Investigating Multi-Agent Reinforcement Learning for adaptive cost-optimized storage allocations within cloud environments

MSc Research Project
Cloud Computing

Oritsejolomi Sillo
Student ID: 22129332

School of Computing
National College of Ireland

Supervisor: Aqeel Kazmi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Oritsejolomi Sillo
Student ID:	22129332
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Aqeel Kazmi
Submission Due Date:	12/08/2024
Project Title:	Investigating Multi-Agent Reinforcement Learning for adaptive cost-optimized storage allocations within cloud environments
Word Count:	XXX
Page Count:	23

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	15th September 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Investigating Multi-Agent Reinforcement Learning for adaptive cost-optimized storage allocations within cloud environments

Oritsejolomi Sillo
22129332

Abstract

This research seeks to explore the applicability of Multi Agent Reinforcement learning(MARL) in developing effective strategies to reduce data cost in cloud storage environments. Existing solutions have demonstrated the effectiveness of single agent reinforcement learning(RL) algorithms in reducing storage cost over traditional methods, however, there is limited research in how effective MARL algorithms will perform in this area, especially as it pertains to the scalability and fault tolerance of the system. In this study, we design and build a custom Gymnasium and cloudsim cloud storage environment in order to simulate a real world cloud storage solution, where MARL agents can be trained using Proximal Policy Optimization (PPO) algorithm. The study was able to demonstrates MARLs ability to significantly reduce cloud storage cost compared to baseline strategies while also being more fault tolerant than existing single RL solutions. Our study also highlights the potential for MARL to address the challenge of scalability in comparable systems.

1 Introduction

A key characteristic of most modern software solution deployment and delivery is cloud based computing. Gone are the days where software is package in some sort of media such as Floppy disk or CD to be installed by the user on a local machine. The most popular applications used today such as Facebook, Instagram, YouTube, are all delivered through cloud based applications with a front-end application installed on the user client.

These companies have complex and expensive data-centers that host their applications and is always available to the customer, easing the computational and maintenance burden associated with running the software on the users, making cloud computing indispensable for many consumer facing applications.

Most organisations , however, might not be inclined or capable of building and operate their own data centers for the purpose of software delivery and thus this is delegated to a third party providers such as amazon web service(AWS) and Microsoft Azure.

This has the benefit of being more energy and cost efficient as the cloud providers are able to pool computational resource and utilize the economics of scale to deliver services at a financial rate that may not be possible to achieve by an individual company.

This also frees the software companies to concentrate on the development and delivery of their software applications and not worry about computing hardware operations and other regulations associated with it.

However, despite the benefits of cloud computing based software delivery over self-hosted data center solutions, there have recently been an increase in the number of companies moving their computational needs to on-premises based solutions and a primary reason sighted for this has been the increasing cost associated with third-party cloud based solutions.

One complaint is that even though the prices of some computer components have decreased, this price has not been passed on to the consumers or has increase in some situations, making up a considerable 30 percent of organizations wasted expense.

In reality, to fully maximise the efficiency potential of cloud based computing, a user needs to be familiar with the various services offered by the provider, their use cases, has well as their weak and strong points.

This fact is complicated by the myriad of services offered by the cloud providers to suit different software needs along with different cost structures. These services are also constantly changing, with new services added and removed depending on market needs and prices updated constantly.
This introduces a high knowledge bar for a user to be able to utilize the right service and the right price structure based on their unique business needs.

Has an example, in 2023, Amazon Prime Video, a video streaming service owned by Amazon Inc, claimed that they were able to reduce there computational cost by 90 percent, simply by changing the computation service they were using to deliver the video streaming from a serverless service called AWS lambda to a serverful service called AWS EC2 and it associated architecture Kolny (2023).

This entails that selecting an ideal strategy for your cloud based solutions might be a more efficient way to maximize cost than on premises based solutions.
In this thesis, we will be focusing on optimizing strategies for cloud based storage solutions, which is a growing area of interest in the modern artificial intelligence and software development landscape. Cloud storage is an area of growth that has seen marked increase in recent years with its market estimated to reach 154.21 billion by 2026 Liu et al. (2023a). As with other cloud based solutions, cloud storage offers several advantages over on-premise based solutions such as scalability, high availability and its practically unlimited storage. This has resulted in cloud based storage being adopted by 46 percent of organizations Liu et al. (2023a).

2 Related Work

The increase in data hungry Artificial intelligence models and growth of cloud storage has consequentially increased the use of cloud storage and its associated cost, which in turn as created a need for effective cost management strategies. This need has resulted in the development of tiered storage solutions that have different price and performance characteristics. It is vital, that for companies to control spending that data is placed in tiers that best suit their characteristics otherwise, significant unnecessary cost can be incurred.

In this section will be looking at existing research in the area of cloud storage and various adaptive algorithms . We will explore the various methods employed by these research papers, identifying and highlighting the positive, negative and other potentially helpful aspect of their works, particularly as it relates to cloud storage placement strategies.

2.1 Optimization strategies currently employed in cloud storage environment

With cloud becoming more essential to modern software development and artificial intelligence training and deployment, the use of cloud based storage has proven critical for use to manage the vast amount of data generated by these services. This as led to storage cost being a considerable part of organizations cloud cost resulting in a requirement for cost effective ways to manage growing storage cost Liu et al. (2023a) specifies strategies already employed that require a user to manually configure and manage them. That is, the user set predefined rules that place certain types of data in specific storage tiers. Example of this is specifying a life cycle rule on Amazon S3 that lists actions that should be taken during the data objects life time, such as moving it slower, cheaper storage after its been inactive for 6 months AWS (2024).

While these methods do offer cost savings compared to placing all incoming data a specific tier, their effectiveness is limited to the knowledge of the user configuring them. The user has to stay abreast of changes to services and prices as they happen and update their configuration accordingly in order to maximize there cost efficiency.

This can introduce a knowledge barrier that can prove too difficult to scale, particularly when it concerns startups or businesses and professionals new to the cloud storage space.

Addressing this barrier with a system that is easy and straightforward to implement can prove to be an extremely valuable product.

2.2 Using Reinforcement Learning for building adaptive solutions

In machine learning, reinforcement learning can be described as a type of agent training where the agent, through series of trail and error, learns to perform tasks in an environment without being explicitly programmed to do so. The agent accomplishes this by receiving feedback from the environment in the form of values called rewards and observations, using this feedback, the agent tries to maximize its rewards by taking certain actions in the environment.

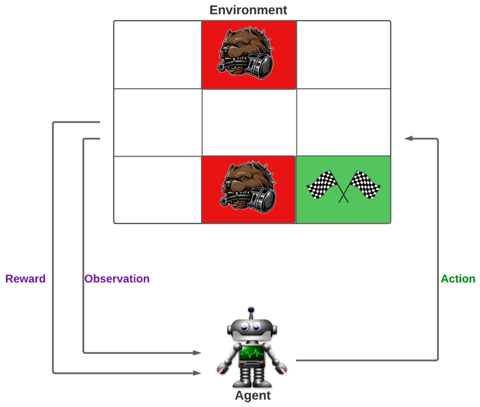

Figure 1: Agent – environment interaction

The interactions the agent has with its environment is crucial, whenever it performs an action in the environment, it changes the existing state of that environment, this new state is what is passed on to the agent as observations.

The state can be described as a representation of the environment in a moment of time, so when referring to the agent in the maze in fig 2.0, the state is the position (X,Y) of that agent in that environment.

This state can be an abstract, such as the coordinates of an airplane , its speed, elevation etc . In summary , the state is what feature of the environment is changing because of the various actions or inaction of the agent at each iteration of learning. How the state changes can be described as the dynamics of that environment.

A type of reinforcement learning, known as Deep Reinforcement learning(DRL) has been shown to be effective in learning adaptive strategies capable of operating in a dynamic cloud environment, particularly when historical data of that environment is such as the customer user patterns of a new software product. This learning ability is a feature DRL agents.

RLTiering is a DRL solution developed by Liu et al. (2023b) . it has demonstrated the capability to dynamically place data in appropriate hot and cold tiered storage solutions , lowering storage cost in the process. RLTiering works by using single agent DRL algorithms that interacts with the environments while having no knowledge of prior storage patterns. It decides on the appropriate storage tier to place the data as it arrives. RLTiering is able to significantly reduce costs by observing its environment state and adjusting its decision accordingly over time based on these observations.

The implemented RLTiering system utilizes an RL algorithm known as the Soft Actor Critic(SAC) in implementing its solution.

DRL effectiveness has also been demonstrated in other areas of cloud computing. Kaler and Toshniwal (2023) uses DRL to also dynamically place data in specific cloud storage environments but with a goal of reducing the latency of fetching that data at a later time. This method uses deep q network and soft actor critic algorithms, and operates by placing data as close to the user as possible based on the current cloud storage conditions.

These solutions first have to train an agent in the environment in which it will operate or a close replica. After training is complete, the agent can adapt to the dynamic cloud environment and different user requests types while significantly improving its performance when compared to standard methods.

Gao and Li (2023), further underscores the adaptability of DRL algorithms. They developed an optimization system that dynamically place services for cloud based gaming in Mobile edge computing systems, with the aim of reducing latency, this improves user experience and is achieved by placing gaming sessions as close to the users as possible.

DRL can also be used in predicting future workloads. Ahamed Zaakki (2023) was able to design a system that can anticipate workloads in federated cloud environments. It accomplishes this by using a predictive layer in addition to data placement techniques for cloud based services management. This then will be used to adjust required resources in anticipation of predicted demand .

The papers highlighted above demonstrate the capabilities of DRL algorithms to effectively adapt to various cloud and other environments. DRL has also shown significant advantages over methods centered around users that often require highly knowledgeable individuals to be effective. Existing research into DRL, However, focuses on single agent implementations.

Modern cloud services are required to be always available scalable and fault tolerant

Rehman et al. (2022) , and as such single RL solutions can be a limitation. This is because a single agent introduces a single point of failure for a potential service.
Our study as thus far highlighted the promising potential of a MARL system for an adaptive solution by adding features inherent to multi-agent systems which justifies our research question, Investigating Multi-Agent Reinforcement Learning for adaptive cost-optimized storage allocations within cloud environment. We can use MARL to address the limitations of centralized RL by distributing decision-making. This type of framework could add scalability and fault tolerance, scalability to adaptive data placement strategies in existing cloud solutions.

2.3 Multi-Agent Reinforcement learning Systems

MARL is an extension of reinforcement learning single agents systems into environments that contains multi agents working in tandem. Single agents also only have to take thier own actions on the environment into account Ikeda and Shibuya (2022), MARL, on the other hand, involves several RL agents whose actions in the a given environment influences not just their own results but the outcomes of all other agents operating in the environment.

This dynamic nature of marl algorithms introduces the issue of non-stationarity that arises from the action of several agents acting simultaneously in a given environment. Because of this, the future rewards of any given agent is determined by the combined actions of multiple agents resulting in the computational complexity increasing, as highlighted by Ikeda and Shibuya (2022).

Regardless of this issue, the benefits of MARL algorithms far outweigh the problems it has, such as its ability to solve problems that standard RL may find almost impossible . This alone make the search of solutions for these MARL problem beneficial.
Gronauer and Diepold (2022) in this survey shows that the implementation of MARL algorithms for problems that are beyond the capabilities of standard RL algorithms to solve, showcases the advantages of MARL over RL in scenarios where competition among agents along with transfer learning are essential to getting an ideal outcome. These features of MARL algorithms also give them innate fault tolerant abilities.

Li et al. (2022) showcases MARls ability to significantly improve system performance in ever changing environments where single agent centralized methods could suffer, it particularly focuses on the internet and other related technologies that require a significant amount of security and resource management .

Wong et al. (2022) in their study gives an overview the various benefits MARL has over single agents RL algorithms. it also highlights the challenges, including partial observavbly, where an agent is able to observe only part of the environment, and prescribe various strategies to overcome them. They mentioned, among others, centralized training with decentralized execution (CTDE), utilizing an efficient communication protocol, and incorporating strategies from psychology and sociology to emulate collaboration amongst agents.

There are some proposed solution to the shortcomings marl algorithms, the ACE methodology Li et al. (2023), addresses the issue of non-stationarity by changing the MARL problem into a sequential single agent framework. This solution is effective but introduces another issue of scalability, because utilizing just one agent to make the final decision could limit the practical scalability of the system.
MARL have been also used in fuel cell hybrid electric vehicle (FCHEV) management

systems. The employ techniques such as MADDPG to optimize their performance Xiao et al. (2023). This system demonstrated noticable energy savings when compared to standard methods. These types of MARL algorithms have not been used in the area of cloud storage, and as such, the scalability of CTDE and MADDPG techniques requires more research.

Another MARL system tries to address communication lag in multi-agent systems that require cooperation between agents. Ikeda and Shibuya (2022) incorporate past communication into the learning process of agents in order to improve their decision making when there is lag in communication with other agents or even an outage. In this scenario, agents can use previous communication patterns to predict the actions of other agents and adjust their own strategies. This can help to maintain system stability and performance even when communication is not ideal.

We can also implement MARL systems for data redundancy. If an agent fails during normal operation, another agent should still be able to access the agents data and carry on its tasks. Combined all these strategies create a MARL based approach to data placement strategies that could potentially deliver fault tolerance and improved scalability when compared against single agent RL methods.

These studies across different industries highlights both the advantages and challenges of MARL. It has the potential to add fault tolerance, scalability while keeping the adaptability of centralized RL makes MARL a promising area to explore. My research seeks to address the existing limitations of centralized training methods and the need for fault tolerance and scalability in cloud storage systems.

2.4 Scalability and Fault Tolerance in Distributed Systems

In current production ready, customer facing cloud solutions, scalability, availability and fault tolerance has become an essential and expected feature Kumari and Kaur (2021). While the adaptive and autonomous system such as single reinforcement learning algorithms has been sufficiently proven, the possibility for them to be applied practically in production settings is doubtful because of the current implementation limitations of the technology.

MARL algorithms, because of their distributed nature, have strong potential for high scalability and fault tolerance. This is made possible because distributing decision making across several dispersed agents enables them to be resilient . However, MARL's full potential, however, is limited by the complexities that can arise from inter-agent coordination among an increasing number of agents.

Methods such as CTDE have been developed to address this shortcomings but the reliance of these algorithms on a centralized training phase introduces potential bottlenecks. Transfer learning is a possible solution, Zhou et al. (2023) in their paper demonstrates a system that could potentially address the challenge of scalability simply by the breaking down MARL structures. This solution, however, has not been applied to cloud storage environments its use in the process of optimization is a promising area that should be investigated.

Our analysis of these papers highlights the need for further research into solving the remaining bottlenecks of MARL-based systems, their scalability and fault tolerant potential is off particular importance in real-world cloud storage environments. The aim of our thesis is to contribute to the to this field by researching transfer learning and other methods within the MARL framework. We aim to develop an adaptive data placement strategy that could enable the implementations of MARL systems for practical cloud storage optimization.

3 Methodology

In this section we will outline the methodology that is employed in this study, from the different environment setup, MARL algorithm and the various training and evaluation processes.

The methodology used incorporates a simulated cloud environment in OpenAi Gymnasium , training our MARL agents in a Jupyter Notebook gym environment and evaluating their performance in the gym environment and further validating our results through a REST API that communicates with a simulated cloud environment using CloudSim.

3.1 Custom OpenAi Gymnasium Environment

In simulating the environment, we made use of a custom cloud environment which we developed using the OpenAi Gymnasium framework. Through several iterations we were able to settle on an environment that closely models cloud based storage systems like AWS S3 , were MARL agents can interact with to learn data placement strategies.
Our gymnasium environment is made up of several components, key among them are:

- ***Action Space*** : This is the actions that each agent can perform in the environment,in this case , they choose between place data in hot(0) or cold(1) storage tiers.
- ***Observation Space*** : This represents the state of the environment and is assessed by the agents after it performs an action in order to evaluate the effect of its action and learn from it. It is a continuous space that makes use of normalized features such as data locations(0 for hot, 1 for cold), data size(normalized between 0 and 1), agents current steps(normalized between 0 and 1), access patterns(normalized between 0 and 1) and previous cost(normalized and clipped between 0 and 1)
- ***Rewards*** : The reward is used as an incentive to steer the agent towards making beneficial decisions, in this case that is saving cost overtime. As such, the agent receives a negative reward when it incurs additional cost and a positive reward when its actions leads to cost savings.

This simulated gymnasium environment is setup to imitate real would cloud storage environments. In the real world, data access patterns changes over times and as such, out agents have to adapt their strategies. Each of the agents acts independently in determining where to place incoming data based on its observation of the environment states. The actions of all the agents in the environment affects the storage cost, which is made up of storage and access cost.

Our MARL agents are developed and trained in a jupyter notebook based environment. We make use of the Ray Library for training and PyTorch for building our neural network models. The setup used for this training are:

- ***System Hardware*** : The training and simulation of our MARL agents was perform on a PC system with a 16 core, 24 threads Intel Core i7 13700KF processor, 32GB of 6400MTS DDR5 memory and an Nvidia GeForce RTX 3090 GPU.
- ***Software libraries used*** : We made use of PyTorch to create our custom neural networks, Ray library for reinforcement learning, Optuna for hyper parameter tuning and OpenAi Gymnasium for the custom environment used to train our agents.

3.2 CloudSim Environment

To in addition to the gymnasium environment that is used to train and evaluate the MARL agents, we designed and implemented a CloudSim environment to evaluate trained agents in an environment that is much more representative of real world environment than what we have in gymnasium.
CloudSim, an open source framework for simulating cloud computing services and infrastructure, presents us with several benefits, such as enabling us to deploy our model in cloud like environments without incurring the cost of an actual cloud deployment. It also makes it possible to run the same experiments multiple times to study its result, i feat that might not be possible in the open cloud environment.

Our cloudsim environment consist of two datacenter types that represents the hot and cold storage tiers.
Each of these are configured with different cost, hardware capabilities and other resources that simulates the characteristics of a cloud storage solution.
In our simulation, virtual machines(VMs) are created for each agents, which are then used to simulate the placement tasks.

We make use of a Spring Boot REST API, which was developed to enable the marl agents communicate with the cloudsim environment and perform tasks. This sends various requests by the agents to the cloudsim environment which then sends back results that include the state of the environment.

3.3 Multi Agent Reinforcement Learning(MARL) Algorithms

We made use of proximal policy optimization (PPO) algorithm to train our MARL agents because of its stability in training and ability to handle continuous action spaces. The PPO algorithm makes use a policy gradient method which makes use of on-policy algorithm that uses a stochastic policy.
PPO is made up of two neural networks which are:

- ***The Actor or Policy Network*** : This is the actor who is responsible for the actions of the agent.
- ***Critic or Value Network*** : The critic network attempts to predict the future occurence from the current state of the environment.

3.4 Training Process

Our agents are trained to reduce cost by making efficient data placement decisions that is based on the observations of the environment states. The process of training the agents involve the following:

- ***Environment Initialization*** :In this step the environment is reset/started and the initial states are sent to the agents.
- ***State observations*** :The agents then interact with the environment and collect observations in the form of sequence of states, rewards.
- ***Update Policy*** : Based on the observations the agents update their policy and value network through gradient ascent on the clipped objective.
- ***Training Loop*** : The training loop involves the agent taking actions in the environment based on their policy and receive either positive or negative rewards.

3.5 Hyperparameter Tuning

Initial training was conducted using time-limited AWS Sagemaker resources until more suitable training hardware could be appropriated. These initial training were limited in scope and were used to established the viability of different algorithms and configuration in training the agents and also establishing a baseline on which to evaluate the agents. This limited our training run to 100 iterations or less. Between training runs, we manually adjusted the configurations and tried different algorithms based on the results we were getting. This process was very time, resource intensive and the results were not showing satisfying improvements.

We then made the decision to employed tools could find the best configurations for out agents more efficiently and effectively.

3.6 Tuning Framework

To efficiently train our agents, we made use of hyperparameter tuning tools. The tools used were Ray Tune and OptunaSearch. This enabled us to find the ideal configurations for training our PPO algorithm without wasting time and compute resources .

The process of hyperparameter tuning involves first defining key hyperparameters we want to tune, such as the learning rate, value function clip parameters, lamgda(GAE parameter), clip parameters entropy coefficient, gamma etc.

We then define a search space, which is a range of values we want the tuning tools look into, of each hyperparameter.

Finally, we used OptunaSearch to explore our specified search space and find our ideal hyperparameter configurations using trail and error.

3.7 Scheduler

We made use of the ASHAScheduler to further optimize our hyperparameter tuning process. Ashascheduler does this by scheduling and pruning trials based on their performance, this way our time and computational resource are efficiently managed.

This further refines our tuning process, making finding ideal hyperparameters faster and more effective.

3.8 Evaluation Metrics

After training, we need to evaluate the agents to see if they are performing desirably. Our agents are evaluate using several metrics, key among them is the mean total reward, mean episode length and agent-specific rewards.

The mean total reward is the average reward obtained by our agents over the course of several training episode. This indicates how effective they were in reducing storage cost.

The mean episode length is used to indicate the stability of the agents training process using the average length of each episode. The longer the episode the more stable the training.

Lastly the agent specific reward is the performance of each agent and serves as an insight into each agents efficiency and consistency.

3.9 Baseline Strategies

To evaluate the performance of our agents we compared their recorded performance against baselines strategies. The baselines are as follows:

- ***Always Hot*** :Data is always placed in the hot storage tier.
- ***Always Cold*** : Data is always placed in the cold storage tier.
- ***Random Strategy*** : Randomly choosing between the hot and cold storage tiers.

This enables us to evaluate if our agents are learning effective strategies.

3.10 Scalability and Fault Tolerance

3.11 Scalability

As it is one of the primary goal of this research, the scalability of our MARL system is also evaluated. This is done by changing the number of agents in the environment and observing the impact these changes have on the mean reward and episode length.

We want to see if the performance of each agent remains consistent or even improve as the size of the system grows.

3.12 Fault Tolerance

We assess the fault tolerance of the system by observing its performance when an agent fails. If the system continues operation as normal then we can determine that it is fault tolerant.

For this project each agent is trained using transfer learning and are independent of each other, this is done to further enhance the fault tolerance of the system.

3.13 Details of implementation

Our custom environment is registered for training using RLlib framework, this is to facilitate and simplify the training and evaluation process.

For the configuration we specified the environment, the number of agents , the maximum number of steps the agents can take in the environment per episode, our neural network architecture along with other parameters. This is used to initialize and start the PPO training process.
As mentioned earlier the agents are evaluated against stated baseline strategies. We run several episodes and calculate the rewards of each strategy, which is then compared against our marl system in order to assess its effectiveness.
After the gymnasium evaluation, the agents are then evaluated in the CloudSim environment to further substantiate their performance.

4 Design Specification

In this section we will be presenting the techniques, frameworks and architectures underlying the implementation of the MARL system used to research data placement optimization strategies in cloud environments. We also identify the requirement of the research.

4.1 Architecture

4.2 Custom Gymnasium Environment

We made use of a custom Gymnasium environment to simulate a real world cloud based data storage solution with hot and cold tiers. Our environment closely models the highly dynamic world of cloud data placement e.g data storage and access cost, different data sizes and access patterns.

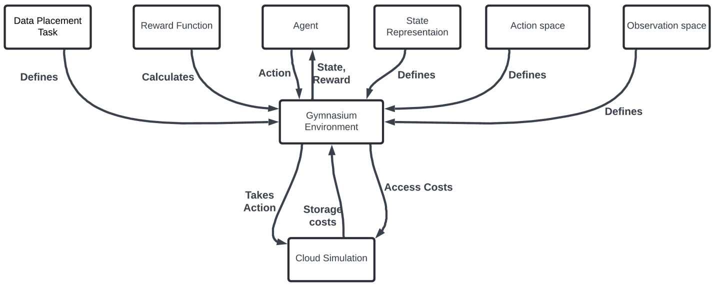

Figure 2: Gymnasium Environment workflow

4.3 CloudSim Evaluation environment

The cloudsim environment is used to evaluate the models trained in the gymnasium environments. Figure 3 shows the architecture diagram used as well as the workflow of the system design.

4.4 Requirements

- ***Scalability*** : A key part of the research is to see if a marl system can scale with customer requirements, which entails adding and removing agents depending on use.

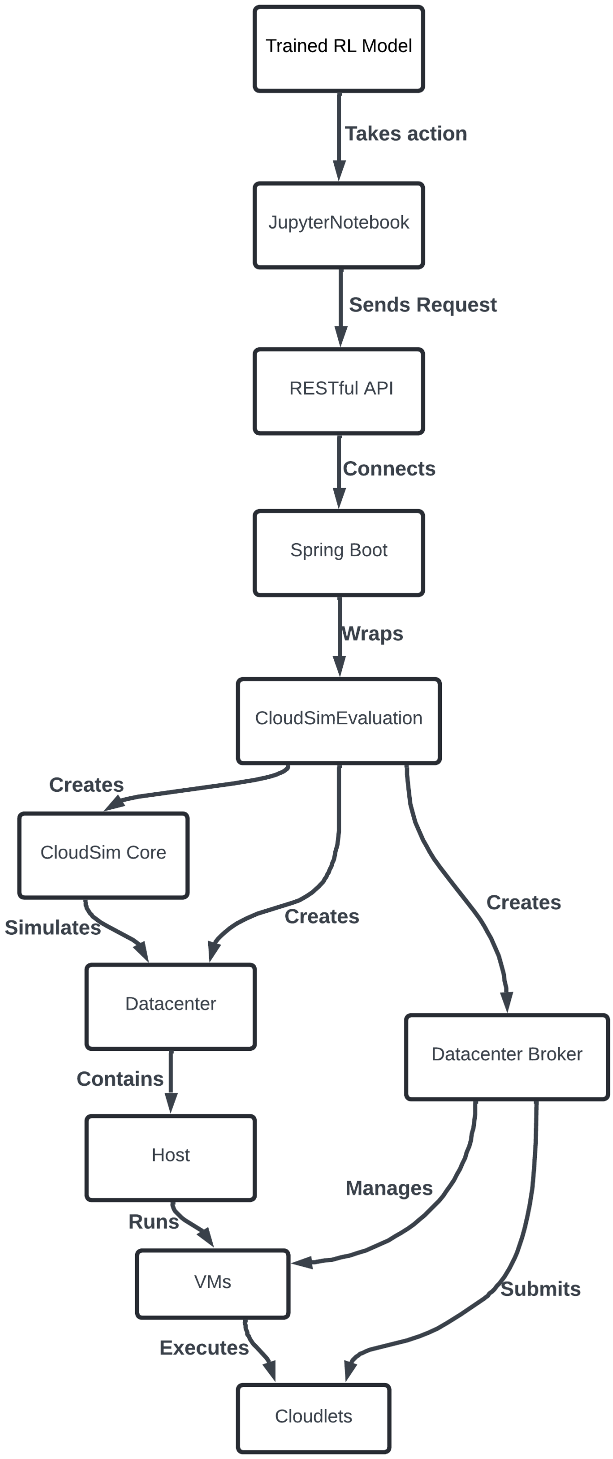

Figure 3: CloudSim environment workflow

- ***Fault Tolerance*** : The availability of the system is critical in a production environment and the system should be able to handle the failures of agents while it maintains overall performance of the system.
- ***Cost Efficiency*** : The system should be able to minimize cost by finding optimal data placement strategies.

5 Implementation

In this section we will be describing the various components of our implementation, these are our Gymnasium custom environment and the training of MARL agents in it and the evaluation of the trained agent in both Gymnasium and cloudsim environments.

5.1 Custom Gymnasium Environment

Our custom cloud environment , "CloudSimEnv" was created to simulate a real world cloud storage system that enables the placement of data by agents of our MARL system. We created the environment using OpenAI Gymnasium Framework and is used to train our agents.

Algorithm 1 CloudSimEnv Initialization and Interaction

```
procedure InitializeEnvironment
    Set action : This can be either hot or cold e
    Set observation space :
        Data locations
        Data sizes
        Access patterns
        Previous costs
end procedure
procedure ResetEnvironment
    Initialize environment:
        Assign random data locations to agents
        Generate random data sizes
        Set access patterns
    Return initial state
end procedure
procedure StepFunction(action_dict)
    for each agent in action_dict do
        Retrieve action (hot or cold )
        Calculate storage and access costs depending on action
        Update agent total cost and state
        Assign reward based depending on cost savings or expenditures
    end for
    Return updated state, reward, and end status
end procedure
```

5.2 Hyperparameter Tuning

Hyperparameter tuning is used to efficiently identify ideal configurations for training our PPO algorithm. We specify a range of values of each hyperparameter to be explored using ray tune and optunasearch.

Algorithm 2 Hyperparameter Tuning using raytune and OptunaSearch

```
procedure DEFINE THE SEARCH SPACE
    Set range for:
        Learning rate
        Gamma
        Entropy coefficient
        Clip parameter
end procedure
procedure RUN SEARCH
    Configure Raytune with the ASHAScheduler
    Schedule and prune underperforming episodes
    Iterate over search space find ideal configurations
    Evaluate the performance to select the best configuration
    Store the best hyperparameters
end procedure
```

5.3 Train agents using best hyperparameters

Having found the best hyperparameters using the tuning tools, we configure the PPO trainer with these values and perform training for a set number of iterations. During the training the models performance will be periodically evaluated to confirm that the results are ideal before saving the best models.

Algorithm 3 PPO Training Process

```
procedure INITIALIZE PPO
    Set up policy
    configure hyperparameters
end procedure
procedure TRAININGLOOP(num_iterations)
    for each iteration in num_iterations do
        Collect state data by performing actions in the environment
        Update policy using collected data:
            Adjust policy to maximize rewards
        if performance threshold is reached then
            Stop training
        end if
    end for
end procedure
procedure SAVEMODEL
    Save the trained model
end procedure
```

5.4 CloudSim Environment

After we train the agents, we perform evaluations to determine thier performance. We use cloudsim in order to evaluate the agents in more realistic cloud environments that can validate if the agent makes adequate cost saving decisions.

Algorithm 4 CloudSim Evaluation Process

```
procedure Initialize CloudSim(num_agents)
    Create hot and cold storage
    Set up VMs for each agent
end procedure
procedure EvaluateAction(agent_id, use_hot_storage, data_size, access_pattern)
    Calculate storage and access costs
    Simulate tasks in cloudsim
    calculate Reward
    Return reward
end procedure
procedure end Simulation
    starts and stops cloudsim
    Calculate and return total cost
    calculate and return average execution time
end procedure
procedure RESTAPICommunication
    receive evaluation requests from jupyternotebook
    execute evaluation in CloudSim
    Return results to Notebook
end procedure
```

6 Evaluation

In this section we will be providing a comprehensive analysis of the results and the main findings of our study. The experiments we conducted are designed to evaluate the scalability and fault tolerance of a MARL system for data placement in a simulated cloud storage environment. We will include the results from both before and after hyperparameter tuning.

For this experiment we wanted to evaluate the performance of the MARL agents and establish a baseline performance. This will be used as a benchmark for improvement

6.1 Methodology

We trained the agent without any tuning using default and random configurations and we evaluate them on their data placement performance

6.2 Results

- ***Mean Total Reward*** :-7084.69 ± 1229.28
- ***Mean Episode Length*** :1000.00

- ***Agent-Specific Results***
    - ***Agent 0: Mean Reward:*** :-1250.33 ± 711.63
    - ***Agent 1: Mean Reward:*** :-1263.70 ± 881.13
    - ***Agent 2: Mean Reward:*** :-1330.86 ± 843.07
    - ***Agent 3: Mean Reward:*** :-1621.33 ± 611.72
    - ***Agent 4: Mean Reward:*** :-1618.47 ± 813.55

Our initial training showed poor performance, with high negative rewards and large variability. This indicates the need for substantial configuration modifications which we perform as shown in the following sections.

6.3 Experiment / Case Study 2: Manual Configuration tuning

With this experiment, we aimed to make extensive alterations to both the environment and implement and test a custom neural network model configuration

6.4 Methodology

We attempted to manually improve the learning efficiency of our agents. We trained for 500 iterations with the modified configurations. The question of if marl algorithms are able to learn to optimize cost is still answered but with trying different setups including different algorithm types we sought to get an indication of the answer.

6.5 Results

- ***Mean Total Reward*** :-2057.17 ± 1144.96
- ***Mean Episode Length*** :1000.00
- ***Agent-Specific Results***
    - ***Agent 0: Mean Reward:*** :-1064.53 ± 666.76
    - ***Agent 1: Mean Reward:*** :-992.65 ± 629.85

Above is one of the results of these configurations. It shows significant improvements compared to our previous training sessions with the agents reward getting closer to zero. This shows signs that the marl agents were learning, however, the agents were still performing poorly, and manually hyperparameter and configuration tuning is time consuming and compute inefficient.

6.6 Experiment / Case Study 3: Hyperparameter Tuning

In this experiment we set out to tune our hyperparameters of the PPO algorithm to see if the performance of the agents could be improved in such a way that enabled them to learn effective strategies.

6.7 Methodology

We used Ray Tune and OptunaSearch to search for the best configurations. Those tuned configurations were then used to train the agents for 1000 iterations

6.8 Results

- ***Mean Reward*** : -0.6681 ± 0.0
- ***Number of Episodes*** : 32
- ***Agent-Specific Results***
  - ***Agent 0: Mean Reward:*** :-0.0261
  - ***Agent 1: Mean Reward:*** :-0.7104

Implementing hyperparameter tuned configurations resulted in significantly better performance improvements of the agents with their mean reward now approaching zero , down from negative seven thousand initially. There is now a clear sign of learning, but we need to perform evaluations to highlight it. This highlights just how effective the hyperparameter tuning process implemented was in training our agents for data placement.

6.9 Experiment / Case Study 4: Evaluation

With this experiment , we compare the performance of our best trained agents against baseline strategies, which include always hot, always cold and random placement.

6.10 Methodology

The agents were evaluated over multiple episodes after which the rewards were calculated and compared.

6.11 Results

- ***Trained Model: Mean Reward:*** : -1.3087 ± 0.4921
- ***Always Hot: Mean Reward*** : -2.4973 ± 0.9317
- ***Always Cold: Mean Reward*** : -1.2062 ± 0.4957
- ***Random: Mean Reward*** : -1.7501 ± 0.7902

Our trained model performs well, outperforming the baseline strategies. This shows that the agents are able to effectively optimize cost in data placement. Even though the always cold strategy performs better than the always hot and always random strategies, highlighting that cold storage tends to be generally more cost effective than the other tiers, our marl agents performance entails that an adaptive approach gives the best results when it comes to balancing both cost and performance.

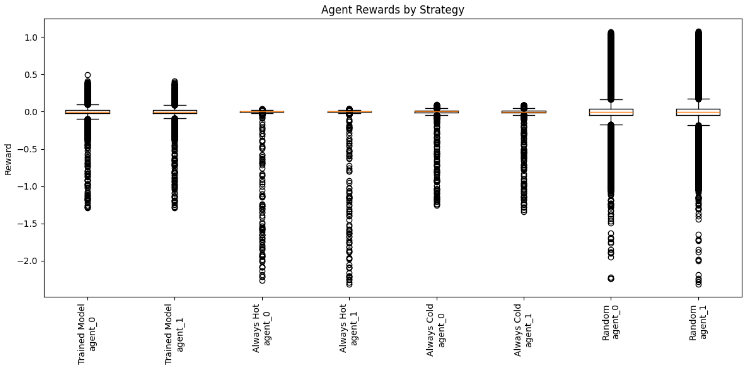

Figure 4: Agent Rewards by Strategy

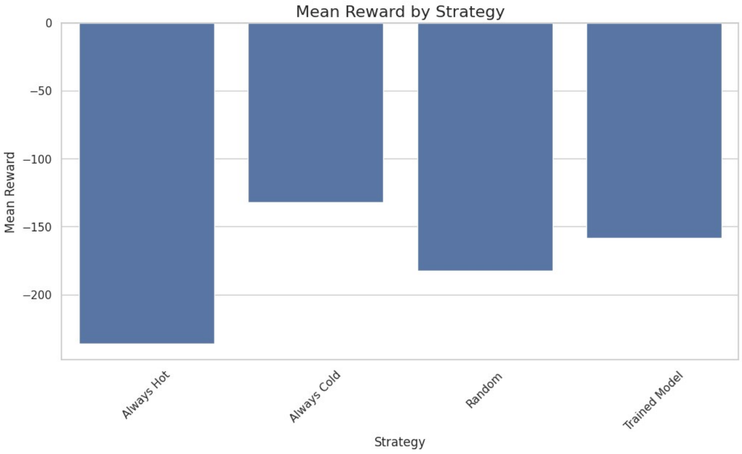

Figure 5: Mean Rewards by Strategy

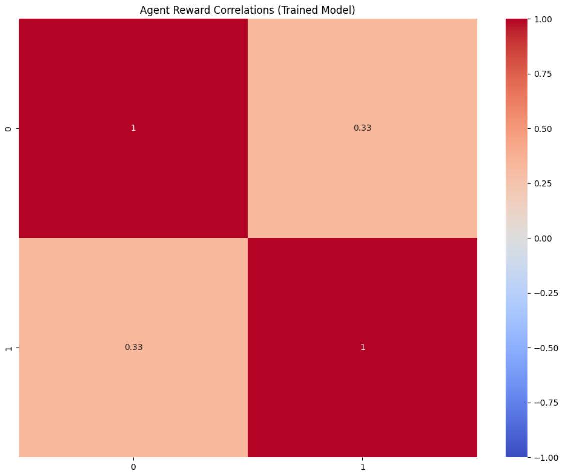

Figure 6: Agent Correlations Heatmap

6.12 Experiment / Case Study 5: Scalability Testing

After highlighting the ability of our marl agents to outperform baseline strategies in in cloud storage cost optimization, we want to test the scalability of our MARL system, evaluating the systems performance has the number of agent in the environment changes. Test were conducted in both the gymnasium and cloudsim environment.

6.13 Methodology

We evaluated the marl system with different numbers of agents , capturing the rewards and episode length of each run.

6.14 Results

- ***Number of Agents: 1***
  - ***Mean Reward:*** :-0.6185 ± 0.3604
  - ***Mean Episode Length:*** :100.00 ± 0.00
- ***Number of Agents: 2***
  - ***Mean Reward:*** :-1.2765 ± 0.5473
  - ***Mean Episode Length:*** :100.00 ± 0.00
- ***Number of Agents: 4***
  - ***Mean Reward:*** :-2.5765 ± 0.7752
  - ***Mean Episode Length:*** :100.00 ± 0.00
- ***Number of Agents: 8***
  - ***Mean Reward:*** :-5.3186 ± 1.0042
  - ***Mean Episode Length:*** :100.00 ± 0.00
- ***Number of Agents: 16***
  - ***Mean Reward:*** :-10.3857 ± 1.5525
  - ***Mean Episode Length:*** :100.00 ± 0.00

The experiment shows that as the number of agents increases , the mean reward(cost) doubles. The standard deviation is also shown to increase as the number of agent increase, indicating growing variability in the system that can indicate that inter-agent interactions presents a significant challenge it.However, the consistent number of steps shows that, even with this increase complexity, the system can handle an increasing number of agents and still remain stable.

The linear scaling does not necessarily shows poor scaling, because an increase in agent results in an increase in data volume which drives cost increase. More experiments need to be conducted to see if the per agent performance remains consistent as the system scales.

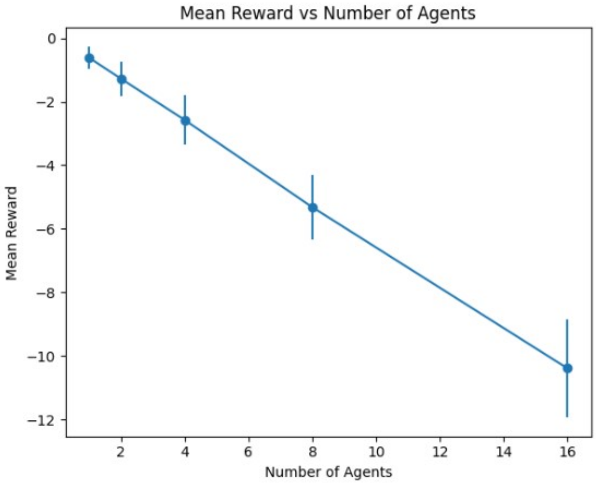

Figure 7: Scalability

7 Experiment / Case Study 6: Fault Tolerance

For this test we want to evaluate how the system performs when agents fail.
To demonstrate the fault tolerant capabilities of our system we run an experiment in which we simulate agent failures and observe the impact they have on the mean reward and other outputs.

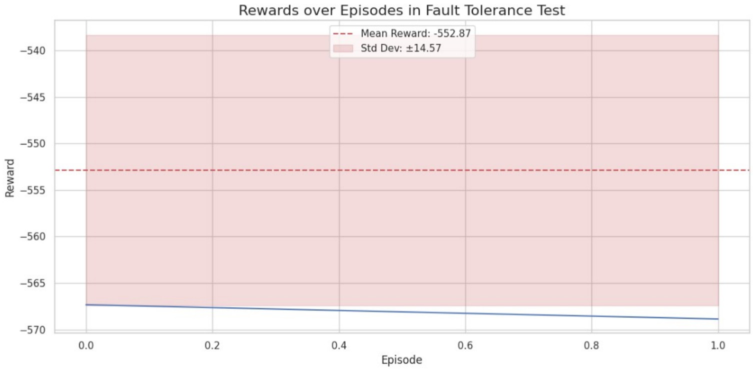

Figure 8: CloudSim Fault tolerance

The result of the experiment shows that our results remain relatively the same over multiple episodes, even when there are random agent failures.
This shows that there is no sizable hit to performance when there is agent disruptions as the system continues to maintain a consistent level of performance as seen in the result.

7.1 Discussion

Through a series of comprehensive experiments, we were able to evaluate the performance of our marl system and see how they perform with data placement optimization in cloud storage environments. Each of our experiments is design to analyze different areas of the marl system.
We started with training agents in a simulated cloud environment using default and random configurations which showed that our marl system was indeed functional but

performing poorly, manual hyperparameter and configuration tuning resulted in better than default performance but still poor showings.

Using hyperparameter tuning we were able to significantly enhance the performance of our marl agents, bringing their mean rewards close to zero. This shows that the agents are capable of learning strategies and demonstrates the benefits of the iterative process to training MARL agents employed in this research.

Comparison with baseline strategies showed that using marl agents for data placements in cloud storage solutions offered significant benefits over always placing data in hot or cold tiers and randomly placements.

Key to this research and the use of MARL agents is fault tolerance and scalability. These are critically important features of any cloud based solution that is required to be always available to the customer. Our MARL system demonstrated their excellent fault tolerance capabilities. Our system was able to continue performing optimally even during agent failure.

Scalability was tested by progressively adding agents to the system to observe the impact. The results demonstrated linear scalability, where adding agents increased the negative rewards but the system remained stable. While first assessment might point to scalability issues, the linear increase was because of the system design which aggregated rewards received across all agents in the system. This means that the increase in negative rewards was a result of the increase in cost and workload of each added agent and not necessarily the poor scalability of the system.

The fact that the system remain stable as agents were added is promising.

Per agent performance needs to be investigated to establish how scalable the system is.

Our findings closely aligns with previous research that highlights the potential of marl agents to effectively handle complex and dynamic environments.

Studies by Liu et al. (2023b) and Gao and Li (2023) also showed how effective RL algorithms were in significantly optimizing cloud based resources, though those research were based on single agents RL and a different cloud resource.

Our research have built on top of those studies by demonstrating the effectiveness of MARL algorithms in optimizing data placement strategies while also highlighting the benefits such a system brings.

8 Conclusion and Future Work

This research attempt to answer the following question : ”How can Multi-Agent Reinforcement learning be applied to optimize data placement in a cloud storage environment to reduce storage cost”. Our objective were to design and build a MARL system that can cost optimize data placement strategies while also ensure the scalability and fault tolerance of the system.

To answer the research question and objectives, we performed several tasks. First, we developed a custom cloud storage environment in OpenAi Gymnasium in order to simulate a real world cloud storage system.

We then implemented a MARL system using the PPO algorithm and RLlib framework. Using Hyperparameter tuning , we were able to efficiently optimize the performance of our agents in the simulated environment.

After we trained agents with acceptable performances, we performed thorough evaluation against a set of baselines that tested their cost effectiveness, scalability and fault tolerance in a cloudsim based environement.

We were able to successfully demonstrate that the MARL system can effectively develop strategies to optimize data placement in cloud storage environment. Our agents were able to outperform baseline strategies while displaying adaptability and fault tolerance in managing costs. Our hyperparameter tuning significantly improve the performance of the marl system.

However, the study also highlighted possible challenges as it relates to the scalability of the system. The main issue with scalability is that the system is constrained by increased complexity as the number of agents in the system increase. more research into per-agent evaluation and a more detailed test environment is required to fully understand the scalability of the system.

Evaluation of the agents was also performed in simulated environments in both OpenAI Gymnasium and a validating run in a separate cloudsim environment, while our best effort was made to model real-world cloud storage environments, the limited time we had meant that we were not able to capture all the nuances of cloud storage solution.

Future research should look into coordination's solution in marl system in order to fully explore the issue of scalability in marl systems. Communication protocols that can be used to facilitate inter-agent coordination while still maintaining fault tolerance should be investigated. More detailed environments with more tiers of storage is also need to fully investigate the potential MARL systems can have in cloud storage environments.

In conclusion, this study has successfully demonstrated the potential of marl system in developing optimization strategies for data placement in cloud storage solutions. While we identify a few challenges, our results and outlined future directions shows that their is huge potential for the practical applicability of MARL solutions in production environments.

References

Ahamed Zaakki, K. M. (2023). Deep reinforcement learning for workload prediction in federated cloud environments, *Springer Nature* .

AWS (2024). Setting a lifecycle configuration on a bucket.
URL: *https://docs.aws.amazon.com/AmazonS3/latest/userguide/how-to-set-lifecycle-configuration-intro.html*

Gao, Y. and Li, Z. (2023). Deep reinforcement learning based rendering service placement for cloud gaming in mobile edge computing systems, *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 502–511.

Gronauer, S. and Diepold, K. (2022). Multi-agent deep reinforcement learning: a survey, *Springer Nature* .

Ikeda, T. and Shibuya, T. (2022). Centralized training with decentralized execution reinforcement learning for cooperative multi-agent systems with communication delay, *2022 61st Annual Conference of the Society of Instrument and Control Engineers (SICE)*, pp. 135–140.

Kaler, R. and Toshniwal, D. (2023). Deep reinforcement learning based data placement optimization in data center networks, *2023 IEEE International Conference on Big Data (BigData)*, pp. 2293–2302.

Kolny, M. (2023). Scaling up the prime video audio/video monitoring service and reducing costs by 90
URL: *https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90*

Kumari, P. and Kaur, P. (2021). A survey of fault tolerance in cloud computing., *Science Direct* pp. 1–2.

Li, C., Liu, J., Zhang, Y., Wei, Y., Niu, Y., Yang, Y., Liu, Y. and Ouyang, W. (2023). Ace: Cooperative multi-agent q-learning with bidirectional action-dependency, *Open Journal Systems* .

Li, T., Zhu, K., Luong, N. C., Niyato, D., Wu, Q., Zhang, Y. and Chen, B. (2022). Applications of multi-agent reinforcement learning in future internet: A comprehensive survey, *IEEE Communications Surveys Tutorials* **24**(2): 1240–1279.

Liu, M., Pan, L. and Liu, S. (2023a). Cost optimization for cloud storage from user perspectives: Recent advances, taxonomy, and survey., *ACM Journals* pp. 1–2.

Liu, M., Pan, L. and Liu, S. (2023b). Rltiering: A cost-driven auto-tiering system for two-tier cloud storage using deep reinforcement learning, *IEEE Transactions on Parallel and Distributed Systems* **34**(2): 1–2.

Rehman, A. U., Aguiar, R. L. and Barraca, J. P. (2022). Fault-tolerance in the scope of cloud computing, *IEEE Access* **10**: 63422–63441.

Wong, A., Bäck, T., Kononova, A. V. and Plaat, A. (2022). Deep multiagent reinforcement learning: challenges and directions, *Springer Nature* .

Xiao, Y., Fu, S., Choi, J. and Zheng, C. (2023). A collaborative energy management strategy based on multi-agent reinforcement learning for fuel cell hybrid electric vehicles, *2023 IEEE 98th Vehicular Technology Conference (VTC2023-Fall)*, pp. 1–5.

Zhou, T., Zhang, F., Shao, K., Dai, Z., Li, K., Huang, W., Wang, W., Wang, B., Li, D., Liu, W. and Hao, J. (2023). Cooperative multi-agent transfer learning with coalition pattern decomposition, *IEEE Transactions on Games* pp. 1–13.