchers and developers to model, simulate, and experiment with resource management strategies in cloud environments, without the need to conduct tests on real infrastructures, which can be complex and costly. It supports the simulation of large-scale cloud platforms, including the ability to describe users, applications, data centers, brokers, scheduling, and provisioning policies, without immersing in low-level details. CloudSim is useful for studying both cloud infrastructure and the services it offers, facilitating research in this field.

Functionalities

CloudSim is a versatile simulation tool for the cloud computing environment that offers a wide range of capabilities to model and simulate cloud infrastructures and services. Its main features and capabilities include:

- **Simulation of Data Centers and Virtual Machines**
  It allows the simulation of various data centers, server virtualization, and customized policies for resource allocation to virtual machines (VMs).
- **Resource Modeling and Provisioning Policies**
  CloudSim facilitates the modeling of energy-aware computational resources, including provisioning policies for CPU, RAM, and bandwidth.
- **Network Simulation and Topologies**
  It integrates capabilities to simulate different network topologies and communication latency and bandwidth, which is essential for analyzing applications that heavily rely on data shuffling.
- **Flexibility and Extensibility**
  It supports the dynamic addition or removal of simulation components, enabling users to define and test custom policies for host allocation to VMs and resource allocation to VMs.
- **Event Management and Simulation Entities**
  CloudSim manages queues of future and deferred events, allowing for the pausing, resuming, and creation of new entities during the simulation.
- **Support for Experimentation and Evaluation**
  Through its structure and capabilities, CloudSim provides a controlled and repeatable environment for testing applications and resource provisioning strategies, helping to identify system bottlenecks, and testing different configurations for adaptive provisioning techniques.

Limitations

Despite the extensive capabilities of CloudSim for simulating cloud computing environments, there are some limitations and areas where the tool could be improved or expanded:

- **Graphical User Interface (GUI)**
  CloudSim lacks a GUI to facilitate the configuration of simulations and the visualization of results. While a GUI could make the tool more accessible to a wider range of users, it is not particularly relevant for the use case.

- **Advanced Network Modeling**
  Although CloudSim supports the simulation of network topologies and latencies, its network model is basic and may not be capable of simulating complex or specific network behaviors, such as the precise simulation of network protocols or security attacks.
- **Support for Parallel Experiments**
  CloudSim's capability to handle parallel experiments is limited, which can be a drawback for users who wish to run multiple simulations efficiently and concurrently.
- **Results Analysis**
  It can be challenging to analyze simulation results due to the lack of integrated tools for data analysis and visualization, or even the ability to export the results for further analysis. This may require the development of these features or the use of external tools for detailed analysis.
- **Simulation of Resource Scaling**
  While CloudSim is highly versatile, there may be specific use cases or cloud application features that are not directly supported or require significant extensions to be modeled adequately. One of such cases, which has particular importance in the case study, is cloud elasticity.
- **Support for New Cloud Technologies**
  With the rapid evolution of cloud computing technologies, such as containers and functions as a service (FaaS), CloudSim might need updates or extensions to effectively model these technologies. Within these new technologies, we can find Cloud Elasticity. By default, CloudSim does not have the capability to scale resources during simulation.

Despite these limitations, CloudSim remains a very powerful simulation tool for cloud computing. It has areas of possible improvement and expansion for future versions of the tool, some of which have already been addressed in other tools or even by developers who saw the need to extend and enhance CloudSim's capabilities.

Extensions

There are several simulation tools based on CloudSim, which aim to refine and extend its functionalities. These simulators include CloudAnalyst and CloudSim Plus.

Use Cases

CloudSim has been used in several research papers. The paper [51] explores the use of cloud technology to improve collaborative real-time media services through the Elastic Media Distribution (EMD) project. It presents new resource provisioning algorithms that allow a balance between service response time and cost for audio/video streams mixed with file-based transfers. An extended version CloudSim is utilized to simulate real-time educational collaborations, testing the performance of these algorithms under interactive conditions.

Another study [2], proposes a model for simulating variable resource allocation tailored to cloud-based business processes, aimed at optimizing cost and efficiency. CloudSim is extended to include a unified description model that allows for the simulation of various business process variants. This integration enables detailed analysis of how different resource configurations affect the performance metrics and cost efficiency of business processes.

The work found in [50] focuses on algorithms that enable scalable and elastic real-time A/V collaborations suitable for requirements Service Level Agreements (SLA). CloudSim is extended to not only generate A/V collaboration patterns but also to gather detailed statistics on resource usage, network congestion, and delays. These added simulation capabilities are pivotal in assessing the feasibility and effectiveness of the proposed resource provisioning algorithms across real-time educational scenarios.

2.4.3 CloudAnalyst

Introduction

CloudAnalyst [48] stands out as a simulator that adopts a graphical interface to facilitate the evaluation of social networking tools, considering how the geographic location of users and data centers impacts performance. This simulator is based on CloudSim, extending its functionalities to provide a more comprehensive simulation environment. Among its notable features, CloudAnalyst allows the configuration and management of virtual data centers on an interactive map, which enables detailed observation of load balancing, monitoring of cloud clusters, and analysis of real-time data flow. It also offers the ability to perform multiple runs of the same experiment, facilitating the comparison and analysis of results, and the capability to present the results visually through charts, improving the interpretation and understanding of the data.

Functionalities

CloudAnalyst provides the same functionalities as CloudSim, but extends the latter by adding new features. Firstly, it adds a Graphical Interface, which allows users to configure experiments quickly and easily. Additionally, it introduces the capability to generate graphical results of the experiments in the form of tables and charts, facilitating the analysis and comparison of data. It also enables the ability to store input parameters and results of experiments in XML files for repetition, ensuring identical outcomes in repeated executions. Moreover, it extends CloudSim by adding the capability to include information about the geographic location of users and data centers, to consider these factors in the simulation. Finally, it integrates policies for resource allocation and data center selection, including strategies for sharing the load during peak demand periods.

Limitations

Being based on CloudSim, it shares several of the limitations already detailed; although, as mentioned in the previous section, it resolves some of them.

Use Cases

Most use cases appear to focus on the study of different load-balancing methods. Within these, we can find the following: The study [34] proposes a framework based on load-balancing strategies inspired by nature, using CloudAnalyst to develop a load-balancing algorithm that can adapt to changing workload patterns in real time. CloudAnalyst is used to explore various load-balancing techniques to efficiently distribute tasks.

The work [18] presents a simulation-based evaluation of three load balancing algorithms: Round Robin, Equally Spread, and Location-Aware, using the tool CloudAnalyst. It demonstrates that Equally Spread surpasses the other two algorithms in terms of balancing efficiency and fault tolerance.

2.4.4 iCanCloud

Introduction

iCanCloud simulator [27] is a flexible and scalable tool for simulating cloud infrastructures. The goal of iCanCloud is to predict the trade-offs between cost and performance of applications running on specific hardware and to provide end-users with information about these trade-offs.

Functionalities

The simulator is characterized by offering a robust and scalable platform for the simulation of cloud infrastructures, notable for its capacity to model both existing cloud architectures and those not yet in existence; therefore, providing a versatile tool for experimentation in this field. One of its most important functionalities is the inclusion of a global hypervisor that supports extensive customization, allowing the integration of any cloud brokering policy, which adds significant flexibility when simulating different resource management policies. Moreover, iCanCloud accurately simulates the types of instances offered in specific cloud infrastructures, such as Amazon EC2, facilitating simulations that approximate real-world deployment scenarios.

Another important feature is its user-friendly graphical interface, which facilitates both the setup and launch of complex simulations, from a single virtual machine to cloud computing systems composed of thousands of machines. This, combined with the platform's extensibility, where new components can be added to expand the simulator's functionality, makes iCanCloud a powerful tool for research and development in the field of cloud computing.

Limitations

The simulator does not provide the capability to scale resources during simulations; instead, the framework must be extended to support this functionality. Another limitation is the very limited number of use cases, which makes it a less attractive option.

Use cases

No relevant use cases were found that employ iCanCloud as a simulator tool.

2.4.5 CloudSim Plus

Introduction

CloudSim Plus [40] is a modern extension of CloudSim, also open source, designed to model and simulate cloud computing infrastructures and services with new features and improvements. It uses Java 8, offering a cleaner API and functionalities such as dynamic creation of VMs and Cloudlets, VM scaling, and VM migration mechanisms. Its goal is to overcome the limitations of CloudSim, providing more flexibility and efficiency in cloud environment simulations.

Functionalities

CloudSim Plus enriches CloudSim by adding several new functionalities. One of the key contributions of CloudSim Plus is its ability to simulate the dynamic creation of VMs and dynamic arrival of Cloudlets, as well as the prioritization of Cloudlets, which allows for the creation of more realistic and detailed simulations. This is particularly useful when exploring resource management strategies, load balancing, and scaling policies in cloud computing environments.

Another functionality it adds is the scaling of VMs, both vertically (changes in a VM's resource capacity) and horizontally (adding or removing VMs), which is crucial for the case study. It also adds mechanisms and strategies for VM migration to optimize resources. Additionally, it offers greater flexibility in simulation, allowing real-time adjustments and the application of custom policies without the need to restart the simulation. These features address the needs of more complex and realistic simulations of cloud environments. CloudSim Plus also introduces a set of classes and interfaces that significantly simplify the extensibility

of the framework. For example, brokers, VM allocation policies, and Cloudlet schedulers have been redesigned to allow users to easily define their own policies and algorithms. Lastly, the framework integrates integration tests and supports code coverage reports, which ensures a more reliable codebase (although it still needs much improvement).

Limitations

As CloudSim Plus is based on CloudSim, it shares many of the same limitations. However, it does add new functionalities, including the capability to model Cloud Elasticity. Additionally, for those use cases that are not covered by the default functionalities, it is simpler to extend the simulator to include such capabilities.

Use Cases

The paper [42] develops a system named P-Cloud that leverages surplus computational resources such as laptops, PCs, and servers to create a cost-effective cloud infrastructure. This system allows resource owners to monetize idle resources while providing users with more affordable cloud services. CloudSim Plus is crucial in implementing and evaluating P-Cloud by simulating the customized cloud environment, managing resources dynamically, and testing various pricing and user satisfaction policies.

Another study [33], addresses the complex problem of task scheduling in cloud environments by introducing the Brain Drain Optimization (BRADO) technique to minimize the makespan of tasks. CloudSim Plus is utilized to model and simulate the virtual resource scheduling, allowing for the comparison of BRADO's performance against traditional algorithms like Particle Swarm Optimization and Simulated Annealing, demonstrating its effectiveness in optimizing task allocations.

The work [19] explores server allocation strategies for system deployment in edge computing, aiming to minimize average response times across geographical regions. It examines both flat and hierarchical deployment models for edge clouds. CloudSim Plus provides a simulation framework capable of modeling edge computing infrastructures, facilitating high-fidelity simulations that validate theoretical findings and support strategic deployment decisions.

2.4.6 Selected Simulator

Among the simulators that were researched, the decision was made to evaluate two of them: CloudSim and CloudSim Plus. CloudAnalyst was excluded as it did not seem to offer significant advantages over CloudSim, since the functionalities it adds are not particularly relevant to the case study. The decision not to choose iCanCloud was primarily justified by its apparent lack of adoption within the academic and professional community. This suggests that there might be less community support, which in turn implies a lesser degree of robustness and a higher likelihood of encountering stability issues. Furthermore, the scarcity of documented use cases or implementation examples makes it difficult to learn and assess its suitability for specific use cases.

Among the two remaining options that were evaluated, the final choice was to use CloudSim Plus over CloudSim. The reasons for this decision were as follows:

- **Resource Scaling**
  CloudSim Plus inherently has the capability to dynamically modify resource capacity. It supports both vertical and horizontal scaling, although the latter with considerable limitations. This feature is crucial for the case study, so having this capability already available is a considerable advantage.
- **Robustness**
  A second factor influencing the decision was the simulator's robustness, both in terms

of its behavior and the results obtained. Several tests were conducted on both simulators, and it was observed in various situations that CloudSim's behavior was not as expected, and occasionally the results obtained were unreliable. For this reason, despite CloudSim Plus also having its defects and various bugs, it was considered the more robust and reliable option.

- **Ease of Extension**
  Lastly, CloudSim Plus is better modeled and organized, which facilitates the understanding of its design and behavior. This enhancement, together with the incorporation of new classes and interfaces, significantly simplifies the adaptation of the simulator's behavior to meet the specific requirements of individual cases.

Chapter 3

Literature Review

This chapter reviews the current state of the art in reinforcement learning applications for cloud elasticity, as well as the latest advancements in transfer learning techniques.

3.1 Reinforcement Learning

3.1.1 Introduction

In this section, various solutions based on reinforcement learning are explored to address the problem of resource scaling in cloud environments. The solutions studied range from the simplest models of reinforcement learning to more complex variants.

3.1.2 State of the Art

Within the studied solutions, there are various degrees of complexity. The simplest solutions implement a basic version of Q-Learning.

An example of this is the study [15], which employs the Q-Learning algorithm in the context of cloud applications requiring dynamic resource scaling to adapt to workload fluctuations. As in several other studies, the Q-Learning algorithm is responsible for selecting the scaling action.

Given the inherent complexity of the problem, which involves a continuous search space, it is essential to utilize techniques such as tile coding to reduce the dimensionality of the search space. Tile coding is a form of feature representation that involves partitioning the input space into a set of overlapping tiles or grids. Each tile represents a specific region of the input space and acts as a feature that can be active or inactive based on the input. Alongside tile coding, function approximation is employed to estimate the reward function. Instead of maintaining a reward value for every possible state-action pair, which is unviable in large spaces, function approximation generalizes reward values from observed states to unseen ones based on mathematical models.

Additionally, the use of Q-value initialization based on domain knowledge and Speedy Q-Learning is proposed to accelerate convergence speed. The controller's performance is evaluated through experiments on the CloudSim platform, using metrics such as the total cost incurred during a test period and the observed response time, considering the trade-off between the cost of allocated resources and penalties for SLA violations.

A similar solution is presented in [16].

Another instance where Q-Learning is applied in this context is [20]. This study has two objectives: firstly, to determine the amount of resources needed to meet demand, and secondly, to select where to execute tasks. Like many other works, it is implemented in the context of web applications hosted on Infrastructure as a Service (IaaS) platforms, but notably, this study evaluates the controller's performance through real experiments conducted on Amazon EC2 rather than simulators, using real web applications with variable workloads. As a notable feature, in order to accelerate training, different servers publish a copy of the Q-values, allowing all controllers to update their databases with the Q-values from the others.

Another study that adopts reinforcement learning is [36]. The paper addresses the efficient and dynamic management of scaling for microservices-based applications running in cloud environments, such as Amazon EC2. Unlike other papers, the reinforcement learning algorithm does not determine the scaling action to be taken; instead, policies based on dynamic thresholds are used, and the values of these thresholds are controlled with a reinforcement learning algorithm. Two types of reinforcement learning policies are used: a model-based, called "MB Threshold", and a model-free, called "QL Threshold". To accelerate the learning phase, the "MB Threshold" (model-based) policy utilizes an approach that leverages (or estimates) knowledge about system dynamics to update the Q function. For the "QL Threshold" (model-free) policy, no specific acceleration technique is applied, which may result in a slower learning rate.

More complex alternatives include the use of deep learning-based algorithms. Deep Learning has significantly impacted machine learning algorithms, including RL. Integrating deep learning with reinforcement learning heightens the understanding of the environment and addresses one of the major challenges faced by traditional Q-Learning: the handling of high-dimensional spaces.

One such instance is DERP [6], which aims to manage resource elasticity in the cloud automatically, optimizing dynamic resource allocation based on dynamic workload demands.

DERP proposes three variants which are based on Deep Q-Learning algorithms: one using the traditional Deep Q-Learning, another using Fully Deep Q-Learning, and a third employing Double Deep Q-Learning.

In the first variant, Deep Q-Learning, a neural network calculates the expected reward of various actions for a given state, and using gradient descent and backpropagation, the network is trained to adjust to the observed values in the environment. In the second variant, Fully Deep Q-Learning, there are two neural networks with the same architecture. The difference compared to the previous variant lies in the training process, where the main network is copied to a secondary one, and the weights of the main network are frozen. This main network, which remains static, is used to calculate the predictions and the loss function, while the parameters of the secondary network are adjusted. Periodically, the updated network is copied back to the non-updated one. This aids in stability during training. The third implementation is based on the observation that networks tend to overestimate the values of the Q-values, but there is reason to believe they do not do so uniformly across all actions [46]. Two networks are used during training to address this, the primary neural network and the target neural network. The primary one is used to determine the best action, while the Q-value of that action is obtained from the target network. The weights of the primary network are updated after each episode while the target network is updated by copying the values from the primary network after a predefined number of steps.

Another interesting feature of this implementation is the use of an Experience Replay, where previous experiences or transitions are stored in a buffer, and then deployed during batch training.

In order to validate the performance of the DERP, the models are tested both in sim-

ulations and in a real cluster environment using the Okeanos cloud service, and applied to NoSQL database environments. To evaluate the performance, the results are compared with more traditional RL approaches such as thresholds and decision trees to demonstrate DERP's effectiveness. Compared to the traditional Deep Q-Learning approach, both variants show noteworthy improvements in all stages of the training process.

The paper [17] presents a particular architecture called ADRL. While it implements DRL to tackle the main problem, it employs other techniques to prevent some of the issues that may arise when using DRL.

ADRL focuses on cloud computing environments for cloud-hosted applications, particularly in situations where it is crucial to manage resource scalability efficiently to adapt to variable workloads and ensure the satisfaction of Quality of Service (QoS) agreements.

The authors argue that traditional RL algorithms often have a specific problem: the agent will attempt to take some scaling action even when it is not necessary, which can lead to loops of increasing and decreasing resource actions. A preliminary stage is used to counter this problem. This stage involves determining whether the system is in an anomalous state, that is, whether it is truly necessary to take action. If so, an RL agent is called upon to select a scaling action.

The solution consists of two modules. This first module, which validates whether action is necessary, has itself two components. The first component predicts the future state of the system. To do so, it gathers the utilization metrics that form the state and makes a future prediction of these metrics using a neural network. The second component takes these future utilization predictions and uses an IForest algorithm to identify whether the system is in an anomalous state and needs a scaling action to recover. If the first module deems that an action is required, then the RL agent selects a scaling action in order to return the system to a stable state.

The performance of the ADRL system is evaluated through simulations using CloudSim, allowing for detailed modeling of the cloud environment and enabling comprehensive evaluations under varying loads and anomalous events. The evaluation includes comparisons with benchmark methods and existing techniques to demonstrate the effectiveness of ADRL compared to reinforcement learning approaches without anomaly awareness.

3.1.3 Resources to Scale

In any system, there are various resources that can be a limiting factor for the processing speed of tasks. These resources might include CPU, RAM, I/O, and network bandwidth [10]. Among the researched literature, some algorithms are capable of scaling various resources, while most focus on managing a single one. A common resource across all papers, and often the only one scaled, is the CPU. This is typically the primary bottleneck in task processing speed and, therefore, receives the most attention. However, the specific resource causing a bottleneck can vary depending on the task and its modeling. In more complex scenarios, this limiting factor may shift over time in response to changing demands, or it might even involve a combination of multiple resources. Therefore, some studies scale not only the CPU but also the memory, network bandwidth, and even disk capacity, allowing for the modeling of more complex problems and further optimizing the use of the resources being deployed.

3.2 Transfer Learning

3.2.1 Introduction

This section studies a variety of transfer learning techniques, subsequently exploring solutions for implementing transfer learning with reinforcement learning algorithms. Additionally, it

mentions different metrics for measuring the performance of an algorithm after applying transfer learning.

3.2.2 Transfer Learning Methods for Reinforcement Learning

The work found in [56] reviews various transfer learning methods that can be applied in the context of reinforcement learning, particularly deep reinforcement learning. It also presents different scenarios in which these methods are applicable.

An important subcategory of transfer learning is Domain Adaptation, which focuses on scenarios where the source and target domains differ but share some underlying structure. The goal of domain adaptation is to bridge the gap between these domains, often by aligning their feature spaces or addressing discrepancies in their dynamics. This concept underpins some of the methods discussed below, particularly those designed to handle variations in the state space, action space, or reward space between domains.

Reward Shaping

The goal of reward shaping is to modify the reward obtained from the reward function with an additional reward function derived from prior knowledge of the target domain, aiming to influence the agent's decision-making to maximize performance in the new domain, or to optimize exploration and achieve faster convergence to the optimum. The new reward function will be $R' = R + F$, where F is a function that can be learned. One method to obtain F is from the potential difference between two states:

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s)$$

The quality of a state is given by the value of Φ which contains the prior knowledge.

Learning from Demonstrations

As the name suggests, learning from demonstrations trains the agent using examples of actions that it aims to imitate, which can come from an expert (possibly human) or a sub-optimal policy.

Demonstrations are formatted as (s, a, s', r), typically with the source and target domains being the same.

There are two variants of the algorithm, online and offline. In the online variant, demonstrations influence decision-making during training to enable more efficient exploration. In the offline variant, the agent is initially trained using external demonstrations followed by traditional training.

Specifically, for Q-Learning, two models are presented: The first one is called DQfD, which uses two replay buffers. The first one is filled with data generated during exploration while the other one with the demonstrations, which remains fixed. During training, data from both buffers is sampled, ensuring that data from the demonstrations are used in a certain proportion. The second model is LfDS. It creates a function to derive the potential value of a state-action pair based on the highest similarity between a given pair and the expert experiences. Using this function alters the reward function value, assigning higher values to actions similar to those of the expert. This leads the agent to behave similarly to the expert.

Learning from Demonstrations is especially valuable in scenarios where autonomous exploration is risky, costly, or inefficient.

Value Function Transfer

The value function transfer technique in reinforcement learning focuses on transferring the value function learned in a source task to enhance or accelerate learning in a target task. The value function, which estimates how much "value" or expected return can be obtained by following a specific policy from a given state, is crucial in reinforcement learning for determining the quality of actions and states. In the context of transfer, the idea is that if a source task is somehow similar to the target task, then the learned value function (or insights on how to act in different states) can be transferred and adapted to aid in learning the new task, potentially reducing the time and data needed to effectively learn the new task. However, this transfer technique is not applicable to purely policy-based methods that do not estimate a value function, as it relies on the direct transfer of value functions rather than policies.

Policy Transfer

Policy transfer involves transferring knowledge from one or more expert policies from the source task to the target task. There are several variants of this technique. One such variant is policy distillation, a supervised learning technique where knowledge from one or more teachers' policies is 'distilled' to a student agent. The goal is to minimize the difference between the action distributions of the teacher(s) and the student. This involves using trajectories from the teacher's policy to minimize the probability distribution differences between both policies, referred to as "teacher distillation". Conversely, "student distillation" uses trajectories based on the student's policy.

Another variant, Policy reuse, is specifically designed for scenarios with multiple expert policies. It directly exploits these expert policies by learning a probability distribution to identify which expert policy to use to maximize results in the target domain.

Inter Task Mapping

This method involves learning a mapping function from the Source task to the Target task. Earlier approaches assumed mapping functions were predefined, but more recent work aims to learn the mapping automatically. The goal is to learn mapping functions that translate the state and actions from one domain to another, allowing for the reuse of learned policies. To train this function, one paper suggests training both models in a common space to deduce the mapping function.

Reusing Representations

This technique generates a universal policy from multiple policies trained for different tasks within the same state domain. One particular study uses a progressive neural network structure, which is progressively trained for each distinct task. This network features a unique structure with multiple columns, each representing a neural network dedicated to training a specific task. The network starts with a single column for the first task, and as new tasks are introduced, additional columns are added. During the training of a new task, the weights of the neurons in previous columns are frozen, and the representations from these frozen tasks are applied to the new column via lateral connections, aiding in the training of the new task. However, this technique has the drawback of creating a large neural network that grows proportionally with the number of original tasks.

Learning Disentangled Representations

The objective of disentangled representation methods is to learn feature state represen-

tations that are independent of domain-specific features, thereby decoupling the specific characteristics of the MDP from the reward distributions.

Two variants of the method are Successor Representation and Universal Function Approximation.

Successor Representation

This method aims to decouple state characteristics from rewards. It enables transfer learning between multiple tasks, as long as they only differ in the reward function. Unlike traditional Q-Learning, where the Q-value describes the state in relation to the reward function, successor representation describes states based on the occupancy measure of its successor states, which is the unnormalized distribution of states or state-action pairs an agent encounters while following a specific policy (π) in the MDP. In other words, it represents the likelihood of reaching a particular state, given the current state and action, essentially decoupling the dynamics of the environment from the rewards. Therefore, it decomposes the value function of any policy into two independent components:

$$V^{\pi}(s) = \sum_{s'} \psi(s, s') w(s')$$

The first component, $\psi(s, s')$ describes a state s in terms of the occupancy measure of its successor states and is agnostic to rewards, so the encoded knowledge can be reused in the other tasks. The second component $w(s')$ is a function that maps states to rewards, and it naturally depends on the reward.

Universal Function Approximation

As the previous method, this technique aims to decouple specific state characteristics in the reward domain. Therefore, transfer can only occur when tasks differ solely by their rewards.

This technique uses Matrix Factorization to generate two embeddings, one for states and one for goals. These embeddings are decoupled from each other. The construction of these embeddings involves two steps: The first step is to create a matrix with states as rows and goals as columns, and the value in a given position of the matrix is the reward for being in that state for the corresponding goal. Low Rank Factorization is performed to find the target embedding of the states, and the target embedding of the goals. In the second step, regression is used to learn the embedding of both networks.

One advantage of UFA is that it is transferable for all tasks that differ only by their goals.

Sim-to-Real Transfer

Sim-to-Real transfer refers to the process of transferring reinforced learning policies from simulated environments to reality. This necessity comes due to limitations in collecting real-world data, such as sample inefficiency and collection costs [54]. Simulated environments provide a potentially infinite data source and mitigate safety or practicality concerns. However, simulators are not perfect and cannot simulate every detail, resulting in discrepancies between simulated and real environments. These differences diminish the performance of policies once they are transferred to real environments.

Several research efforts focus on bridging this Sim-to-Real gap to achieve more efficient policy transfers, including learning with perturbations or domain adaptation. One technique focuses on introducing perturbations in simulated environments to make agents less susceptible to differences between simulation and reality. Another technique involves using data from the source domain to improve the performance of a model learned in a different target domain, where data is less abundant. It attempts to unify the feature spaces of both domains to facilitate the transfer of knowledge.

3.2.3 Evaluation Metrics

There are various metrics for objectively evaluating the effectiveness of transfer learning.:

- **Jumpstart Performance (JP)**
  This is the initial performance (outcomes) of the agent. Comparing the results with and without knowledge transfer provides a measure of how successful the knowledge transfer was.
- **Asymptotic Performance (AP)**
  This is the final performance of the agent after re-training, with and without knowledge transfer.
- **Accumulated Rewards (AR)**
  The area under the learning curve of the agent with and without knowledge transfer. This metric shows the agent's performance during the initial phase of training after transfer, evaluating how effective the transfer is and how long it maintains a performance advantage.
- **Transfer Ratio (TR)**
  The ratio between the asymptotic performance of the agent with TL and the agent without TL.
- **Time to Threshold (TT)**
  The learning time (iterations) required for the target agent to reach certain performance thresholds, with and without knowledge transfer.
- **Performance with Fixed Training Epochs (PE)**
  The performance achieved by the target agent after a specific number of training iterations, with and without TL.
- **Performance Sensitivity (PS)**
  The variation in results using different hyperparameter settings, with and without TL.

3.2.4 State of the Art

Although considerable research has been conducted on reinforcement learning algorithms to control resource scaling, none have adopted transfer learning as a technique to accelerate training. Therefore, this section will focus on papers that make use of various transfer techniques on reinforcement learning models to accelerate training, but not necessarily for use cases regarding resource scaling.

In the study [25], the authors address the optimization of Radio Access Network (RAN) slicing for 5G. The research focuses on the application of transfer learning techniques within the context of deep reinforcement learning. A notable aspect is the focus on meeting Service Level Agreements (SLAs) by defining a reward function. The proposed approach involves training a model on a source base station (BS), followed by transferring the learned parameters to a target base station. Traffic loads differ between the source and target stations, naturally resulting in imperfect transfer. The authors emphasize that this approach is equally applicable when initial training is conducted in a simulator, although caution is raised that this might lead to suboptimal policies. Performance is evaluated with various loads and using multiple DRL variants, but results are presented only when using A2C (Actor-Critic) and PPO (Proximal Policy Optimization) algorithms, and in all scenarios, it achieves improved performance.

In the article [24] the challenge of network slicing for IoT devices is addressed. With the growth in the number of IoT devices and the emergence of new applications, it has become

more challenging to comply with different Quality of Service (QoS) levels on the same physical network. The goal is to partition the physical network into multiple virtual networks, each tailored to different business requirements, to improve QoS, energy efficiency, and reliability. To perform the transfer, initially, a central controller collects data from multiple distributed gateways and constructs a replay buffer. With this data, the model is trained, and once it converges, the trained models are sent to the gateways to be used as a starting point. It is found that the use of transfer learning results in faster convergence, although it starts from an equally unfavorable point compared to other algorithms without transfer.

The article [5] focuses on the detection of Economical Denial of Sustainability (EDoS) attacks in virtual Content Delivery Network (vCDN) environments. These attacks represent a significant threat in cloud-based network environments, where they can cause misuse of resources and an increase in operational costs. To address this challenge, the study proposes a framework that uses DRL with a two-stage neural network; one for feature extraction and the other for forecasting.

The objective of the transfer learning process is to transfer knowledge from one vCDN to another. For this, the weights of all layers from the feature extraction stage of the model from one vCDN (source) are copied to another (target), while the weights of the forecasting layer are reset. Subsequently, the target vCDN is retrained, freezing the weights of the feature extraction stage and allowing only the modification of the forecasting stage weights. For this retraining, a few data points from the target vCDN are used.

To evaluate the effectiveness of the proposed method, a comparison is made between different approaches. The first approach copies the model trained on vCDN1 and applies it to vCDN2, a second one trains a new model from scratch with the few available data of vCDN2, and a third utilizes transfer and retraining as previously explained. The results demonstrate that the third approach, which involves the use of transfer learning, proves to be the most effective, achieving the best performance among the three approaches.

As previously mentioned, the transfer techniques used are relatively simple for cases similar to the case study. To discover the use of more advanced techniques, it is necessary to look at use cases that differ significantly from the case study. Nonetheless, some are discussed here.

The first technique can be found in [13]. The primary objective of this work is to present an algorithm, Deep Q-Learning from Demonstrations (DQfD), which is a Deep Q-Learning algorithm that leverages small data sets of demonstrations to massively accelerate the learning process. The particular case study focuses on Atari games, with demonstrations obtained from human experts, but the algorithm is not limited to games and is also applicable to other use cases including autonomous vehicles, recommendation systems, and other scenarios where prior control data is available but generating new data through simulation is difficult or impossible.

The training of DQfD is composed of two stages: an initial stage where training occurs solely with demonstration data, and a second stage where training involves interaction with the environment. The objective of the first stage is to ensure that the agent can mimic the behavior of the expert, while satisfying the Bellman equation to achieve good performance once training against the environment begins. For this purpose, four loss functions are used:

- 1-Step temporal difference
- N-Step temporal difference
- Supervised large margin classification loss
- L2 regularization loss on the network weights

The objective of the first two losses, the 1-Step and N-Step temporal difference losses, is to keep the Q-values updated and to satisfy the Bellman equation, while the supervised loss is for classifying the expert's actions. The regularization loss aims to prevent overfitting.

The supervised loss is critical for the effectiveness of pre-training. Given that the demonstration data only covers a narrow part of the state space and does not involve all possible actions, many state-action pairs have never been explored and it is not possible to assign them realistic values. If pre-training relied solely on the Q-Learning training function, which updates Q-values based on the maximum value of the next state, it would inadvertently propagate unrealistic values throughout the Q-function. This could result in distorted predictions, undermining the agent's ability to generalize effectively during subsequent training against the environment.

After this pre-training phase, the agent begins to interact with the environment, collecting new data that is added to the replay buffer until it is full. It is important to note that the demonstrations are never overwritten. During training, the network is updated with a mix of self-generated data and the demonstration data. The technique uses a prioritized replay mechanism to automatically control the proportion of demonstration data during learning, allowing the agent to improve its policy based on the demonstrations and its own experience. The same losses are used for training, with the exception of the supervised loss, which is not typically used in RL.

To show the effectiveness of the technique, a comparison was made with a Prioritized Dueling Double DQN algorithm, and DQfD demonstrated superior initial performance in nearly all games during the first million steps and achieved target performance levels much faster than PDD DQN. It was also shown that DQfD surpasses three related algorithms for incorporating demonstration data into DQN.

The study [49] introduces an innovative approach to partial domain adaptation using deep reinforcement learning. The key to this method is to select only those examples from the source domain whose probability distribution is similar to that of the target domain. The aim is to avoid "negative transfer", which occurs when non-representative examples of the source domain are used.

The results of the study are tested in the context of image classification. It is observed that the inclusion of this algorithm consistently improves results compared to methods that do not leverage it. The proposed approach proves particularly effective in filtering out source domain examples that could induce negative learning, thereby optimizing the model's adaptation to the target domain.

The objective of [7] is to obtain an algorithm that accelerates the learning phase of reinforcement learning, which is one of its main limitations in many use cases. To achieve this, the authors propose a technique that integrates expert demonstrations into the RL process, using reward shaping to accelerate learning. This approach allows for the use of human input without erroneously assuming the optimality of the demonstration, aiming to significantly reduce the number of necessary demonstrations, increase robustness against the suboptimality of the demonstrations, and achieve faster learning.

The use case explores how to integrate expert demonstrations to accelerate reinforcement learning in tasks such as balancing a pole (a classic problem in reinforcement learning known as "Cart Pole") and navigating complex game environments like the Mario domain. These scenarios represent problems in which RL alone would require a large number of interactions to learn an effective policy due to sparse reward signals.

The proposed technique uses demonstrations from an expert to influence the RL agent's exploration process. Specifically, it suggests modifying the reward function with a function based on the similarity between the actions taken by the agent and those demonstrated by the expert, calculating this similarity using a Gaussian function. This new reward function

encourages the exploration of state-action pairs similar to the demonstrations first, guiding the agent's learning. These state-action pairs from the demonstrations represent states with higher potential, and therefore the outcome of this guided exploration is faster learning. It is noteworthy that these modifications do not affect the Q-values and therefore the algorithm will not be negatively impacted in the case of poor demonstrations.

Experiments demonstrate that the technique used can significantly improve learning efficiency compared to traditional RL and other techniques that integrate demonstrations. In the "Cart Pole" problem, the proposed technique allowed the agent to learn effective policies with far fewer demonstration samples. In the Mario domain, the technique proved to be more efficient in using samples and more robust against suboptimal demonstrations, including those provided by humans, which can exhibit great variability and potentially contain errors.

The study [14] focuses on separating informative (specified) and non-informative (unspecified) variation factors in data. The goal is to eliminate irrelevant factors that may negatively impact learning while at the same time enabling the generation of new samples.

The proposed solution relies on the use of variational auto-encoders (VAEs) and cycle consistency. A VAE comprises two components: an encoder and a decoder. In this case, the encoder's function is to partition the input into two components: one specified which is specific to a given label, and unspecified with non-informative label data.

$$\text{Enc}(x) = (f_z(x), f_s(x)), \quad \text{with}$$

$f_z(x)$ being the unspecified encoding and$f_s(x)$ being the specified encoding

The decoder then recombines these two components to regenerate the original input.

A fundamental aspect of the VAE's functionality is Cycle Consistency. This specifies that consecutive transformations should result in the identity function:

$$G(F(x_i)) = x_i', \quad x_i \sim x_i',$$

$$F(G(y_i)) = y_i', \quad y_i \sim y_i'.$$

The training is semi-supervised and consists of two alternating phases: a forward cycle and a reverse cycle. If only the forward cycle is utilized, a degenerate solution might arise where specific factors are completely ignored, and all information stems from unspecified factors. This issue is corrected by the use of the reverse cycle.

Forward cycle

Two images x_1 and x_2 with the same label are taken. Both images are processed through the encoder, resulting in

$$\text{Enc}(x_1) = (z_1, s_1), \quad \text{Enc}(x_2) = (z_2, s_2)$$

The input to the decoder is constructed by swapping the s_i components. Then

$$\text{Dec}(z_1, s_2) \approx x_1, \quad \text{Dec}(z_2, s_1) \approx x_2$$

Given that s_1 and s_2 contain the label-specific data, their content should be the same; hence, swapping should yield the same result.

Obs: the actual labels are never needed.

Reverse Cycle

For this second stage, a point z_i is sampled with $P(z) = \mathcal{N}(0, I)$. Then, it is combined with s_1 and s_2, with $s_1 = f_s(x_1)$, and similarly for x_2. The decoder receives these combinations:

$$\text{Dec}(z_i, s_1) = x_1'' \quad \text{and} \quad \text{Dec}(z_i, s_2) = x_2''$$

Then, $z_1'' = f_z(x_1'')$ and $z_2'' = f_z(x_2'')$ are obtained.

Since both z_1'' and z_2'' were generated using the same z_i, they should be mapped close to each other. This step leads to the learning of disentangled representations between Z and S.

Obs: It is not necessary for s_1 and s_2 to have the same label.

Instead of using label-specific data for semi-supervised learning, the study utilizes domain-specific data. During training, domain-specific information is not used; rather, information that is independent of the domain is employed. This approach allows for a clearer and more useful representation of the data, facilitating tasks such as classification or the generation of new samples that are domain-relevant but independent of their specific factors.

The model was tested on three datasets: MNIST, 2D Sprites, and LineMod. The experiments were divided into evaluating the quality of disentangled representations and the image generation capabilities. The results show that classification accuracies on the specified latent space were very high, indicating effective learning of specified factors, and accuracies on the unspecified latent space were low, suggesting good disentanglement. The visual inspection of the generated images demonstrated that the model effectively managed to keep specified and unspecified factors separate, with minimal leakage of specified information into the unspecified latent space. The work concludes that cycle-consistent VAE architecture prevented degenerate solutions and was robust to variations in the dimensionality of latent spaces.

The work [52] employs Cycle-Consistent Variational Auto-Encoders (VAEs), similarly to the previous study, although the objectives differ.

This study focuses on enhancing domain adaptation in reinforcement learning through a Latent Unified State Representation (LUSR). The aim is to generate two representations from a given instance of a specific domain: one containing domain-specific information and another independent of the source domain. Subsequently, the domain-independent representations are used to train the desired reinforcement learning model. Thus, the trained model will be capable of operating across instances from any domain once the domain-independent information has been extracted.

The algorithm is appropriate when the state space differs between domains, but the actions, reward functions, and transition probabilities are similar.

The algorithm is tested in two different challenges: the first one is a 2D car racing game (with variations in terrain color for different domains), and the second one is in the context of autonomous driving using CARLA (with changes in daylight conditions). In both scenarios, the algorithm demonstrates highly positive results, even surpassing models that are specifically trained for a single domain.

This approach to domain adaptation in RL through Latent Unified State Representation proves to be a powerful and versatile technique. By segregating domain-specific and general characteristics and focusing RL training on the latter, effective adaptation to various domains is achieved without compromising the effectiveness of learning. This method opens new possibilities for RL applications in environments where differences in state spaces pose significant challenges for transfer learning.

This page intentionally left blank.

Chapter 4

Design and Implementation

The purpose of this chapter is to describe the implemented solution and its design in detail, outline its components, the interactions among them, and discuss and justify the decisions made. Additionally, links to the implemented solution are provided.

4.1 Deep Reinforcement Learning Controller

This section provides a detailed description of the deep reinforcement learning controller model, outlining its architecture and functionality while also justifying the decisions made during its development.

4.1.1 Utilized Algorithm

From its inception, the approach selected was to employ a Reinforcement Learning algorithm, specifically, Deep Q-Learning. The rationale for this decision is as follows:

- **Model-Free:** It is unnecessary to learn or model the environment, which is not only inherently complex and dynamic but also non-deterministic, as the transition dynamics depend on user behavior, which contains a random component. The neural network-based model is capable of implicitly learning these transition dynamics and identifying the actions that optimize rewards.
- **Ability to Handle a Very Large State Space:** One of the major challenges observed during the research was managing the extensive size of the state space. The complexity is not only due to the high number of input variables but also because these variables are continuous. Initial approaches such as tile coding offer a rudimentary approach to the problem, however, employing a neural network allows for a controller with a superior ability to model the environment.
- **Optimization of Long-Term Decision Making:** Deep Q-Learning or Deep Q-Networks do not only seek to maximize immediate rewards but also considers the long-term value of its actions. This is crucial for cloud resource management, where decisions not only have immediate impacts in terms of costs and performance but also long-term consequences that can affect the stability and scalability of the system.
- **Continuous Learning:** Contrary to other algorithms, Q-Learning algorithms do not follow the train-then-deploy paradigm; they are constantly learning. This presents a significant advantage, as the dynamics of the environment may slowly evolve over time, which could significantly impact performance from models developed with other algorithms. However, with DQN, the model continually adapts to these changes.

- **Adaptability to Different Elasticity Policies:** The DQN framework exhibits adaptability to various elasticity policies, enhancing its applicability across diverse operational scenarios within cloud environments.

N-Step Deep Q-Learning

In this research, a variant of Q-Learning known as **N-Step Q-Learning** was employed. In traditional Q-Learning, the model learns through interactions with the environment. Given a state of the environment, the algorithm must make a decision that, when applied, results in a new state and a reward. This information is used to update the model.

$$Q_{\text{new}}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

The challenge arises from the fact that a scaling action has an almost imperceptible effect immediately after its execution, but can have a significant impact in the long term. In the case study, there are no traditional episodes, making it a continuing task with an infinite horizon. In this scenario, standard Q-Learning tends to focus on short-term rewards, making it difficult to capture long-term effects and thus limiting the model's ability to accurately evaluate the true impact of an action. It is in this context that N-Step Q-Learning proves beneficial, as it does not solely rely on the immediate reward obtained upon taking an action, but rather harnesses a sequence of future rewards to train the model. The equation for updating the model in N-Step DQN is articulated as follows:

$$Q_{\text{new}}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[\sum_{k=1}^{N} \gamma^{k-1} r_{t+k} + \gamma^N \max_a Q(s_{t+N}, a) - Q(s_t, a_t)\right]$$

This longer-term vision enables a better understanding of the relationship between an action and its long-term consequences, thereby generating more accurate transition dynamics aiding in the model training. Furthermore, more stable estimates and a reduction in the variance of transitions are achieved. By averaging the results over multiple steps before making an update, N-step Q-Learning can reduce the variance in policy updates compared to the 1-step approach [43]. Specifically, N-step Q-Learning was used with N = 3. The resulting equation is:

$$Q_{\text{new}}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 \max_a Q(s_{t+3}, a)) - Q(s_t, a_t)\right]$$

4.1.2 Replay Memory

A prevalent technique within these algorithms is the deployment of a **Replay Buffer** or **Replay Memory**. A replay buffer maintains a historical record of past transitions (states, actions, rewards and subsequent states). During training, a random set of transitions from the buffer is sampled, and these transitions are used to update the model. The capacity of the replay buffer is finite, storing a limited number of transitions. As new transitions are inserted, older entries are removed to accommodate the most recent ones, thus maintaining a First In, First Out (FIFO) configuration.

One advantage of this technique comes from the reuse of *Past Experiences*. By preserving earlier transitions, the replay buffer enables the learning algorithm to reuse this information multiple times to update the policy. This repeated utilization of samples significantly increases learning efficiency by extracting more value from each interaction with the environment.

Another advantage is that it facilitates learning within dynamically evolving (non-stationary) environments. In scenarios where environmental conditions may evolve rapidly and frequently, the use of a replay buffer mitigates the risk of the learning system becoming outdated

by allowing previous experiences, which may become relevant again under new conditions, to continue influencing the agent's policy.

Additionally, using a replay buffer ensures that the training process is not heavily influenced by a sequence of atypical or infrequent experiences. This leads to more robust and stable learning by averaging the effects of a broader range of possible situations.

4.1.3 Input variables for the Model

The model requires knowledge of the current state of the system to make the appropriate scaling decisions. The selection of input variables was guided by a combination of domain knowledge, insights from the literature, and iterative experimentation to ensure their relevance and utility. With the aim of maximizing the information available to the model and thus maximizing performance, the following attributes have been selected:

- **CPU Utilization:** This field contains the CPU utilization by the Virtual Machine. It is essential to understand the system's state and whether it is operating at capacity or underutilized.
- **Percent Assigned CPU:** This represents the amount of CPU resources allocated to the VM. Only these resources are used to process requests, and the resource cost is derived from this value. A VM can be allocated up to the entirety of the host's resources. This parameter represents the fraction of the host's total resources currently allocated to the VM.
- **Response Time:** This represents the average response time observed over a given period. This metric serves as the principal indicator for assessing the quality of the user experience being provided.
- **Waiting Requests:** This field contains the count of requests that are queued for execution. It provides information about both the system's state and the user experience.
- **Ratio Requested Available CPU:** This final parameter is a ratio calculated between the system's processing capacity in its current configuration and the demand for processing in a given time period. Positive values indicate that there is a surplus of resources relative to the demand, whereas negative values indicate the opposite.

4.1.4 Scaling Actions

The possible actions to be performed by the agent are straightforward and directly map to scaling actions. They include the following actions:

- Increase resources
- Maintain current resources
- Decrease resources

When modifying resource levels, the adjustment represents a proportion of the resources currently allocated, around 10%. However, given that resource allocations are quantified as integer values, discrepancies may occur where the actual scaling adjustment does not precisely correspond to the predetermined percentage.

4.1.5 Illegal Actions

Illegal actions refer to decisions or movements that are not allowed, according to the rules or constraints of the environment. Within the context of this study, there is an environmental constraint, which refers to the inability to add more resources to a Virtual Machine when it has already been allocated all the resources of the host. Designating this situation as an

illegal action prevents the model from contemplating increments in resources under these conditions, thereby facilitating and accelerating the learning process. From an implementation perspective, considering an action illegal involves masking the neural network output of such action, ensuring it is not the most valuable choice and avoiding its selection.

4.1.6 Reward Function

A variety of reward functions were explored, and two of them were evaluated. *Reward Function A* is a more innovative approach, while *Reward Function B* is more traditional.

The classical reward function, *Reward Function B*, comprises two distinct penalties: one related to response time and another to resource allocation. For the former, an acceptable response time threshold is defined, within which it is considered that a good user experience is being provided. As long as the response time remains within this threshold, no penalty is applied; however, if the threshold is exceeded, the penalty increases as the response time deviates further from it. The second penalty corresponds to the amount of resources allocated. That is, the more resources are assigned, the higher the operational cost of these resources and the lower the reward will be.

Therefore, the reward function is a multi-objective function with two specific objectives: to minimize resources used and to maximize user experience.

$$Reward\ Function\ B = 3 * \frac{R_B\ Response\ Time}{R_B\ Resources}, \text{ with}$$

$$R_B\ Response\ Time = \begin{cases} 1 & \text{if } Resp.\ Time < Max\ Resp.\ Time, \\ e^{-\left(\frac{Resp.\ Time - Max\ Resp.\ Time}{2 * Max\ Resp.\ Time}\right)} & \text{otherwise} \end{cases}$$

$$R_B\ Resources = \frac{Percent\ Assigned\ Resources}{100}$$

The more innovative reward function that was studied, *Reward Function A*, was extracted from the reviewed literature [17], and it adopts a different approach. Like the first reward function, it assigns a score based on the response time. However, unlike the first, it does not penalize the usage of resources directly; rather, it includes a utilization score. This score involves defining a CPU utilization target, set at **80%**, which the controller should strive to get as closely as possible to minimize penalties. The purpose of this is to avoid low utilization levels, thereby preventing the waste of resources, while at the same time providing a sufficient margin to adapt to unexpected increases in demand.

$$Reward\ Function\ A = \frac{R_A\ Response\ Time}{R_A\ Utilization}, \text{ with}$$

$$R_A\ Response\ Time = \begin{cases} 1 & \text{if } Resp.\ Time < Max\ Resp.\ Time, \\ e^{-\left(\frac{Resp.\ Time - Max\ Resp.\ Time}{Max\ Resp.\ Time}\right)} & \text{otherwise} \end{cases}$$

$$R_A\ Utilization = |Target\ CPU\ Utilization - CPU\ Utilization| + 1$$

4.1.7 Neural Network Architecture

The structure of the neural network used is simple and compact.

It consists of a fully connected network with three hidden layers. The dimensions of these hidden layers are 16, 32, and 16 nodes, respectively. Between each layer, a ReLU activation function is used to augment the modeling of more complex relationships between the input and output variables and to accelerate convergence.

The input variables to the network include previously defined metrics, with particular focus on response time and the number of waiting requests in the queue, which are unbounded and thus must be normalized beforehand.

For the network's output, there are three neurons, each corresponding to a possible action within the system. The selected action will be the one associated with the neuron displaying the highest value, indicating the optimal decision according to the network's evaluation.

4.1.8 DDQN

An alternative to the conventional DQN, known as Double Deep Q Networks (DDQN), was explored. This model is designed to mitigate one of the fundamental issues that arise with traditional DQN: the tendency to overestimate action values [6]. In traditional DQN algorithms, the same network performs both the selection and evaluation of actions, which can lead to an overestimation of action values. Such overestimation occurs because the simultaneous maximization of action selection and evaluation tends to produce estimates that are biased toward higher values. To counteract this bias, DDQN addresses this problem by employing two distinct networks: one network is dedicated to selecting the best action (the policy network), while the other is tasked with assessing the value of the selected action (the target network). By decoupling these two processes, the risk of overestimations is significantly reduced.

Ultimately, it was expected that using DDQN would yield better results than DQN. Unfortunately, this was not observed in practice, and therefore, DDQN was discarded for the final version.

4.1.9 Exploration vs Exploitation

A crucial aspect of reinforcement learning is managing the balance between exploration and exploitation. Exploration involves selecting actions that may not be optimal in the short term, with the goal of discovering new states or strategies that could lead to better rewards. Exploitation, on the other hand, leverages the agent's current knowledge to select actions that maximize reward based on its learning so far.

Striking this balance is essential; excessive exploration may degrade performance due to suboptimal actions, while insufficient exploration or excessive exploitation may prevent the agent from discovering better strategies.

In the case study, the ϵ-greedy strategy was employed, where the agent explores randomly with probability ϵ and exploits the best-known actions otherwise. The value of ϵ is gradually reduced during training, starting with high exploration to encourage discovery and shifting towards more exploitation as the model becomes more accurate. This controlled reduction ensures that the model benefits from early broad exploration while focusing on optimal actions later in training.

$$\epsilon(\text{episode}) = \epsilon_{\min} + (\epsilon_{\max} - \epsilon_{\min}) \cdot e^{-\alpha \cdot \text{episode}} \text{ , where:}$$

$$\begin{aligned} \epsilon(\text{episode}) &: \text{Exploration rate at a given episode} \\ \epsilon_{\min} &: \text{Minimum exploration rate} \\ \epsilon_{\max} &: \text{Maximum exploration rate} \\ \alpha &: \text{Exploration decay rate} \\ \text{episode} &: \text{Current episode number} \end{aligned}$$

4.1.10 Training

The model's training is a continuous process. At each predefined Scaling Interval, the system requests a new scaling action to the RL controller. The resultant system state, along with the initial state, the executed action, and the associated reward, formulates a transition that is subsequently transmitted to the model for storage in the Replay Buffer. In the case of N-Step Q-Learning, this transition encompasses actions and states extending N-steps into the future.

Once a sufficient number of new transitions accumulate in the Replay Buffer, the model training is initiated. This phase involves the extraction of data batches, which are randomly selected subsets of transitions from the Replay Buffer. These batches are utilized to refine the estimated Q-values of potential actions stored implicitly within the neural network through backpropagation. This process involves calculating the average loss per batch, which is the discrepancy between the model's initial predictions and the observed outcomes. This calculated loss is then used to update the neural network's weights accordingly and generate more accurate predictions.

4.1.11 Last Layer Retraining

Literature review reveals that it is common to only retrain the last layer of the neural network following the transfer. This approach is justified because the more general knowledge is already well encapsulated within the model, and only minor adjustments are necessary to fine-tune the model to accommodate the discrepancies between the source and target domains. Therefore, in addition to standard training, there is a variant in which only the last layer of the neural network is trained, potentially leading to a more expedited training process.

4.2 Transfer Learning

This section provides a detailed description of the two transfer learning methodologies selected for pre-training the model prior to deployment in the real environment. Sim-to-Real and Learning from Demonstrations were identified as the methodologies best suited to the requirements of this study.

4.2.1 Sim-to-Real

The objective of Sim-to-Real is to train the RL controller using a simulator designed to closely mimic the real system's environment. By pre-training the model in this simulator, it develops a policy that ideally resembles the one required in the actual environment, eliminating the poor initial performance phase typically associated with reinforcement learning.

The process involves conducting the standard RL training procedure previously described, but with the RL controller interacting with a simulator rather than the actual environment. Since the process is done in a simulated environment, it can be performed for as long as it is necessary to properly train the model.

Once this pre-training phase is complete, the model is ready to be deployed directly in the real environment.

Naturally, it is crucial for the simulator to accurately represent the dynamics of the real environment, as significant discrepancies between the two can lead to a model that fails to adapt effectively when transferred to the actual system.

4.2.2 Learn from Demonstrations

Learning from demonstrations involves using demonstrations sourced from either an expert or a pre-existing controller to pre-train the model. This approach embeds the model with prior knowledge derived from the demonstrations and helps establish a foundational policy.

The training methodology employed to perform Learn from Demonstrations is based on the approach outlined in [13].

Firstly, it is necessary to obtain the demonstrations, which were extracted from the logs of a threshold-based controller. These demonstrations should have the same format as the transition used for RL training. Subsequently, those demonstrations are loaded into the replay buffer.

Although there are similarities with the conventional reinforcement learning training process, the method for updating the model's weights differs, using a supervised learning approach. Firstly, batches of demonstrations or transitions are selected from the replay buffer, which remains unchanged. However, the process diverges during the calculation of the loss function. Instead of solely relying on the N-Step Q-Learning loss, two additional loss metrics are incorporated:

- **Supervised large margin Classification Loss**
- **L2 Regularization Loss**

The supervised large margin classification loss, a supervised loss function, is intended to encourage the model to replicate the actions observed in the demonstrations. It achieves this by creating a positive margin between the Q-value of the demonstrated action and those of the other possible actions. Essentially, it assigns a higher reward value to the action seen in the demonstration, while assigning lower values to the other actions. This adjustment is crucial, as the demonstrations provided exclusively contain optimal actions, without any reward information from suboptimal actions. Without addressing this limitation, there is a risk that the Q-values for less optimal actions could exceed those for the optimal actions, since they are never set, which would compromise the model's performance after transfer.

The second loss function, the L2 regularization loss, is intended to prevent the model from overfitting to the relatively small number of demonstrations used during training. The total loss is compounded by the three previously mentioned losses and is used to update the neural network weights.

Another distinction is that this training process is repeated until the loss falls below a predefined target. Once this target is achieved, the training concludes. This measure reinforces the efforts to prevent the network from overfitting, which would diminish its performance once tested in real conditions.

4.3 Definitions

In this section, definitions of core concepts are presented.

4.3.1 Requests or Cloudlets

In this study, a request represents an API call submitted to the system, and it is the basic processing unit. Each request inherently requires a certain amount of processing time, memory, disk usage, and arrives at a specific point in time.

When simulating these requests in CloudSim Plus, they are modeled as cloudlets.

Cloudlets are designed to capture and simulate the characteristics of requests. They include parameters such as the amount of processing required (measured as millions of instructions to be executed), memory usage during execution, disk space usage, and arrival time.

4.3.2 SLA

Service level agreements (SLAs) are predefined agreements between the cloud provider and the client who will use the cloud platform to host their application. Of particular importance for this case study is the response time, defined as a certain percentage of API calls not exceeding a specific time threshold. For example, 99% of requests will not exceed 200ms response time. There are additional SLAs covering aspects such as availability and fault tolerance; however, these are not relevant for the analysis of resource management controllers within the context of this study.

4.3.3 Workloads

A workload consists of a set of requests that need to be processed, each associated with an arrival timestamp. Each workload spans a total duration of 24 hours, but the number of requests varies between workloads. The difference between various workloads lies in the number of requests they contain and when these requests arrive. All workloads have stages with a higher number of requests, representing higher demand, and stages with fewer requests, representing lower demand. The rate of increase or decrease in the arrival rate of requests is progressive, and the controller must be capable of adjusting the resource allocation at the same pace as the change in cloudlet arrival rates.

There are two distinct types of workloads: synthetic and real. Synthetic workloads are artificially generated to mimic expected real-life behavior. Real workloads are generated from actual data from an e-commerce site, using logs from Kaggle, an online platform that hosts public datasets for data science and machine learning. From these logs, which cover a total duration of almost five days, five workloads were generated, one for each day.

4.3.4 Episodes

In traditional reinforcement learning, each episode has a specific goal. At the end of the episode, a reward is received according to the performance of the agent, and this reward is subsequently used for training. In this case study, the scenario is more complex since there are no traditional episodes. This arises because of the continuous nature of the task at hand, rather than finite task, which introduces several challenges.

Firstly, the reward system differs in that rewards are not aggregated post-episode but are instead awarded after each action is performed. These immediate rewards focus on the immediate effects of actions rather than their long-term impacts, necessitating an adaptation of traditional training methods to better suit continuous tasks.

Despite this, in order to facilitate training and to better evaluate performance, an artificial variant of episodes is used. In these episodes, what is essentially a continuous process is partitioned into several segments. Each episode covers a specific time period limited to 24 hours, during which a specific workload is executed, containing the arrival of requests within that 24-hour period.

During the controller's training, these segments are treated as distinct episodes to simplify the learning process. However, from the controller's perspective, it experiences a seamless, ongoing sequence of actions rather than discrete episodes. This approach allows for the systematic training of the controller under conditions that mimic continuous operation while still leveraging the structured framework of episodes for ease of analysis and performance evaluation.

4.4 Design Decisions Taken

This section discusses several decisions and choices made during the design of the solution.

4.4.1 Type of Scaling

The selected approach for scaling was vertical scaling, where the resources are adjusted within a single virtual machine rather than increasing the number of virtual machines.

This decision was supported by two factors. Firstly, it was observed that studies and implementations typically favor vertical scaling configurations, leading to the decision to adhere to this common practice for consistency and comparability in our research. Furthermore, the simulation tool utilized offers a more robust implementation of vertical scaling compared to its horizontal counterpart, which remains underdeveloped and would require substantial additional development to achieve a functioning implementation.

Should horizontal scaling be added to the current solution, a global controller would be needed. Although this global controller could conceivably be based on deep reinforcement learning, current literature suggests that simpler, frequently rule-based controllers are more commonly employed for such applications.

4.4.2 Resources to Scale

In the implemented solution, a strategic decision was made to monitor and control a single system resource. Specifically, the number of central processing unit (CPU) cores was chosen, which is frequently identified as a primary bottleneck in application performance. To eliminate the impact of other potential system bottlenecks, such as RAM, sufficient resources were provisioned to eliminate their interference with system performance.

The rationale for concentrating on a single resource was driven by several reasons: reducing the complexity of the implementation, facilitating smoother integration, simplifying the training process, and extracting more precise insights into system behavior and performance metrics. Future enhancements could include extending the model to concurrently manage multiple resources, thus broadening the scope of the system's adaptability and efficiency.

4.4.3 Evaluation Metrics and Performance Targets

To effectively evaluate the system's performance, it is necessary to establish objective metrics that accurately measure system performance. Service Level Agreements (SLAs) commonly stipulate performance guarantees, such as processing 99% of requests within a specified timeframe. Correspondingly, clear thresholds for acceptable performance must be delineated on the system's side, which are integral to configuring the reward function that guides the controller's operations.

Such values are defined in Section 4.6.2.

4.4.4 Differences between Simulation and Simulated Reality

The use case for which the solution was designed when utilizing Sim-to-Real, involves an initial phase where the controller is trained in a simulator, transferred to the real environment, and finally retrained in reality. For several reasons, it was decided not to conduct the second training phase in a real environment, but instead in a modified version of the simulator acting as the real environment, called **Simulated Reality**. This decision was primarily motivated by the substantial time and effort required to implement such an approach. First, extensive research would be necessary to determine how to implement a solution in the real environment, including tools and libraries. Second, significant modifications to the controller would be required to ensure its functionality and compatibility with real-world systems. Finally, acquiring the appropriate infrastructure for testing would pose an additional challenge.

This modified version of the simulator is designed to approximate, to some extent, the potential discrepancies the controller might encounter between simulated and real environ-

ments. It ensures that the transfer is not flawless, mimicking the challenges of moving to a real-world setting.

The differences between running in simulation and simulated reality are as follows:

- **Workload:** During the training phase in the simulation, synthetic workloads are employed, which are synthetically generated based on specific heuristics. In contrast, the workloads deployed in the simulated reality environment derive from real logs obtained from an e-commerce website. These workloads exhibit different behaviors, the real workloads display much greater fluctuations within brief intervals, with specific peaks and drops, while synthetic logs are smoother.
- **UpScaling Delay:** A delay in resource scaling is implemented. When the reinforcement learning controller controller determines that additional resources are needed, these new resources are not immediately available. Rather, there is a notable delay before their deployment, as would be the case in a real system. This delay was set to **1000** seconds, a substantial period.
- **Monitoring Delay:** The controller must take actions based on the current state of the system. However, in practice, it is common that the system's state is monitored at intervals. Therefore, when a decision regarding scaling is needed, the exact state of the system at that specific moment may not be available. Instead, the decision may have to rely on data that reflects the system's state at a previous time point.

While these adjustments are applicable within the simulator, there exist additional variations that cannot be implemented. The purpose of these artificial differences is to create distinctions between the two environments, the simulator and the simulated reality, similarly to the disparities between a simulator and an actual production system.

4.5 CloudSim Plus and Extensions

As previously mentioned, CloudSim Plus was chosen as the simulation tool. This section provides a brief overview of the simulation process and describes several extensions made to CloudSim Plus, which introduce necessary functionalities.

4.5.1 Simulation Process

To initiate a simulation, it is necessary to establish, at the very least, a data center architecture along with a corresponding workload for execution. The architecture of the data center comprises a sequence of hosts (a minimum of one is required), each equipped with specific resources. Subsequently, each host will operate at least one virtual machine, to which designated resources from the host will be allocated.

Regarding the workload, it is composed of a series of cloudlets. These cloudlets represent the fundamental processing units, requiring specified amounts of resources and computational effort. Furthermore, each cloudlet is assigned an arrival time, which is delineated in units of simulation time and indicates the moment the cloudlet arrives to be executed.

Once the architecture and the workload are set, the simulation can be run, and the results can be analyzed.

4.5.2 Added Functionalities

Although many of the essential functionalities are already implemented, it became apparent that additional features were required. The implementation of these features was facilitated by the inherent design of the tool, which is structured to be extensible. This design allows

the integration of new functionalities without necessitating significant modifications to the existing codebase.

The subsequent extensions were implemented.

Loading a workload from a file

The ability to load the workload to be simulated from a file was introduced, providing a straightforward method to select which workload to execute by simply changing a parameter.

Logging

The ability to log performance metrics, including utilization and resource deployment throughout the simulation, was incorporated. These metrics are instrumental for evaluating the performance of the controller and are stored in files upon the completion of the simulation.

Capability to Connect to an External API

During the simulation, it is imperative to communicate with an external controller for a variety of functions, including interfacing with the RL controller and further logging of information.

Control of Scaling Actions by an External Controller

By default, CloudSim Plus permits scaling actions to be conducted exclusively by an internal controller based on predefined thresholds. To align with the requirements, it was essential to upgrade the tool to enable the execution of scaling decisions made by an external controller.

Decoupling the Scaling Interval from the Scheduling Interval

Initially, the simulator's design links the scaling of resources and the scheduling of cloudlets, maintaining a uniform interval between successive actions. This arrangement is impractical, as the interval for scheduling (deciding the execution resource for a cloudlet) is typically very brief (on the order of milliseconds), whereas the interval for scaling actions is considerably longer, often spanning several minutes. Aiming for a more realistic approach that is reflective of actual operational scenarios, it was decided to decouple these two intervals.

Enhanced Realism in CPU Scheduling (Resource Sharing Capability)

CloudSim Plus offers the functionality to specify the resource utilization for a cloudlet, such as allocating 50% of a CPU core. This feature is intended to mimic scenarios where a cloudlet does not continuously utilize the CPU, possibly awaiting I/O tasks. Under typical circumstances, it would be possible for another task in the queue to utilize the resource during this idle period. However, such functionality was not available. To address this, a new scheduler was introduced, allowing two cloudlets, whose combined resource utilization does not exceed 100%, to operate concurrently in the same core. Although this feature boosts realism, it was ultimately not used due to its significant negative impact on performance.

Cloudlet Timeout and Cloudlet Dropout

When demand exceeds processing capacity, cloudlets start to accumulate in the waiting queue. This situation can lead to two outcomes. On one hand, it may trigger a user-side timeout, where the user ceases to wait for a response. On the other hand, the server's waiting buffer may become full, which will cause new cloudlets to be rejected. These two

functionalities were incorporated into the simulator to make it more reflective of real-world scenarios.

Dynamic Cloudlet Submission

While the simulator offers the possibility for a delay in arrival time, all cloudlets must be created and submitted at the start of the simulation. When handling large workloads, consisting of millions of cloudlets, this can become a bottleneck for performance. The capability for a cloudlet to be submitted only at the appropriate time was added, minimizing the number of cloudlets the simulator needs to manage at any given time.

More Accurate Resource Utilization Metrics

The resource utilization metrics reported by CloudSim Plus are instantaneous, therefore reflecting only the conditions at any given moment. Due to the dynamic nature of demand, there may be specific peaks in demand, and the reported metrics may not provide an accurate representation of ongoing conditions. Additionally, there have been observations where metrics were collected at the exact moment one or more cloudlets were initiating or finishing execution, resulting in unreliable utilization data in those instances.

To address this, a new utilization metric was implemented based on two components: the available processing capacity over a period of time, and the processing performed during that same period. This approach provides an average utilization value over a specified interval that more accurately represents the system's state.

Other System State Metrics

To assist the controller in making the most informed decisions, several additional metrics were introduced. Among these metrics is a ratio that compares the system's processing capacity over a specified period with the processing demands required to execute the cloudlets that arrived during that same period. This metric provides a clear indication of the relationship between demand and the system's processing capacity, offering further insights for effective system management.

Downscaling when all Resources are Utilized

When a downscaling action is instructed, the simulator validates whether there are free resources that can be released. If there are no free resources, then the scaling action is rejected. While this approach appears logical, it can lead to unexpected outcomes from the controller's perspective, in which the chosen action has no effect. On the other hand, rejection of the scaling action prevents the controller from properly exploring the action space, potentially hindering its learning process. To address this issue, the capability to release resources, even when they are momentarily in use, was implemented.

Monitoring Delay and Scaling Delay

In order to more accurately replicate real-world conditions within the simulator, two different types of delays were implemented. The first one, named monitoring delay, is designed to reflect the fact that the system state accessible to the controller might not be up to date. The second type of delay, the scaling delay, refers to the duration required for the system to update its resources following a scaling action, specifically when additional resources need to be incorporated. During this period, the system enters a deadlock state, during which no further actions are requested until the completion of the pending upscaling action. This process commonly experiences a delay in real systems, and was considered crucial to simulate.

Action Effect Delay and State history

When a scaling action is performed, its effects are not immediate but can be observed over the long term. Therefore, it is not useful for the RL controller to retrieve the system state immediately after the action is taken, as the effects of the action are not yet apparent. Thus, a specified time interval has been defined to wait before retrieving the resulting state.

Additionally, a historical record of the system states was implemented, which is employed for both the monitoring delay and the particular variant of Deep Reinforcement Learning used.

4.5.3 Synchronization between Simulator and Controller

In a real-world environment, the controller would constantly monitor the system's state and execute the corresponding actions when it deems it necessary. However, to synchronize the simulator with the controller, an inverse approach was adopted, where the simulator is responsible for requesting scaling actions. This arrangement is due to the simulator's inability to receive external connections, rendering it impractical for the controller to coordinate both systems. Therefore, it was decided that the simulator, at designated time intervals, would communicate with the controller by sending the state of the system, and subsequently, the controller would provide a scaling action to be executed.

Although this method of coordination does not replicate the real-world process, practically, there is no difference in results between the two methods.

4.6 Implementation Nuances

4.6.1 Simulation Times and Cloudlet Execution

As previously mentioned, a workload comprises a set of cloudlets whose arrival is distributed over a 24-hour simulation period. In order to execute a workload, the simulator processes these cloudlets by simulating the execution of tasks and allocating the necessary resources for each cloudlet according to its specific characteristics. This implies that the more cloudlets that need to be executed, the longer it takes to complete the simulation. This is particularly evident when dealing with simulations involving millions of cloudlets, as would be the case in real-world applications, where executing the simulation becomes exceedingly time-consuming. To address this, a decision was made to create a smaller number of cloudlets but with an increased processing time (beyond what would be reasonable in the scenario of the case study), significantly reducing the time required to execute a simulation.

One approach to understanding this is that each cloudlet represents or consolidates 1000 requests or tasks. However, implementing this aggregation has certain implications.

Firstly, both the execution and response times appear "inflated" compared to the scaling times. For example, the minimum processing time of a cloudlet (representing 1000 requests) is 20 seconds, while a scaling decision is made every 200 seconds, creating the impression that scaling decisions are made very frequently. However, this is not the case, as the variation in the arrival rate of cloudlets over time remains consistent compared to the non-consolidated approach. In other words, whether this aggregation is used or not, the percentage of variation in the number of cloudlets received per unit of time throughout the simulation remains consistent, thus justifying the maintenance of the same scaling interval.

Secondly, while the execution time of one cloudlet is equivalent to the execution time of 1000 requests, this does not hold true for response times. This discrepancy arises because the processing time for a cloudlet is significantly longer than that for a single request, leading to substantially longer waiting times if resources are occupied.

Consequently, the controller's behavior may not be exactly the same compared to scenarios without aggregation, although the differences observed in preliminary tests were minor. Despite these nuances, this behavior remains consistent across all case studies and algorithms tested, thereby ensuring that the results are comparable between them.

4.6.2 Selected Configuration Parameters

To perform the simulations, several key parameters were established, all defined in terms of "simulation time". As noted earlier, each simulation spans 24 hours.

Two types of cloudlets with distinct processing requirements are simulated. The more common "GET" cloudlet requires 20 seconds of processing time, while the far less frequent "POST" cloudlet necessitates 200 seconds.

The scaling interval was set to 200 seconds, indicating the time between consecutive scaling actions, provided no scaling delay is enabled.

When the scaling delay is activated, it introduces a 1000 seconds delay in provisioning additional resources, a significant duration during which no further actions are taken by the controller.

Additionally, the monitoring delay results in 50 seconds of lag in the system's state information, hindering the controller's ability to make timely decisions based on the current conditions.

Finally, the acceptable response time before incurring penalties was set to 200 seconds. This value matches the minimum processing time required for a "POST" cloudlet, meaning that if there are any previous cloudlets in the waiting queue, then the cloudlet will inevitably exceed the predefined threshold.

4.7 Code

The code for the Deep Reinforcement Learning controller can be found in `https://gitlab.fing.edu.uy/santiago.serantes/drl-controller`, and for the Cloud Sim Plus Simulator, along with its extensions, the code can be found in `https://gitlab.fing.edu.uy/santiago.serantes/tesis-cloudsimplus-v2`

Chapter 5

Experiments and Evaluation

This chapter explains the methodology of the experiments conducted and outlines the various experiments designed to comprehensively analyze the controller's performance and the effectiveness of transfer learning. The goal is to demonstrate their applicability and significance in addressing the challenges of cloud resource management. Furthermore, the chapter discusses the performance metrics used to evaluate these results and provides insights into their interpretation.

Additionally, a detailed study of the distinct hyperparameters is performed to understand their impact on the controller's behavior and overall performance.

5.1 Experimentation and Evaluation Framework

The project has two main objectives. The first is the development of a reinforcement learning algorithm that surpasses traditional algorithms in managing resources within a data center. The second objective is to study the application of transfer learning techniques to ensure that the performance of this reinforcement learning (RL) controller is optimal from the outset.

The context in which the algorithm was designed involves replacing traditional resource scaling policies or algorithms. In this scenario, it is crucial for the controller to demonstrate satisfactory performance initially, as sub-optimal initial performance could compromise the proper functioning of client applications, resulting in violations of the Service Level Agreements. Consequently, this would incur losses for both the cloud service client and the provider.

To evaluate these objectives within the defined context, a series of experiments were designed and structured into distinct stages. These stages are designed to assess the performance of the reinforcement learning algorithm and its ability to leverage transfer learning techniques.

5.1.1 Training and Testing Stages

The training and testing process follows a series of distinct stages. The first is the Transfer Learning Pre-training stage, where the model undergoes pre-training using transfer learning techniques to establish a foundational model. This stage is further divided into two distinct parts, each dedicated to a specific transfer learning technique. Upon completion of this first stage, the second stage begins. The pre-trained model is deployed in the simulated reality environment and its initial performance is evaluated. The following stage is the Training in the Simulated Reality Environment phase, and consists of refining the pre-trained model. To do this, the model is trained using the RL training process where the model interacts

with the environment and adapts to the new conditions. In the last stage, the final model is evaluated to assess its performance.

Transfer Learning Pre-training

To ensure that the Reinforcement Learning model demonstrates robust performance immediately upon deployment, a preliminary training phase is essential. The purpose of this stage is to embed the model with foundational knowledge, crucial for its initial effectiveness. To achieve this, two specific transfer learning techniques are employed: Sim-to-Real and Learning from Demonstrations. These techniques are integral in preparing the reinforcement learning controller for real-world applications right from the start.

Sim-to-Real Transfer

This approach involves pre-training the model within a simulator, with the goal of preparing the model for deployment.

The greatest difficulty of Sim-to-Real lies within the simulation process. The simulation must be sufficiently realistic such that the policy learned is applicable to the real-world scenario. If this is not achieved, the knowledge intended for transfer will not be applicable, leading to poor outcomes. While it is impossible for a simulation to capture every nuance and aspect of reality, it is crucial that the simulation training closely resembles real-world conditions to maximize the effectiveness of the transferred knowledge. However, developing such a realistic simulator can be an exceedingly costly and time-consuming endeavor, making it a significant factor to consider when evaluating Sim-to-Real transfer learning.

In the case study, this issue does not pose a problem since the simulator is utilized in both scenarios. However, this situation does not accurately represent situations where transfer to a real system is performed. To address this and approximate the conditions of transferring to a real system as closely as possible, artificial modifications are introduced to the simulator to generate a difference between the two environments. These modifications include the addition of a Monitoring Delay and a Scaling Delay, creating a simulated reality environment as previously discussed in Section 4.4.4. While these delays can be found in real environments, they are not necessarily present in simulators such as CloudSim Plus.

The training in the simulator is conducted by iterating over a set of synthetic workloads, each one with different characteristics, which differ from the workloads based on real data used in the simulated reality environment. One such sample can be observed in Figure 5.1.

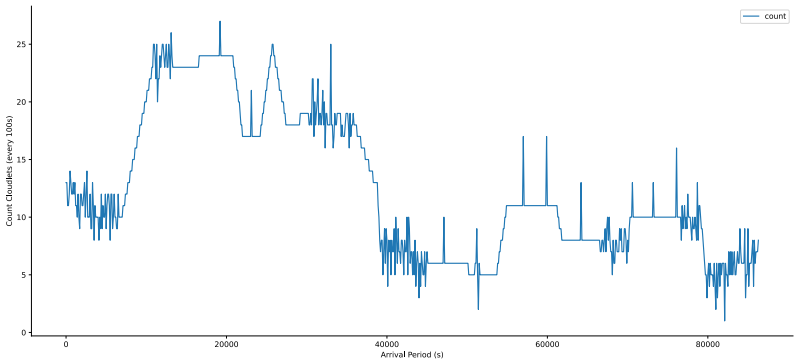

Figure 5.1: Synthetic Workload Sample

Since the training is based on a simulation, it affords the opportunity to conduct as many simulations as desired to achieve a satisfactory level of performance. However, it is crucial that the learned knowledge does not overfit the simulator's behavior as this could lead to suboptimal knowledge transfer due to discrepancies between the domains. Nonetheless, no such behavior was observed, even after prolonged training. Consequently, the training was conducted over an extended simulation period of 200 episodes, equating to 200 days of simulated time.

Performance is continuously monitored throughout this process to track the progression of learning and its convergence.

Once the transfer is performed and the model is deployed in the real environment, retraining becomes essential for the model to adapt to the new environment.

Learn from demonstrations

In this approach, the model learns from demonstrations obtained from another policy. Therefore, an imperative step in implementing learning from demonstrations lies in the existence of a source for these demonstrations. This source may originate from either an expert or a pre-existing controller. In the scenario involving the pre-existing controller, the demonstrations are not required to be optimal; nonetheless, the greater their quality, the better the resulting performance will be.

The decision was made to extract demonstrations from an existing controller, which operates based on thresholds. To acquire these demonstrations, a simulation based on the simulated reality environment is executed, utilizing a conventional threshold-based algorithm as the controlling mechanism. Throughout the simulation process, the actions, along with the initial and subsequent states, are recorded. The inclusion of demonstrations from a threshold-based controller is based on the assumption that they provide a viable basis for training an RL controller, despite potential limitations in the optimality of their actions.

Three simulation runs were conducted in the simulated reality environment using the predefined controller. During its execution, approximately 600 demonstrations were collected that were used to train the model. Each demonstration is comprised of an initial state, an action, and N-Step future states. Additionally, a reward is calculated using the same function that the agent will employ, and training is conducted based on these demonstrations.

This methodology aims to illustrate a scenario in which a rudimentary cloud elasticity controller, based on threshold values, is already operational within a real system. The objective here is to replace this controller with a more powerful alternative. In such instances, it is easy to acquire demonstrations that are suitable for the pre-training of a reinforcement learning controller.

The main advantage of this technique is that it provides demonstrations from the real system's behavior, thereby eliminating the discrepancies between the two environments found in Sim-to-Real. However, the disadvantage of this method is its inability to precisely replicate the behavior of the original policy, which itself is not an optimal policy.

During the learning process, training metrics are continuously monitored, as they provide critical information for evaluating the training progress.

Once the model has been pre-trained, it can be transferred into the real environment.

Following the transfer, it is imperative to retrain the model to further optimize the model and its performance, ideally surpassing that of the original controller. During the fine-tuning phase, new transitions will gradually replace the original demonstrations in the replay buffer. This represents a deviation from the approach of utilizing an expert, where it is preferable to maintain the original demonstrations in the replay buffer indefinitely to continue influencing the model. This is based on the premise that the expert's demonstrations are as optimal as possible, and thus, it is crucial that this knowledge is not lost during the retraining process. However, in this case study, the demonstrations do not come from an expert and do not provide long-term value, making it unnecessary to retain them in the replay buffer during

the retraining process.

Transfer Learning Performance Evaluation

Subsequent to the pre-training of the model using transfer techniques, this next stage focuses on evaluating its performance and effectiveness within the simulated reality environment. This evaluation involves the utilization of a workload generated from authentic data, in addition to the incorporation of previously discussed scaling and monitoring delays. The performance metrics obtained from this evaluation are indicative of the transfer's efficacy and determine the feasibility of deploying a DRL algorithm, especially given that the sub-optimal initial performance is a critical challenge for reinforcement learning algorithms in contexts where initial inefficiency is unacceptable.

Training in Simulated Reality Environment

During this stage, the focus is on retraining or fine-tuning the model in the simulated reality environment to adapt it to the unique characteristics of this setting and maximize its performance. This retraining phase follows the standard RL training process, much like the Sim-to-Real pre-training phase. The simulation operates under real-world conditions, where the model's ongoing training aims to reconcile the discrepancies between knowledge acquired in the pre-training phase and what it experiences in the new environment. Throughout this phase, the model's performance is continually monitored to evaluate its learning speed and adaptability to environmental changes.

This stage comprises 100 episodes, corresponding to 100 days. While such an extended period is not strictly required for this phase, it was selected to thoroughly assess the model's performance progression and asymptotic behavior.

The workloads employed during this retraining phase are derived from genuine operational loads, exhibiting distinct characteristics from those synthetically generated during the initial training phase, typical of Simulation-to-Real transitions. In the Figure 5.2, a far greater level of variation in the arrival rate of cloudlets can be observed compared to the synthetic workloads.

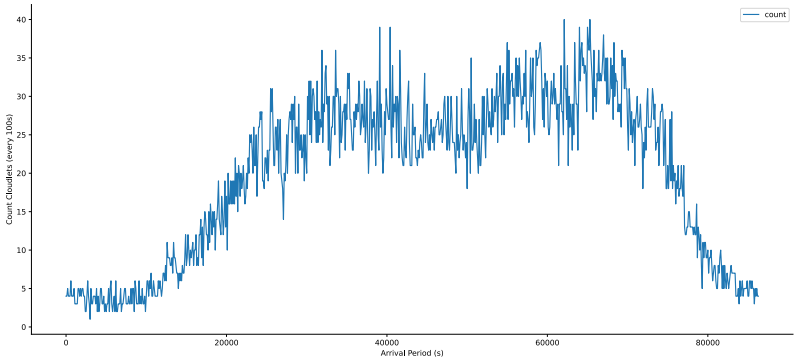

Figure 5.2: Real Workload Sample

For Sim-to-Real transfer, after the transfer has been completed, it is not necessary to reset the exploration rate for two reasons. Firstly, resetting the exploration rate increases the likelihood of the system entering undesirable states as a higher number of random,

and therefore suboptimal actions are taken. Furthermore, the model should already have a well-formed notion of the potential value of different actions, obviating the need for a comprehensive re-exploration of the action space.

Conversely, in the case of learning from demonstrations, a higher initial exploration rate could prove beneficial. Firstly, it inherently starts at a high value due to the relatively brief pre-training phase, which prevents a significant reduction in the exploration rate that occurs with prolonged training. Additionally, this increased exploration enables the reinforcement learning controller to more rapidly explore the outcomes of suboptimal actions, from which no information was provided in the demonstrations. This strategy accelerates the training process by enhancing the understanding of the action space and the expected rewards of less effective actions.

Final Performance Evaluation

The final stage is dedicated to evaluating the model's performance after completing the main training phase in the simulated reality environment. This evaluation facilitates the assessment of the RL controller's final performance and enables a comparative analysis with alternative solutions.

5.2 Performance Evaluation Design

5.2.1 Performance Evaluation During Training

Measuring performance throughout the training process presents significant challenges for various reasons. On the one hand, each training episode utilizes distinct workloads, rendering the results obtained across different episodes incomparable. This is attributed to the varying resource requirements necessary to meet demand, leading to greater costs, and the fluctuations in demand which can augment the complexity of resource management from one workload relative to another. Furthermore, during the early stages of training, high exploration rates may lead to sub-optimal initial outcomes, which gradually improve as the exploration rate decreases.

To address these issues, an alternative approach is used, which consists of periodically pausing the training to conduct a test run on a predefined testing workload, leveraging the knowledge acquired thus far. Evaluating overall performance during these testing runs provides an assessment of the training progress. The metrics employed here are the same as those utilized for the final performance evaluation, which are detailed in a subsequent section.

5.2.2 Performance Evaluation of Transfer Learning

The performance of the model immediately post-transfer is crucial to validate the model's feasibility for real-world deployment. Following the transfer, the model is introduced to a new environment featuring new transition dynamics and workloads, where the previously acquired knowledge may not entirely align with the new domain. Nevertheless, it is imperative that the model achieves a satisfactory performance level in this new context to prevent potential usability issues for end-users.

To accurately measure the effectiveness of transfer learning, several metrics are employed:

- **Jumpstart Performance:** Measures the initial performance immediately following the transfer.
- **Time to Threshold:** The number of episodes required to reach a pre-established performance threshold.

- **Accumulated Rewards:** The sum of rewards accumulated during the initial episodes post-transfer.
- **Performance with Fixed Predefined Epochs (10):** Evaluates performance after a set number of episodes.
- **Asymptotic Performance:** This is the best performance of the agent during and after re-training.

5.2.3 Performance Evaluation of the RL Controller

To assess the controller's final performance, multiple metrics are defined to measure various aspects of its performance.

The initial set of metrics focuses on response time, which serves as a critical indicator of the end-user experience. These include the **average response time** and **percentage of requests exceeding the specified time threshold**.

While average response time itself is not part of the Service Level Agreements, it is included to provide a better picture of the controller's performance. On the other hand, the percentage of requests exceeding the threshold is specified in the SLAs, where exceeding this threshold incurs a penalty.

In addition to response time, the cost associated with resource utilization is evaluated. This parameter is quantified by the **average number of CPU cores** allocated to the virtual machine, with costs assumed to be proportional to the volume of resources employed. The number of cores assigned to tasks represents the allocated resources, naturally incurring a monetary cost.

Furthermore, a composite metric, called ***Score***, integrates both the penalties for exceeding the response time thresholds and the cost of resources utilized. This metric provides a comprehensive view of the model's performance compared to analyzing the individual metrics in isolation. The objective is to minimize both the resources allocated and the number of requests exceeding the threshold, which are inherently in competition with each other. The *Score* is generated at the end of the simulation and is distinct from the rewards obtained from the reinforcement learning agent's reward functions. It is calculated as follows:

$$Score = 1 + \frac{10 - \text{Percentage Response Time Over 200} - \text{Percentage Assigned PEs}}{100}$$

This composite score differs from the values of the reward functions as the latter penalizes the margin by which response time thresholds are exceeded, while the former penalizes the percentage of cloudlets exceeding these thresholds. This approach is adopted for two main reasons. Firstly, it aligns with practices documented in the reviewed literature. Secondly, using the percentage of cloudlets that exceed the threshold as input for the model would be impractical; it tends to act as a binary variable because the response times of contiguously processed cloudlets are very closely related, so if one cloudlet exceeds the threshold then the subsequent cloudlets are very likely to do the same. Therefore, in the reward functions, the margin by which thresholds are exceeded provides a clearer understanding of the system's current state, assisting the controller in making more informed decisions.

One final metric to analyze is the **number of upscale and downscale actions** taken during the simulation. Although this is not a direct measure of the controller's performance, unlike the metrics previously mentioned, it does highlight the significance and relevance of the actions executed. A large number of actions taken without the associated performance improvement suggests that these actions may be unnecessary and would lead to increased operational costs and potentially system instability. Additionally, this behavior may indicate

an unstable controller that loops between states without converging to a stable state.

Overall, five distinct metrics are established to comprehensively evaluate the controller's effectiveness in managing cloud elasticity.

- **Average response time**
- **Percentage of requests exceeding a specified time threshold**
- **Allocated Resources**
- **Score**
- **Number of Scaling Actions performed**

5.2.4 Considerations

Run to Run Variability

The training of machine learning algorithms is typically non-deterministic. Several aspects of the training can lead the model to converge to different outcomes. The case study is no different.

This can be partially attributed to the exploration conducted by the reinforcement learning agent, where, during training, a random action is selected with a certain probability. This random action has long-term effects, and due to the continuous nature of the state variables and the large number of actions taken during training, it results in the agent's trajectories being different on each occasion.

Additionally, during training, random batches from the replay buffer are taken, meaning that the samples used for adjusting the weights of the neural network also have a component of randomness.

Furthermore, there is the non-deterministic component of the environment to take into consideration, where the workload demand can change in unpredictable ways, leading to different outcomes for the same action in the same state.

Another aspect contributing to this variability is that it is a multi-objective problem, and similar final results can be achieved despite using different policies. For instance, one policy might be more aggressive in resource allocation than another, but at the expense of worse response times.

This results in no two training runs being identical, and the trained model obtained being different in each execution, even though the same workloads are used for training and the training period is the same. This variability implies that reliance on a particular execution will not yield accurate results; rather, it is important to observe the behavior across a series of executions and training runs to obtain a better picture of the real performance of the model.

Testing Data

An important aspect to consider is that the results presented, including the training progression, are obtained from testing runs, selecting the model's optimal actions, and utilizing a constant workload to facilitate comparison of results over time. The rationale for selecting testing runs over training runs is that, during the training phase, the simulation cycles through a variety of workloads, each one having a different maximum *Score* potential. This variability renders it impossible to accurately assess the performance evolution.

However, testing runs exclusively apply the optimal policy, exploiting the acquired knowledge while entirely omitting exploratory actions, which deviates from a real system's dynamics. Exploration is critical for the reinforcement learning agent to learn the optimal policy by occasionally selecting suboptimal actions, which impacts overall performance. In order to

investigate the implications of omitting exploratory actions, a specific test was conducted. This test involves three consecutive simulations per episode, and a 100 episode duration. Firstly, a standard training episode is performed. With the updated model, a second simulation is executed utilizing the testing workload and disabling training while maintaining the agent's exploration. Finally, a third simulation is performed with the exploration disabled, adhering strictly to the optimal policy. By comparing the second and third simulations, an evaluation of the impact of reinforcement learning exploration on performance can be obtained.

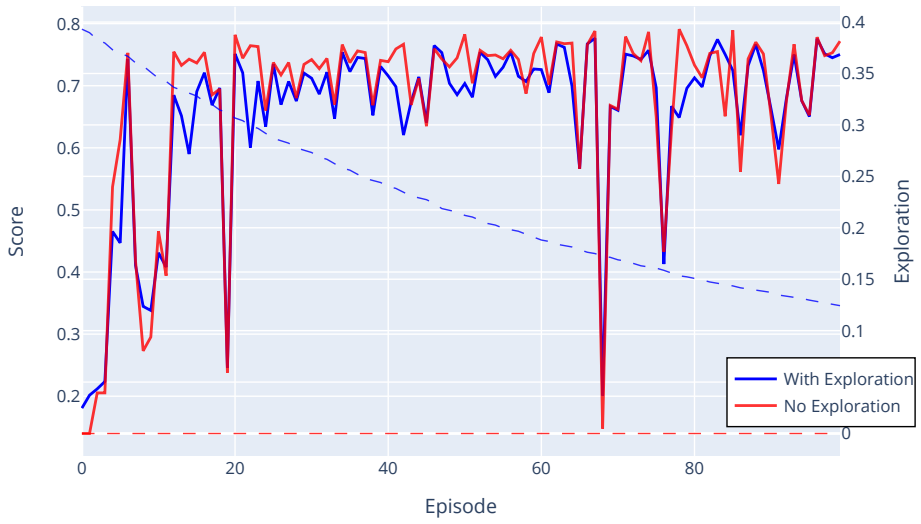

Figure 5.3: RL Exploration Effects

The results, obtained after performing Learn from Demonstrations transfer learning, show minimal deviation between the operational modes, as shown in Figure 5.3. Therefore, testing runs are presented in the forthcoming results for the following reasons:

- More accurate performance measurement: Exploration introduces run-to-run variance, complicating the evaluation of the policy's performance.
- Execution time: The necessity to execute a third simulation for each episode substantially increases the overall execution time.

Additional Observations during Training

An observation made during the training process is that in certain scenarios, particularly at the beginning of training, the model may execute several incorrect actions. This can lead to a state where cloudlets accumulate extensively in the waiting queue. In the most extreme cases, it becomes exceedingly difficult for the controller to recover from this situation, as it requires the selection of correct actions and a considerable period of time for it to return to a stable state. To accelerate the speed of training, a decision was made to limit the number

of cloudlets in the waiting queue, discarding them once a specific threshold is exceeded. The rationale behind this decision is that the system will be in a state from which recovery is simpler, thus affording the controller a greater opportunity to identify the correct sequence of actions to recover and learn from them.

However, this approach has a potential drawback: while the correct actions to recover the system are not identified, the system's state may not be penalized as severely as it should be. By dropping cloudlets, the average response time decreases, and the reward improves. Therefore, it is crucial to find a threshold that strikes an appropriate balance between aiding the controller in finding a solution to recover from a poor state and adequately penalizing incorrect actions that fail to ameliorate the situation. The threshold was determined by observing various simulations and the controller's behavior in a variety of complex scenarios. This process involved analyzing the queue length in cases where the controller successfully recovered from adverse conditions and those where it failed to do so. A value was then selected that lay between the values observed in those two scenarios.

5.3 Overview of Experiments

A series of experiments was conducted to evaluate the efficacy of the various algorithms and techniques studied within the scope of this research.

5.3.1 Hyperparameter Testing

Firstly, a hyperparameter optimization study was conducted with the objective of identifying parameters that maximize the performance of the algorithm. These hyperparameters include:

- Learning rate
- Discount rate
- Optimizer
- Size of replay memory
- Reward function

A grid search technique was employed to evaluate the best combination of parameter values, and the performance was measured on a workload designated for validation.

5.3.2 Transfer Learning Training

A relevant part of transfer learning is the training process using the two distinct techniques. A concise section explores the training evolution for specific samples using both techniques.

5.3.3 Transfer Learning Performance

A focal point of the study concerns the various transfer techniques that have been explored. It is of vital importance that the DRL controller demonstrates robust performance from the moment of its deployment.

The evaluation of different transfer techniques is intended to determine which method is most effective for the specific case study. Reference points for this evaluation include a controller without any prior training and a model based on threshold algorithms. The transfer techniques assessed are:

- None
- Simulation to Reality (Sim-to-Real)
- Learning from Demonstrations

5.3.4 Controller Performance

A fundamental aim was to ascertain the final performance of the implemented model by comparing it against other established solutions addressing cloud resource scaling challenges, as well as against different variants of deep reinforcement learning algorithms. It is crucial to validate that the implemented model can enhance the performance of existing solutions, especially given the augmented complexity inherent in a reinforcement learning algorithm.

The algorithms selected for this comparative analysis include:

- Threshold-based algorithm
- Deep Q-Networks (or Deep Q-Learning)
- N-Step Deep Q-Networks

The threshold algorithm, a commonly employed technique in practice, serves as a benchmark for evaluating the performance of the implemented algorithm. The threshold values were based on the default values from CloudSim Plus and later tuned for better performance, as no real-world references were found. These values were set at 55% utilization for the lower threshold and 80% utilization for the upper threshold. Furthermore, the inclusion of traditional DQN allows the examination of the advantages of incorporating future states in transitions with N-Step Q-Learning, showing the long-term implications of decision-making actions.

All the performance metrics were obtained utilizing a testing workload previously unseen by the reinforcement learning controllers.

5.3.5 Last Layer Training

A common practice noted in the literature during the retraining process is to exclusively update the weights of the final layer of the neural network, while keeping the remaining weights unchanged. Therefore, two distinct approaches for retraining are examined: a conventional method, where all network weights are adjusted during retraining, and an alternative method, where training is restricted to the last layer's weights of the network.

5.4 Training Challenges Encountered

5.4.1 RL Controller Challenges

During the development of the controller, several challenges were encountered, two of which proved particularly complex.

One such challenge was ensuring that the reinforcement learning controller learned the correct actions for a given state. Initially, even when tested with extremely simple workloads, the controller struggled to make optimal decisions. Identifying the cause of this behavior took a considerable amount of time. The reason this occurred was due to an initial limitation of the simulator, which did not allow resources to be scaled down if all resources were being utilized; that is, the action of downscaling was not permitted under such circumstances. This technical limitation initially appeared reasonable, since removing resources which are actively being utilized might seem counter-intuitive. However, this assumption inadvertently hindered the controller's learning process, as it restricted the controller's ability to explore all available actions for certain states, resulting in incomplete information about expected rewards. Consequently, the model was unable to fully comprehend the environment, impairing its ability to make optimal decisions. Once this functionality was incorporated into the simulator, there was a marked improvement in the controller's performance, enabling it to effectively execute its designated task.

Similarly, a related issue was encountered when pre-training utilizing learn from demonstrations. Initially, the training algorithm simply loaded the expert's samples into the replay memory buffer and used these transitions for training. Since the transitions used come from an expert, they all involve optimal actions, providing no information about the environment's behavior when performing suboptimal actions. To ensure that the selected actions are correct, a supervised large margin classification loss is used, which ensures that actions different from those selected by the expert receive a lower estimated reward. This method is not without its difficulties, which are detailed later.

The second challenge encountered involves the inconsistency of trained models, particularly when subjected to a new round of training. Specifically, a model that has been trained for an extended period and achieved a certain policy and level of performance may exhibit larger than expected variations in these aspects if subjected to a new training episode. Several measures were implemented to address this issue. These included testing various hyperparameters, such as the optimizer and learning rate, adjusting the input parameters to ensure they provide representative data, and employing more robust algorithms like N-Step Deep Q-Learning. While these strategies have mitigated the performance fluctuations and resulted in more stable models, they have not fully resolved the issue, and the models continue to exhibit a higher than desired degree of variance.

5.4.2 Transfer Learning Challenges

Both transfer techniques present practical difficulties.

In the case of Sim-to-Real, the challenge lies in developing a simulator that accurately replicates the behavior of the real system. The greater the differences between simulation and reality, the worse the outcomes when applying transfer learning. While this challenge would apply in the proposed scenario, it does not apply to the specific solution implemented. No additional challenges were encountered.

For the case of learning from demonstrations, the challenges are not as obvious. On the one hand, it is necessary to obtain expert demonstrations. This is trivial in the proposed use case, where an existing controller is being replaced. The challenge arises in training the model with these demonstrations.

The training process is similar to traditional reinforcement learning training, but additional considerations must be taken into account to mimic the expert's policy, including the manual selection of a number of parameters. In training, it is necessary to consider not only the Q-Learning loss function but also two additional losses: the supervised large margin classification loss and the regularization loss.

For the first loss, a margin must be defined, which is used to generate a difference between the expected Q-values of the action selected by the expert and the other actions. The correct selection of this margin is crucial, as it must best represent the differences encountered in the real environment. If a large margin is selected, it may help the model better imitate the expert's policy; however, once retraining begins, it may lead to poor results as the expected Q-values implicitly stored in the model will differ significantly from those observed in practice, leading to a retraining phase with suboptimal outcomes. This behavior can be seen in Figures 5.4 and 5.5, where utilizing a larger margin provides better initial performance but suffers once training is resumed. For the results evaluation, two models are created, one with a larger margin and one with a smaller margin.

In cases where the difference in Q-values between actions is very small, as is the case in this study, the selection of this margin value becomes even more complex. If very small values are used, the effect of the supervised loss will be minimized and may be diluted by

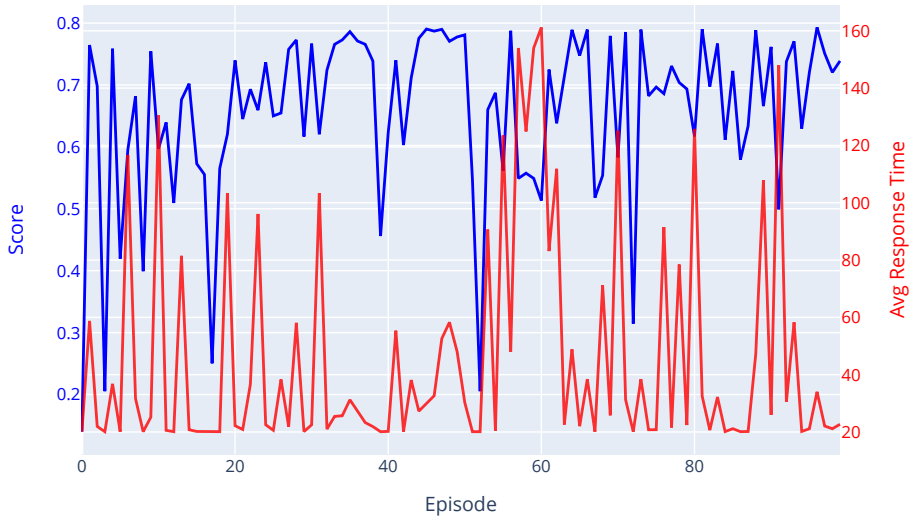

Figure 5.4: Retraining Learn from Demonstrations - Small Margin

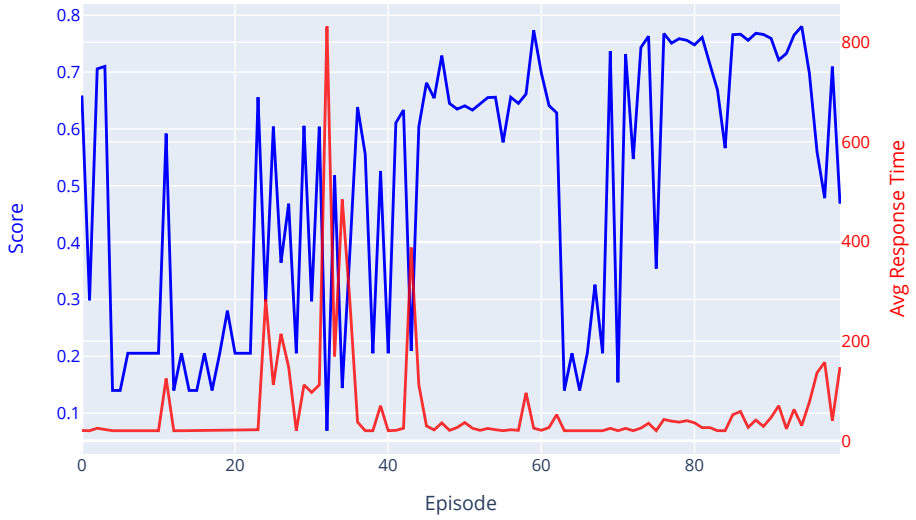

Figure 5.5: Retraining Learn from Demonstrations - Large Margin

Q-Learning loss values from other transitions as well as by the regularization loss.

Regarding the regularization loss, it is necessary to define a value that prevents overfitting to the demonstrations, which can be particularly problematic in this use case where the model input consists of several parameters, but only one of them, the CPU utilization, is relevant for imitating the policy of the original expert.

In addition to determining the large margin classification loss and regularization loss values, another critical consideration is establishing the stopping condition for the training process. The model must be trained sufficiently to converge but not excessively, in order to avoid overfitting to the limited set of demonstrations. Selecting the optimal threshold value is challenging, especially given the complex interactions between these factors.

In summary, the definition of all these parameters is a highly complex task that can produce very different results depending on whether the parameters are selected correctly or not. This complexity is further exacerbated by the difficulty in testing the pre-trained model before deploying it in the actual environment. While it is possible to validate that the actions taken by the model imitate those of the expert, it is not trivial to assess whether the policy, once retraining begins, will be successful.

5.5 Hyper Parameter Testing

Several tests were conducted to determine which hyperparameters maximize the performance of the controller. For this, a common validation workload was utilized for all experiments.

In the case of four of the hyperparameters, the Reward Function, Learning Rate, Discount Rate, and Replay Memory Size, a predefined set of values was selected and tests were conducted with all combinations of these four hyperparameter values, a method known as grid search. To analyze the optimizer, the best values of the previously mentioned hyperparameters were used, iterating over possible reward functions and the potential optimizers to identify those that yielded the best performance. The rationale for this approach was to reduce the number of combinations to be evaluated, thus enabling the tests to be conducted within a reasonable time frame.

Subsequently, for each combination of hyperparameters, multiple runs were conducted in order to obtain a more comprehensive understanding of the actual effects of each hyperparameter, thereby minimizing the impact of the fluctuations observed in different executions.

Finally, for the analysis of the results, data from all tests were aggregated by the values of each hyperparameter, yielding results for the execution that maximized the *Score*, as well as for the median execution. The median rather than the average was chosen for analysis, as certain parameter combinations could result in particularly poor outcomes that significantly skew the average.

The results are presented in the Table 5.1, and analyzed in the following sections.

5.5.1 Study of the Reward Function

The behavior of the two previously mentioned reward functions is examined. One of these, $RF\,B$, is more traditional, composed of both response time and the amount of resources allocated. In contrast, $RF\,A$ also considers response time but replaces the quantity of allocated resources with a target value for resource utilization. This target indirectly penalizes resource overprovisioning and maintains a reserve of unutilized resources, serving as a buffer against sudden increases in demand. Setting a slightly lower target results in a more resilient algorithm to sudden changes in demand as well as suboptimal decisions.

Among the two reward functions employed, the more innovative approach, $RF\,A$, generally achieves better performance. This is attributed to the CPU utilization margin it provides. When evaluated with suboptimal hyperparameters that do not lead to an optimal

model, it still manages to accommodate the load during sudden changes in demand. Meanwhile, the more traditional reward function often leaves little CPU margin, and in the event of a poor decision or failure to anticipate demand changes, it quickly becomes overwhelmed.

While $RF\,A$ performed better, it was of interest to conduct the subsequent experiments with both reward functions, as it is considered important to assess the behavior of each one under optimal conditions, compare different reinforcement learning models, and evaluate each of their performances in transfer learning.

5.5.2 Study of Learning Rate

Three predefined learning rate values were evaluated:

- 0.001
- 0.003
- 0.01

No significant differences were observed among these three learning rate values, although the value 0.003 yielded slightly better results. Therefore, this value is used for the remainder of the experiments.

5.5.3 Study of Discount Rate

Again, three possible values for the discount rate were defined:

- 0.95
- 0.8
- 0.5

In this case, a noticeable difference was observed in the results obtained for the studied values, where the lowest discount rate value produced notably worse results. The remaining two values showed similar outcomes. Ultimately, the value of 0.95 was chosen as it obtained slightly better results for median, average, and best runs, while also being the most consistent one.

5.5.4 Study of Replay Memory Size

For the possible values of Replay Memory size, only two distinct values were extensively evaluated: **(1000, 5000)**. It can clearly be observed that the larger Replay Memory size yields superior results across all metrics compared to the smaller one. Initially, further tests using even larger values were conducted, and it was observed that further increasing the size of the replay memory did not provide additional benefits.

5.5.5 Study of the Optimizer

Finally, four different optimizers were evaluated:

- Adam
- AdamW
- RMSprop
- ASGD

In this case, the best values of the previous hyperparameters were used, and tests were conducted with these fixed values. RMSprop was selected as the best option because it provides the best average performance and, additionally, shows the most consistency in resource utilization.

Table 5.1: Hyperparameter Testing Results

Reward Function				
Result	Reward Function	Score	Avg RT	Avg CPU Cores
Median	RF A	0.49	93.68	7.72
Best	RF A	0.63	53.25	7.38
Median	RF B	0.34	196.67	7.08
Best	RF B	0.62	55.20	7.42
Replay Memory Size				
Result	RMS	Score	Avg RT	Avg CPU Cores
Median	1000	0.42	251.41	7.71
Best	1000	0.62	43.04	7.67
Median	5000	0.48	22.48	9.88
Best	5000	0.63	53.25	7.38
Learning Rate				
Result	Learning Rate	Score	Avg RT	Avg CPU Cores
Median	0.001	0.44	152.39	7.73
Best	0.001	0.62	49.21	7.36
Median	0.003	0.45	150.17	7.48
Best	0.003	0.63	53.25	7.38
Median	0.01	0.43	135.16	8.43
Best	0.01	0.62	55.20	7.42
Discount Rate				
Result	Discount Rate	Score	Avg RT	Avg CPU Cores
Median	0.95	0.51	73.63	7.67
Best	0.95	0.63	53.25	7.38
Median	0.8	0.48	106.19	7.89
Best	0.8	0.62	55.20	7.42
Median	0.5	0.36	178.32	6.92
Best	0.5	0.57	34.67	8.26
Optimizer				
Result	Optimizer	Score	Avg RT	Avg CPU Cores
Median	ASGD	0.24	20.20	13.84
Best	ASGD	0.46	21.95	10.29
Median	Adam	0.44	97.98	7.80
Best	Adam	0.61	38.18	7.59
Median	AdamW	0.53	42.44	8.37
Best	AdamW	0.61	37.16	7.79
Median	RMSprop	0.54	55.04	7.76
Best	RMSprop	0.61	34.71	7.88

This page intentionally left blank.

Chapter 6

Evaluation Results

In this section, the results obtained from executing the experiments described in the previous section are shown and analyzed. A testing workload was employed for all the final results analysis, as well as the testing runs performed between episodes to evaluate the performance progression. Naturally, it differs from the workloads used to train the model and to validate the hyperparameters.

The decision to use only one specific workload for validation and another for testing stems from the fact that simulations with different workloads will yield different *Score* values, as a workload with a higher average number of cloudlets per unit of time will require more cores to satisfy demand, which incurs a higher cost and consequently, the *Score* will be lower. Additionally, the limited availability of workloads generated from real logs restricted the number of workloads that could be reserved for validation and testing.

6.1 Code

A Jupyter Notebook with the analysis presented below can be found in this notebook.

6.2 Transfer Learning Training

The training process for both transfer learning techniques is characterized by distinct evaluation methods and performance trajectories, reflecting their unique approaches to learning.

For Sim-to-Real, the training evolution can be observed in Figure 6.1. The performance shown in this figure is measured during testing runs between training runs. It can be observed that it achieves a high-performance level within 20 episodes, and during the remaining episodes, it fluctuates as new scenarios are explored and new policies learned, but ultimately the performance ceiling remains largely unchanged.

For Learn from Demonstrations, it is not as simple to measure the performance. Therefore, the training loss is used to evaluate the learning process. With this technique, the learning evolution follows a typical training process, with large improvements in the early stages and slowly converging to a baseline value. This can be observed in Figure 6.2.

It is worth mentioning that this graph can have different convergence speeds and baseline values depending on the parameters selected for training, as previously discussed in Section 5.4.2.

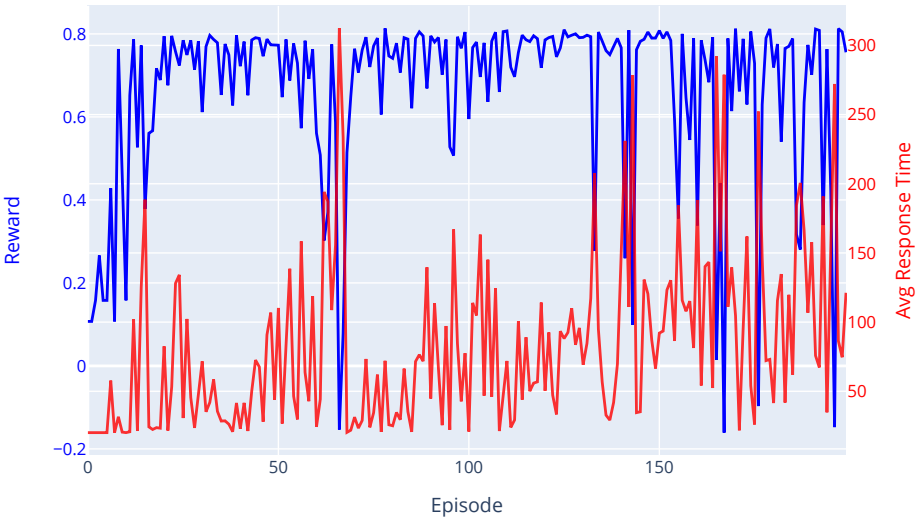

Figure 6.1: Sim-to-Real Training

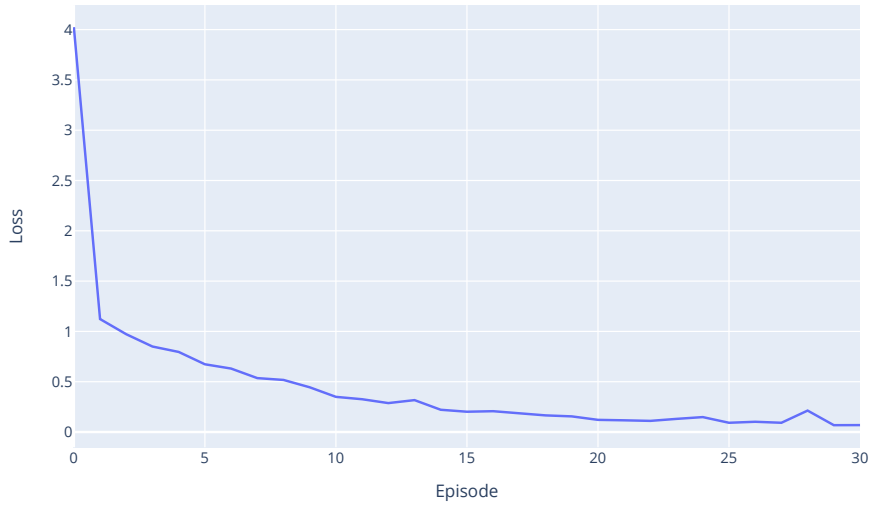

Figure 6.2: Learn from Demonstrations Training

6.3 Transfer Learning Performance

This section presents the performance evaluation of transfer learning. Once the models have been pre-trained using both transfer learning techniques, they are evaluated in the simulated reality environment. The models trained are based on the N-Step Deep Q-Learning variation previously described.

6.3.1 Differences between Simulation and Simulated Reality

During the pre-training phase in the simulator, a simulated environment is used, which then undergoes certain changes in order to obtain the simulated reality environment. These differences include distinct workloads and the addition of two delays: one for resource scaling, representing the time between issuing a command and resource availability, and another for monitoring, where the controller relies on outdated system state information.

Table 6.1: Sim-to-Real - Transfer Effects - Average

Rew. Fun.	**Delays**	**Avg Score**	**Avg RT**	**Avg CPU Cores**
RF A	No Delays	0.78 ± 0.02	24.81 ± 2.07	5.17 ± 0.28
RF A	Scaling Delay	0.75 ± 0.03	54.65 ± 22.72	4.80 ± 0.18
RF A	Scaling & Monitoring Delay	0.76 ± 0.03	49.87 ± 14.45	4.78 ± 0.18
RF B	No Delays	0.75 ± 0.09	80.97 ± 50.56	5.18 ± 1.53
RF B	Scaling Delay	0.43 ± 0.25	201.82 ± 143.85	4.91 ± 1.33
RF B	Scaling & Monitoring Delay	0.40 ± 0.25	230.29 ± 158.54	4.95 ± 1.34

Table 6.2: Learn from Demos. - Transfer Effects - Average

Rew. Fun.	**Delays**	**Avg Score**	**Avg RT**	**Avg CPU Cores**
RF A	No Delays	0.78 ± 0.02	31.14 ± 4.15	5.13 ± 0.25
RF A	Scaling Delay	0.60 ± 0.04	126.24 ± 14.30	4.69 ± 0.06
RF A	Scaling & Monitoring Delay	0.62 ± 0.08	101.41 ± 47.17	4.82 ± 0.19
RF B	No Delays	0.76 ± 0.03	34.28 ± 14.45	5.36 ± 0.56
RF B	Scaling Delay	0.63 ± 0.11	88.65 ± 67.76	5.06 ± 0.65
RF B	Scaling & Monitoring Delay	0.60 ± 0.14	108.95 ± 90.60	5.10 ± 0.70

In Table 6.1 and 6.2, the effects of these delays can be observed. These tables compare the performance of the same models on the same workload but gradually introduce the previously discussed delays. Analyzing the results for Sim-to-Real, it becomes evident that the scaling delay has a much larger impact than the monitoring delay. This can be observed particularly with reward function B, which does not have any explicit buffer for CPU utilization. The model trained with reward function A is better able to handle the delay in response to its actions, managing to maintain performance.

In the case of Learn from Demonstrations, both reward functions are significantly impacted by the introduction of the scaling delay. Additionally, they are impacted to a similar extent, which is expected given the fact that they were trained to follow the same policy.

It may seem unexpected that the controller suffers so much with the addition of these delays, considering that the demonstrations were derived from a controller based on thresholds operating in the simulated reality environment with such delays already present. However, the model is trained to mimic the threshold-based controller, which lacks the capability to alter its behavior to account for the scaling delay. The monitoring delay seems to have little effect.

It can be observed that in all scenarios the average number of resources used prior to the introduction of the scaling delay is higher than after the introduction of said delay. This is because it takes longer to add them, but not any longer to remove them. However, this reduction in cost is negated by the increase in response time and the increase in penalties for exceeding the predefined threshold.

6.3.2 Transfer Learning Results

The results obtained from transfer learning can be observed in Table 6.4 for the average results and Table 6.3 for the best run. These results are based on the metrics previously discussed in Section 5.2.2. As a reference point, the *Score* achieved by the thresholds-based policy is **0.72**.

Jumpstart Performance

The results obtained without any pre-training of the model are predictably poor. The performance varies significantly depending on the random initialization of the neural network weights. Two outcomes are commonly observed: either the actions taken by the agent primarily involve increasing resources, leading to good response times but excessive resource allocation, resulting in very high costs; or the predominant actions involve decreasing resources, causing demand to far exceed availability, which renders the system unable to meet the demand.

For the Sim-to-Real approach, promising results are observed, particularly with reward function A. In this case, the initial results are even better than those achieved with the threshold-based controller. For reward function B, while the results are not poor, they are inferior to those of the threshold-based controller, and experience a notable decrease in performance when the scaling delay is introduced.

The reason why the scaling delay affects the second reward function more significantly is due to the fact that the first function has a specific target for CPU utilization of 80%. This target implicitly creates a safety margin, allowing for residual resource capacity to meet sudden increases in demand. When the scaling delay is introduced, the action of increasing resources takes a certain period to complete, and if the system is already near its capacity limit, it will not be able to satisfy the increased demand.

In the case of Learn from Demonstrations, the results are not as favorable as those achieved with Sim-to-Real, although they are still satisfactory and present a marked im-

provement over the variant with no transfer. In the best-case scenario, the performance is comparable to that of the threshold-based controller, which is as good as could be expected, given that it uses the threshold-based controller as the expert to imitate. On average, the performance is slightly below what can be achieved with a threshold-based model. There is little performance difference between the two reward functions, which follows the fact that the policy from which the model was trained was the same for both reward functions.

Here, the difficulty of achieving good model training from the demonstrations becomes apparent. Although it is possible to consistently obtain good results by selecting a bigger value for the large margin classification loss, this would have negative effects on other aspects of the transfer process.

Accumulated Rewards

Another crucial aspect to consider is the performance obtained in the first few episodes following the transfer. For this purpose, the accumulated rewards metric is used, which represents the sum of the rewards over the first five episodes. However, in this case, the sum of the *Score* is used instead of the reward.

The importance of this metric lies in the fact that, once the transfer is performed, retraining begins. During this process, the controller interacts with the new environment and gathers new information about its behavior. The model's expected reward values are updated during this process, starting from the expected values calculated based on the previous training phase and adjusting them with the new observed values. If these values differ significantly, the system may encounter a situation where the values of some actions have been updated while others have not, potentially leading to an unpredictable policy. Therefore, it is crucial that the expected reward values remain as consistent as possible to ensure a good performance level during the early stages of the retraining process.

In the case of Sim-to-Real, the same reward functions are used, with the only difference being the environment dynamics.

In the case of Learn from Demonstrations, the situation is more complex. Initially, only reward data for "optimal" actions was available, and none for "non-optimal" actions. A synthetic gap is created between the rewards for desired and non-desired actions, which may not align with the gap encountered when interacting with the new environment.

For the Sim-to-Real approach, the results are good, with only a slight decrease in average performance over the following five episodes.

In the case of Learn from Demonstrations, the performance starts at a lower point. However, depending on the execution, performance sometimes quickly recovers, while in other cases, there is an initial decline before any improvement is observed. This variation depends on the selected values for the margin and regularization, as previously shown in Figures 5.5 and 5.4.

In the case of the model without pre-training, the average results are notably underwhelming, although there are instances where it can quickly learn an effective policy and achieve satisfactory performance levels. Nevertheless, this metric clearly highlights the challenges of deploying reinforcement learning solutions without pre-training in real-world environments, as the initial performance is insufficient to meet practical demands.

Time to threshold

To define the performance threshold, a target value close to that achieved by the threshold-based controller was selected **0.7**, as the performance is considered satisfactory beyond this point.

For the Sim-to-Real approach, the performance can reach the threshold from the very first episode, and on average, it achieves a satisfactory level of performance within a few

episodes. This implies that Sim-to-Real is a suitable transfer learning technique for the case study.

In the case of Learn from Demonstrations, a satisfactory level of performance can also be achieved from the first episode, although it generally takes slightly longer to raise the performance up to the threshold level.

Performance after 10 Episodes

Both transfer learning techniques show significant improvement compared to no previous training. In the case of Sim-to-Real, the performance obtained after only ten episodes nearly matches the best performance attainable for both reward functions. Even on average, for reward function A, the performance comfortably surpasses that of the threshold-based policy.

Learning from demonstrations also provides a notable performance boost, particularly in the average results. For the best run, it produces a slightly lower result than no training for reward function A, but a significantly higher score for reward function B. For the average run, it produces markedly better results than with no previous training, but it lags behind Sim-to-Real, and particularly struggles to achieve good results with reward function B.

Asymptotic Performance

After retraining, all models show improved performance compared to the results obtained immediately after the transfer. Additionally, the different transfer learning techniques, in all executions, produce almost identical results, leading to average results that are on par with the results obtained in the best runs.

For the Sim-to-Real approach, the improvements observed with reward function A are modest, partly because the initial performance was already strong. The improvements stem from a reduction in penalties, despite a slight increase in resource usage. This minor enhancement is attributed to the previously mentioned fact that the reward function creates a buffer, allowing the system to effectively handle changes in the new environment, even reducing the average number of cores used without significant penalties for excessive response times. In the case of reward function B, the improvement is much more pronounced, with a significant reduction in response times without a notable increase in resource usage.

For Learn from Demonstrations, although the initial performance is slightly weaker, the retraining process brings the performance up to similar levels, as expected.

For the model without any pre-training, although the initial performance is quite poor, it eventually improves to reach comparable levels to that of the other transfer learning methodologies.

Transfer Learning Results Summary

Overall, both techniques significantly enhance the controller's performance upon deployment, demonstrating the potential to make reinforcement learning viable in practical scenarios. Among the two, Sim-to-Real exhibited particularly strong results, outperforming the threshold-based controller from the outset.

While the results of Learning from Demonstrations were positive, they did not match the performance levels achieved by Sim-to-Real. However, the training process for this technique, particularly the selection of parameters such as the margin for the large-margin classification loss, is not yet fully understood. Further research and investigation could provide a deeper understanding of the process, enabling refinements that improve the technique itself and, consequently, the controller's performance.

Table 6.3: Transfer Learning Performance Metrics - Best Results

Rew. Fun.	Metric	No Transfer	Sim-to-Real	Learn from Demos.
RF A	Jumpstart Performance (Score)	0.21	0.79	0.69
RF A	Accumulated Rewards (Sum Score)	2.09	3.74	3.37
RF A	Time to Threshold (Episodes)	4	0	1
RF A	Performance after 10 Episodes (Score)	0.77	0.77	0.74
RF A	Asymptotic Performance (Score)	0.79	0.79	0.79
RF B	Jumpstart Performance (Score)	0.47	0.71	0.73
RF B	Accumulated Rewards (Sum Score)	3.23	3.37	3.37
RF B	Time to Threshold (Episodes)	18	0	0
RF B	Performance after 10 Episodes (Score)	0.21	0.78	0.60
RF B	Asymptotic Performance (Score)	0.80	0.80	0.80

Table 6.4: Transfer Learning Performance Metrics - Average Results

Rew. Fun.	Metric	No Transfer	Sim-to-Real	Learn from Demos.
RF A	Jumpstart Performance (Score)	0.18 ± 0.05	0.76 ± 0.03	0.62 ± 0.08
RF A	Accumulated Rewards (Sum Score)	1.52 ± 0.50	3.51 ± 0.47	2.28 ± 1.01
RF A	Time to Threshold (Episodes)	20.67 ± 24.66	0.25 ± 0.71	6.50 ± 3.62
RF A	Performance after 10 Episodes (Score)	0.36 ± 0.35	0.75 ± 0.01	0.65 ± 0.07
RF A	Asymptotic Performance (Score)	0.79 ± 0.00	0.79 ± 0.01	0.79 ± 0.00
RF B	Jumpstart Performance (Score)	0.32 ± 0.18	0.40 ± 0.25	0.60 ± 0.14
RF B	Accumulated Rewards (Sum Score)	1.65 ± 1.36	2.60 ± 0.49	2.76 ± 0.31
RF B	Time to Threshold (Episodes)	22.67 ± 4.51	2.62 ± 3.85	1.83 ± 2.23
RF B	Performance after 10 Episodes (Score)	0.16 ± 0.04	0.61 ± 0.18	0.36 ± 0.20
RF B	Asymptotic Performance (Score)	0.79 ± 0.00	0.79 ± 0.00	0.80 ± 0.01

6.4 Controller's Final Results

This section presents a comparative analysis of different controller alternatives to assess the performance of the implemented solution. Additionally, it examines the number of actions taken by each controller, providing insights into their efficiency. Finally, the results of retraining the neural network by updating only the weights of the last layer are evaluated and compared to a full-network update approach.

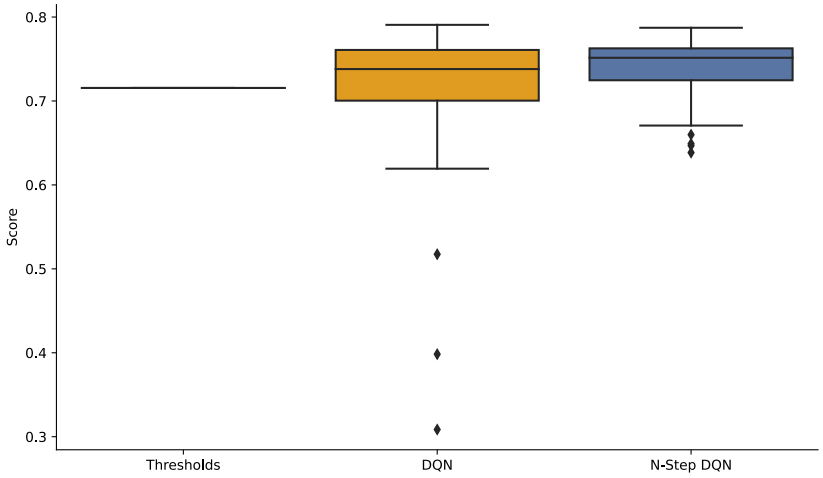

Figure 6.3: Controller Performance Comparison - RF A

6.4.1 Controller Performance - Thresholds vs DQN vs N-Step DQN

In this section, a comparison between the three different controller implementations is presented. These include a threshold-based approach as a reference point and two RL-based models: the first one using a standard Deep Q-Network and the second based on N-Step DQN. For the machine learning-based algorithms, the best hyperparameters have been selected.

Furthermore, two outcomes are presented: one that represents the best result achieved and another that averages the results obtained across multiple executions.

The results are presented in Table 6.5, the metrics for which were previously discussed in Section 5.2.3.

The different solutions provide different advantages and disadvantages. The advantages offered by the solution based on Thresholds pertain to its interpretability, as it is simple to validate the controller's decisions by simply checking the system's state. Moreover, it maintains consistency over time and across various situations, making it predictable. Although the outcomes may not be the most optimal, this algorithm ensures satisfactory performance in terms of response times in most scenarios without incurring excessively high resource costs. Additionally, since no training is required, it delivers consistent performance, which can be observed in Figure 6.3.

The results obtained for the Deep Q-Network (DQN) place it on par with the threshold-based solution for average execution for Reward Function A, and below for Reward Function

B. Utilizing Reward Function A, which targets an 80% CPU utilization, yields more consistent results compared to Reward Function B, due to its margin in CPU usage. Reward Function B, lacking this margin, is more susceptible to incorrect decisions by the agent, which is evident from a significant difference between average and best executions.

For the best results of DQN, it significantly surpasses the threshold-based implementation. It is noteworthy that in all runs, the DQN achieves values higher than the Threshold solution at some point, but the model fails to maintain this performance level all throughout the training process, exhibiting variations in performance.

The performance level achieved with N-Step Deep Q-Learning surpasses that achieved by standard DQN. Although no significant improvements are observed in the best case, differences are noticeable in the average results. For Reward Function B, the improvement is particularly notable, highlighting the advantages of the N-Step approach. This variant more effectively anticipates demand fluctuations, which is crucial given that Reward Function B lacks the safety margin present in Reward Function A. As a result, it helps prevent situations where demand cannot be met, as can be seen by the reduced response times and the percentage of requests that exceed the predefined threshold. Additionally, the N-Step model demonstrates greater consistency, with reduced performance fluctuations, making it more reliable for practical applications.

Notably, in all executions of the N-Step model, the controller demonstrates the ability to achieve performance levels that match or even surpass the presented "best" result at some point during training. However, it fails to sustain this level of performance by the end of the training process, when these measurements were extracted.

Based on these observations, it can be stated that, out of the two reinforcement learning-based solutions, the N-Step model is more robust and capable of yielding better results in less optimal or more complex situations due to a deeper understanding of the long-term effects of actions.

Comparing the best execution of the N-Step Deep Q-Learning controller to that of the threshold-based controller reveals a substantial improvement. While the *Score* reflects only a 10% increase (**0.72** vs. **0.79**), this translates to a more significant 20% reduction in CPU resource usage (**5.99** vs. ∼**4.9**), while at the same time reducing the number of SLA violations.

DRL Controller Results Summary

Overall, the reinforcement learning models, particularly N-Step Deep Q-Learning, show significant promise. When achieving its best results, it can offer substantial advantages over traditional methods.

However, stability remains a key limitation in the current implementation of Deep Q-Network (DQN)-based models. Notable differences can be observed between contiguous episodes, even after prolonged training periods, during which only minor changes would typically be expected.

As a result, the average performance of DQN remains comparable to or slightly lower than traditional methods, while N-Step DQN achieves slightly better results.

Single Simulation Comparison

While the previous results provide an overall comparison of the different algorithms, a more detailed analysis of each controller's performance in specific workload executions offers valuable insights. In particular, Figures 6.4, 6.5, and 6.6 illustrate the evolution of the *Score*, *Average Response Time*, and *Number of CPU Cores* over the 24-hour simulation period, alongside the corresponding demand for reference. To enhance clarity, the data has been smoothed for better visualization. These figures compare the performance of the N-Step DQN controller against the threshold-based controller.

Table 6.5: Algorithms Comparison

Thresholds					
Result	Reward Function	Score	Avg RT	% RT > 200	Avg CPU Cores
NA	NA	0.72	24.74	0.98	5.99
Deep Q-Learning					
Result	Reward Function	Score	Avg RT	% RT > 200	Avg CPU Cores
Average	RF A	0.72	64.25	2.79	5.68
Best	RF A	0.79	34.04	0.0	4.95
Average	RF B	0.64	68.13	9.31	5.80
Best	RF B	0.79	31.44	0.02	5.03
N-Step Deep Q-Learning					
Result	Reward Function	Score	Avg RT	% RT > 200	Avg CPU Cores
Average	RF A	0.74	34.74	2.15	5.40
Best	RF A	0.79	40.58	0.70	4.89
Average	RF B	0.73	40.98	2.94	5.37
Best	RF B	0.79	38.40	0.0	4.91

Two key observations emerge from these results. First, the threshold-based controller exhibits a noticeable drop in *Score* early in the simulation. This decline is associated with a sharp increase in response time, which, although not visible in the plot due to smoothing, exceeds the SLA-defined threshold of 200 seconds. The steep rise in response time results from a sudden increase in demand, which the threshold-based model is unable to anticipate. Second, the N-Step variant consistently allocates fewer resources, leading to slightly higher average response times. However, since these response times remain within the predefined penalty threshold, the overall *Score* remains consistently higher.

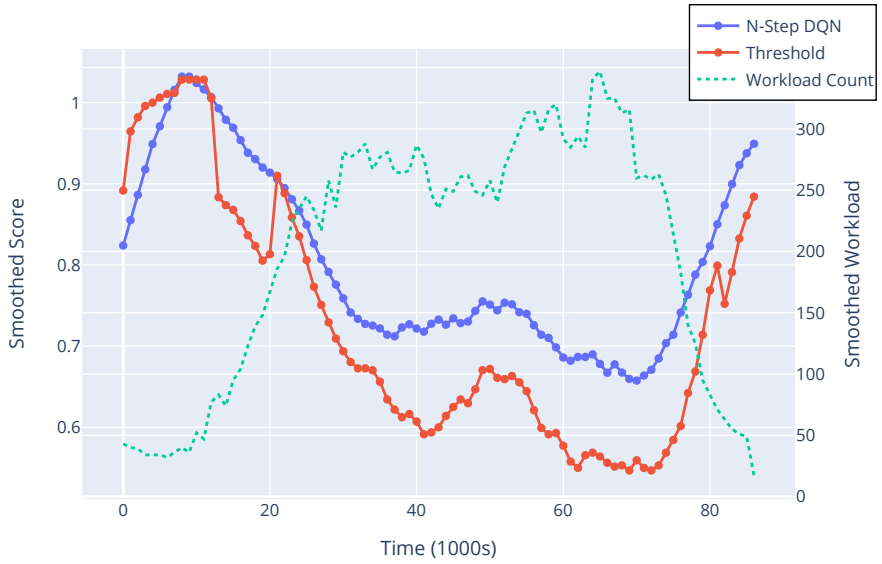

Figure 6.4: Score - N-Step DQN vs Threshold

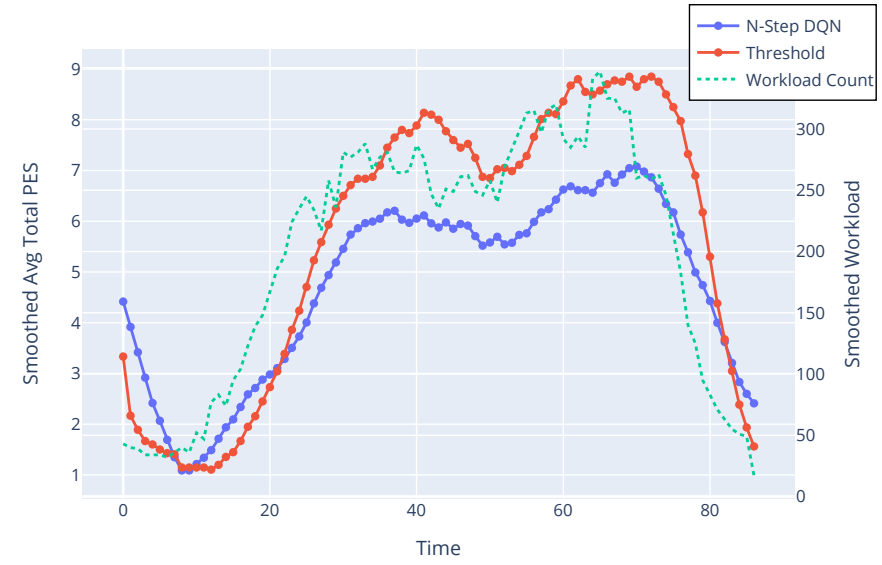

Figure 6.5: Assigned Resources - N-Step DQN vs Threshold

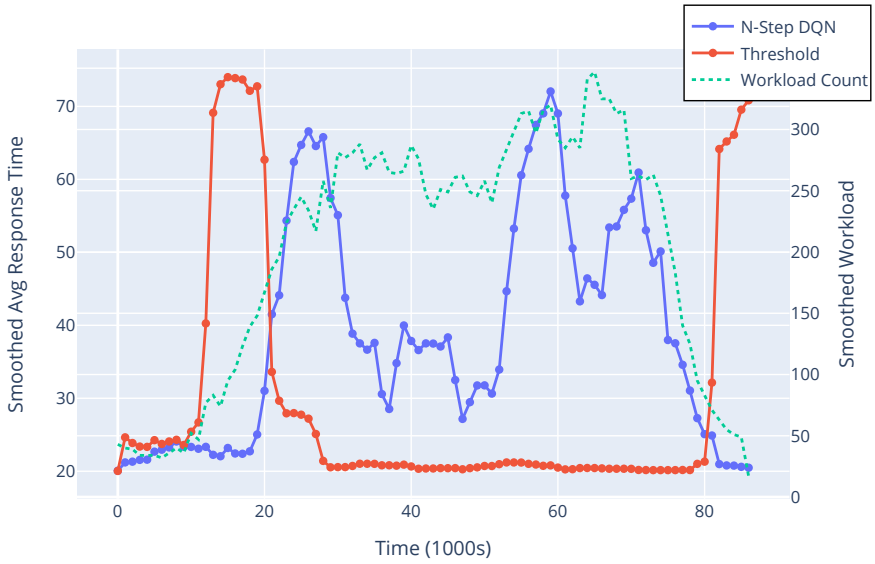

Figure 6.6: Avg Response Time - N-Step DQN vs Threshold

6.4.2 Total Number of Actions Taken

Another key performance metric for comparing the different models is the number of scaling actions taken by the controller. Given the interesting results obtained and their potential implications, it is considered valuable to explore this metric in a dedicated section.

A high number of scaling actions is considered disadvantageous for several reasons. Firstly, frequent scaling actions reduce the stability of the allocated resources, increasing the uncertainty about the available resource pool for the cloud provider. This instability may lead to an over-provisioning of the resource pool for concurrently running applications to ensure that the total capacity is not exceeded. Additionally, during the provisioning period of a new resource, that resource is reserved but cannot yet process incoming requests, causing it to be effectively unutilized. Therefore, minimizing the number of actions is a desirable goal.

Upon analyzing the number of scaling actions taken by each algorithm, significant run-to-run variance was observed in the RL-based model. On average, the RL-based models executed **70** scaling actions during the testing run, compared to **88** actions by the threshold-based controller, representing a 20% reduction in scaling actions. However, in some instances, the RL controller performed up to **125** scaling actions during the simulation, while on other occasions it took as few as **14** actions.

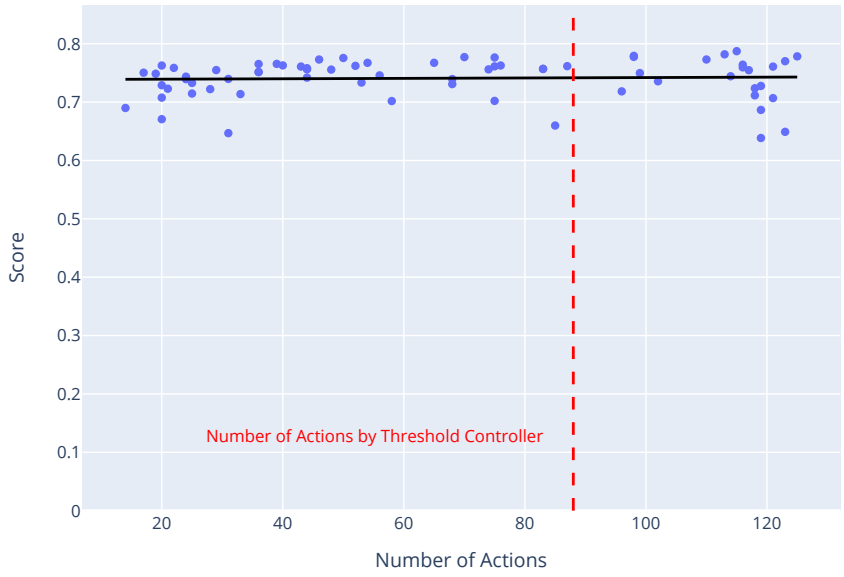

Figure 6.7: Correlation between Number of Actions and Performance

Given that a lower number of actions is desirable, an analysis was conducted to determine whether the number of scaling actions impacts overall performance. Pearson's correlation statistical analysis revealed a correlation coefficient of **0.038** and a p-value of **0.755**, indicating little to no correlation between the number of actions taken by the controller and the resulting performance. The high p-value suggests that this weak correlation is statistically insignificant, meaning it is unlikely that there is a meaningful relationship between the two

variables. These results can be clearly observed in Figure 6.7.

However, this statistical test may not fully capture the underlying relationships. In simulations where the highest performance level is achieved, the number of scaling actions typically exceeds 40. Additionally, there are some cases with a high number of scaling actions but poor performance, which skews the results. These instances may correspond to scenarios where the controller executes unnecessary actions or loops between scaling actions due to an unstable policy, which in turn compromises performance.

In summary, the results suggest that the number of scaling actions does not have a significant impact on the model's performance, up to a certain extent. This opens up an interesting avenue for future work: modifying the RL controller to minimize the number of actions taken while maintaining high performance, thus further enhancing the benefits already observed with deep reinforcement learning.

6.5 Last Layer Training Results

Another aspect of interest to evaluate is the difference between retraining all layers of the network and retraining only the last layer after performing the transfer.

To investigate this, experiments were conducted where both approaches start from the same pre-trained base model, followed by the fine-tuning phase. Specifically, the initial model used was obtained after pre-training in the simulator.

Multiple runs were performed to gain a more accurate understanding of the real behavior of both techniques. The metrics used for analysis are the same as those applied to transfer learning, with a particular focus on *Asymptotic Performance* to validate that both methods can achieve optimal results after retraining, as well as *Accumulated Rewards* and *Time to Threshold* to assess the initial behavior once retraining begins.

Table 6.6: Comparison between Full Network and Last Layer Re-Training - Average

Rew. Fun.	**Metric**	**Full Network Training**	**Last Layer Training**
RF A	Jumpstart Performance (Score)	0.74 ± 0.0	0.74 ± 0.0
RF A	Asymptotic Performance (Score)	0.79 ± 0.01	0.77 ± 0.01
RF A	Accumulated Rewards (Sum Score)	3.74 ± 0.08	3.66 ± 0.09
RF A	Time to Threshold (Episodes)	0.0 ± 0.0	0.25 ± 0.50
RF B	Jumpstart Performance (Score)	0.71 ± 0.0	0.71 ± 0.0
RF B	Asymptotic Performance (Score)	0.80 ± 0.0	0.80 ± 0.0
RF B	Accumulated Rewards (Sum Score)	2.29 ± 0.37	1.90 ± 0.09
RF B	Time to Threshold (Episodes)	4.25 ± 0.50	13.0 ± 8.83

According to the results observed in Table 6.6, retraining only the last layer of the neural network does not provide any advantages over a full network retraining. In particular, the initial phase of retraining not only fails to show improvements compared to full network

retraining, but instead, a slight performance regression is observed, with a longer training time required to achieve equally good results. Regarding the final performance achieved by the model, both techniques yield similar results, though a slight improvement is noticeable with full retraining. Therefore, in the case study, fully updating the network weights during the retraining phase yielded better results.

This page intentionally left blank.

Chapter 7

Conclusions

This thesis set out to achieve two main goals: firstly, to develop a deep reinforcement learning (DRL)-based controller that dynamically manages cloud resources in response to fluctuating demands while outperforming existing solutions; and secondly, to study the use of transfer learning techniques in this context, assessing their potential to improve the controller's initial performance and reduce the training time required to learn effective strategies.

To establish a testing environment, a variety of simulators were analyzed, culminating in the selection of CloudSim Plus due to its extensibility and robustness. This simulator allowed the development of the necessary customizations to accurately emulate realistic cloud environments, enabling the evaluation of the DRL controller under different configurations.

Further enhancements were made in order to introduce a "real mode", the simulated reality environment, featuring scaling and monitoring delays, as well as the incorporation of workloads reflective of real-world scenarios. These adjustments aim to mimic, to some extent, the potential discrepancies one might encounter when transitioning from a purely simulated environment to actual cloud dynamics, thus providing a differentiated yet still simulated platform that better reflects real-world conditions.

Reinforcement learning is a frequently employed technique for cloud elasticity. The integration of deep reinforcement learning, which leverages deep neural networks, provides a more effective approach to managing high-dimensional state spaces while simultaneously improving the controller's ability to model complex scenarios.

While RL has been applied to cloud elasticity, existing research recognizes initial performance limitations and employs various strategies to accelerate training. Nonetheless, these methods do not fully overcome the challenge. In contrast, transfer learning allows the RL controller to start from a significantly strengthened position, leveraging prior knowledge to bypass the initial learning curve in traditional setups.

The investigation included several transfer learning techniques, with a particular focus on Sim-to-Real transfer and Learning from Demonstrations. Sim-to-Real transfer was selected for its ability to pre-train the RL controller in a simulated environment that closely resembles real-world conditions. This process embeds the controller with a baseline policy that provides a strong starting point in the target environment. Learn from Demonstrations was chosen for its ability to effectively train the RL controller to imitate the policy demonstrated by an expert. In this instance, demonstrations from a threshold-based controller were used to provide the RL agent with a solid starting policy.

These strategies maximize the RL controller's initial performance; therefore, reducing the extensive training period typically required and ensuring that the controller can manage

dynamic workloads efficiently right from the outset.

The findings revealed that the RL controller significantly outperformed traditional threshold-based strategies, particularly in optimizing resource utilization while simultaneously maintaining satisfactory levels of performance. The controller exhibited a robust capacity to adapt to dynamic workloads, boosting both system performance and cost efficiency.

The initial performance of the controller, enhanced by transfer learning, demonstrates that both transfer techniques can significantly optimize the controller's efficacy from the outset. The Sim-to-Real technique showed superior performance, although it requires a realistic simulator to be effective. On the other hand, Learning from Demonstrations, while less performant and more challenging to configure, benefits from only requiring demonstrations from an existing controller to get started. Overall, the results confirm the effectiveness of using a reinforcement learning model to develop an efficient controller for cloud elasticity, with transfer learning proving to be a crucial strategy for achieving robust initial performance.

In summary, this thesis corroborates the potential of reinforcement learning to optimize cloud elasticity. The successful integration of RL with transfer learning techniques to improve initial performance validates the effectiveness of this approach and highlights its promise for future applications in cloud computing.

7.1 Future Research

This thesis establishes a foundation for several promising avenues of future research.

One key area involves enhancing the stability of the controller. Exploring techniques such as reducing the learning rate during training, with one particular method being reduce learning rate on plateau, where the rate is lowered when performance plateaus, or implementing permanent transitions in the replay memory could significantly improve stability. These adjustments aim to create a more reliable and consistent performance during the controller's operation.

Additionally, a promising direction for future research involves minimizing the number of scaling actions while maintaining high performance. Given the significant variability in the number of actions taken with minimal impact on performance, refining the controller to reduce unnecessary actions could further amplify the benefits already observed with RL. This could involve exploring advanced exploration strategies or adjusting the reward structure to better align with this objective. Moreover, reducing the number of actions may also contribute to improving the stability of the controller.

Another area that could be further explored is the improvement of the deep reinforcement learning controller's performance. One of the most compelling upgrades would be the integration of a predictive parameter for future demand, calculated using time series analysis. This addition would likely improve the controller's ability to preemptively scale resources in anticipation of demand fluctuations, further optimizing resource allocation.

A valuable avenue for future research lies in the further investigation, exploration, and testing of the Learning from Demonstrations technique. By examining the selection and impact of various configurable parameters, valuable insights could emerge, potentially leading to more effective policies for their selection. These improvements could, in turn, lead to more effective transfer learning, refining the technique and improving its overall applicability and performance in dynamic resource management.

One final direction for future work involves testing the developed controller and transfer learning techniques in a real-world environment. While this thesis relies on a simulator for all tests, including for Sim-to-Real transfer learning, transitioning to a real-world setting would provide valuable data on the controller's performance and the effectiveness of transfer learning under real operational conditions. This could help identify challenges not detected

in the simulations and help further refine the controller's robustness and applicability for dynamic cloud resource management in real-time scenarios.

These proposed research directions not only build upon the work presented in this thesis but also aim to advance the practical implementation of these technologies, bringing them closer to real-world applicability.

This page intentionally left blank.

Bibliography

[1] Mohammad Oqail Ahmad and Rafiqul Zaman Khan. Cloud computing modeling and simulation using cloudsim environment. *International Journal of Recent Technology and Engineering*, 8:5439–5445, 7 2019.

[2] Mehdi Ahmed-Nacer, Kunal Suri, Mohamed Sellami, and Walid Gaaloul. Simulation of configurable resource allocation for cloud-based business processes.

[3] Yahya al dhuraibi, Paraiso Fawaz, Nabil Djarallah, and Philippe Merle. Elasticity in cloud computing: State of the art and research challenges. *IEEE Transactions on Services Computing*, PP:1–1, 06 2017.

[4] Ahmed Ali-Eldin, Johan Tordsson, and Erik Elmroth. An adaptive hybrid elasticity controller for cloud infrastructures. In *2012 IEEE Network Operations and Management Symposium*, pages 204–212, 2012.

[5] Chafika Benzaïd, Tarik Taleb, Ashkan Sami, and Othmane Hireche. Fortisedos: A deep transfer learning-empowered economical denial of sustainability detection framework for cloud-native network slicing. *IEEE Transactions on Dependable and Secure Computing*, pages 1–18, 2023.

[6] Constantinos Bitsakos, Ioannis Konstantinou, and Nectarios Koziris. Derp: A deep reinforcement learning cloud system for elastic resource provisioning. *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, 2018-December:21–29, 12 2018.

[7] Tim Brys, Anna Harutyunyan, Halit Bener Suay, S. Chernova, Matthew E. Taylor, and Ann Nowé. Reinforcement learning from demonstration through shaping. In *International Joint Conference on Artificial Intelligence*, 2015.

[8] Rodrigo N. Calheiros, Rajiv Ranjan, Cesar A. F. De Rose, and Rajkumar Buyya. Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services. 2009.

[9] Xavier Dutreilh, Aurélien Moreau, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. From data center resource allocation to control theory and back. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 410–417, 2010.

[10] Maram Mohammed Falatah and Omar Abdullah Batarfi. Cloud scalability considerations. *International Journal of Computer Science Engineering Survey*, 5:37–47, 8 2014.

[11] Soumyajit Guin and Shalabh Bhatnagar. A policy gradient approach for finite horizon constrained markov decision processes. 10 2022.

[12] J. Fernando Hernandez-Garcia and Richard S. Sutton. Understanding multi-step deep reinforcement learning: A systematic study of the dqn target, 2019.

[13] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Deep q-learning from demonstrations, 2017.

[14] Ananya Harsh Jha, Saket Anand, Maneesh Singh, and Vsr Veeravasarapu. Disentangling factors of variation with cycle-consistent variational auto-encoders.

[15] Indu John, Aiswarya Sreekantan, and Shalabh Bhatnagar. Auto-scaling resources for cloud applications using reinforcement learning. *2019 Grace Hopper Celebration India, GHCI 2019*, 11 2019.

[16] Indu John, Aiswarya Sreekantan, and Shalabh Bhatnagar. Efficient adaptive resource provisioning for cloud applications using reinforcement learning. *Proceedings - 2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems, FAS*W 2019*, pages 271–272, 6 2019.

[17] Sara Kardani-Moghaddam, Rajkumar Buyya, and Kotagiri Ramamohanarao. Adrl: A hybrid anomaly-aware deep reinforcement learning-based resource scaling in clouds. *IEEE Transactions on Parallel and Distributed Systems*, 32:514–526, 3 2021.

[18] Navneet Kumar Rajpoot, Prabhdeep Singh, and Bhaskar Pant. Load balancing in cloud computing: A simulation-based evaluation. In *2023 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES)*, pages 564–568, 2023.

[19] Dawei Li, Chigozie Asikaburu, Boxiang Dong, Huan Zhou, and Sadoon Azizi. Towards optimal system deployment for edge computing: A preliminary study. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6, 2020.

[20] Han Li and Srikumar Venugopal. Using reinforcement learning for controlling an elastic web application hosting platform. *Proceedings of the 8th ACM International Conference on Autonomic Computing, ICAC 2011 and Co-located Workshops*, pages 205–208, 2011.

[21] Harold Lim, Shivnath Babu, and Jeffrey Chase. Automated control for elastic storage. pages 1–10, 06 2010.

[22] Chunhong Liu, Chuanchang Liu, Yanlei Shang, Shiping Chen, Bo Cheng, and Junliang Chen. An adaptive prediction approach based on workload pattern discrimination in the cloud. *Journal of Network and Computer Applications*, 80:35–44, 2017.

[23] MultiMedia LLC. Vmware cloud elasticity.

[24] Tianle Mai, Haipeng Yao, Ni Zhang, Wenji He, Dong Guo, and Mohsen Guizani. Transfer reinforcement learning aided distributed network slicing optimization in industrial iot. *IEEE Transactions on Industrial Informatics*, 18:4308–4316, 6 2022.

[25] Ahmad M. Nagib, Hatem Abou-Zeid, and Hossam S. Hassanein. Transfer learning-based accelerated deep reinforcement learning for 5g ran slicing. pages 249–256. IEEE, 10 2021.

[26] Ali Yadavar Nikravesh, Samuel A. Ajila, and Chung-Horng Lung. Cloud resource auto-scaling system based on hidden markov model (hmm). In *2014 IEEE International Conference on Semantic Computing*, pages 124–127, 2014.

[27] Alberto Núñez, Jose L. Vázquez-Poletti, Agustin C. Caminero, Gabriel G. Castañé, Jesus Carretero, and Ignacio M. Llorente. Icancloud: A flexible and scalable cloud infrastructure simulator. *Journal of Grid Computing*, 10:185–209, 3 2012.

[28] Oladosu Oyebisi Oladimeji, Dasola Oyeyiola, Olayanju Oladineji, and Pelumi Oyeyiola. A comprehensive survey on cloud computing simulators. *Scientific Journal of Informatics*, 8, 2021.

[29] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[30] Suraj Panwar, M. Rauthan, and Varun Barthwal. A systematic review on effective energy utilization management strategies in cloud data centers. *Journal of Cloud Computing*, 11, 12 2022.

[31] Valerio Persico, Domenico Grimaldi, Antonio Pescapè, Alessandro Salvi, and Stefania Santini. A fuzzy approach based on heterogeneous metrics for scaling out public clouds. *IEEE Transactions on Parallel and Distributed Systems*, 28(8):2117–2130, 2017.

[32] Chenhao Qu, Rodrigo N. Calheiros, and Rajkumar Buyya. Auto-scaling web applications in clouds: A taxonomy and survey. *BodyNets International Conference on Body Area Networks*, 9 2016.

[33] Sayed Ali Rad and Alireza Basiri. Brain drain optimization: A novel approach for task scheduling in the cloud computing. In *Proceedings - 2022 27th International Computer Conference, Computer Society of Iran, CSICC 2022*. Institute of Electrical and Electronics Engineers Inc., 2022.

[34] Navneet Rajpoot, Prabhdeep Singh, and Bhaskar Pant. Load balancing strategies for cloud computing: A simulation-based study. *Journal of Nano- and Electronic Physics*, 15:03023–1, 01 2023.

[35] Kamran Razavi, Mehran Salmani, Max Mühlhäuser, Boris Koldehofe, and Lin Wang. A tale of two scales: Reconciling horizontal and vertical scaling for inference serving systems, 2024.

[36] Fabiana Rossi, Valeria Cardellini, and Francesco Lo Presti. Self-adaptive threshold-based policy for microservices elasticity. *Proceedings - IEEE Computer Society's Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, MASCOTS*, 2020-November, 11 2020.

[37] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 500–507, 2011.

[38] Upendra Sharma, Prashant Shenoy, Sambit Sahu, and Anees Shaikh. A cost-aware elasticity provisioning system for the cloud. In *2011 31st International Conference on Distributed Computing Systems*, pages 559–570, 2011.

[39] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, New York, NY, USA, 2011. Association for Computing Machinery.

[40] Manoel C. Silva Filho, Raysa L. Oliveira, Claudio C. Monteiro, Pedro R. M. Inácio, and Mário M. Freire. Cloudsim plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 400–406, 2017.

[41] Parminder Singh, Pooja Gupta, Kiran Jyoti, and Anand Nayyar. Research on auto-scaling of web applications in cloud: Survey, trends and future directions. *Scalable Computing*, 20:399–432, 6 2019.

[42] Shahadat Hossain Sohag, Selina Sharmin, and Tanveer Ahmad. P-cloud-a cost efficient definitive cloud with user satisfaction using non-cloud resources. In *ICIET 2019 - 2nd International Conference on Innovation in Engineering and Technology*. Institute of Electrical and Electronics Engineers Inc., 12 2019.

[43] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[44] G. Tesauro, N.K. Jong, R. Das, and M.N. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *2006 IEEE International Conference on Autonomic Computing*, pages 65–73, 2006.

[45] Wenhong Tian, Minxian Xu, Aiguo Chen, Guozhong Li, Xinyang Wang, and Yu Chen. Open-source simulators for cloud computing: Comparative study and challenging issues. 6 2015.

[46] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

[47] Karl Weiss, Taghi M. Khoshgoftaar, and Ding Ding Wang. A survey of transfer learning. *Journal of Big Data*, 3, 12 2016.

[48] Bhathiya Wickremasinghe, Rodrigo N Calheiros, and Rajkumar Buyya. Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications.

[49] Keyu Wu, Min Wu, Jianfei Yang, Zhenghua Chen, Zhengguo Li, and Xiaoli Li. Deep reinforcement learning boosted partial domain adaptation. 2021.

[50] Rafael Xavier, Hendrik Moens, Jurgen Slowack, Wim Sandra, Steven Delputte, Bruno Volckaert, and Filip De Turck. Cloud resource allocation algorithms for elastic media collaboration flows. In *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, volume 0, pages 440–447. IEEE Computer Society, 7 2016.

[51] Rafael Xavier, Hendrik Moens, Bruno Volckaert, and Filip De Turck. Design and evaluation of elastic media resource allocation algorithms using cloudsim extensions. In *Proceedings of the 11th International Conference on Network and Service Management, CNSM 2015*, pages 318–326. Institute of Electrical and Electronics Engineers Inc., 12 2015.

[52] Jinwei Xing, Takashi Nagata, Kexin Chen, Xinyun Zou, Emre Neftci, and Jeffrey L. Krichmar. Domain adaptation in reinforcement learning via latent unified state representation. *35th AAAI Conference on Artificial Intelligence, AAAI 2021*, 12A:10452–10459, 2 2021.

[53] Hanyang Zhao, Wenpin Tang, and David D. Yao. Policy optimization for continuous reinforcement learning. 5 2023.

[54] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744, 2020.

[55] Guangyao Zhou, Wenhong Tian, and Rajkumar Buyya. Deep reinforcement learning-based methods for resource scheduling in cloud computing: A review and future directions. 5 2021.

[56] Zhuangdi Zhu, Kaixiang Lin, Anil K. Jain, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. 9 2020.

[57] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Senior Member, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. 2020.

List of Tables

This page intentionally left blank.

List of Figures

This is the last page.
Compiled Tuesday 15$^{\text{th}}$ July, 2025.