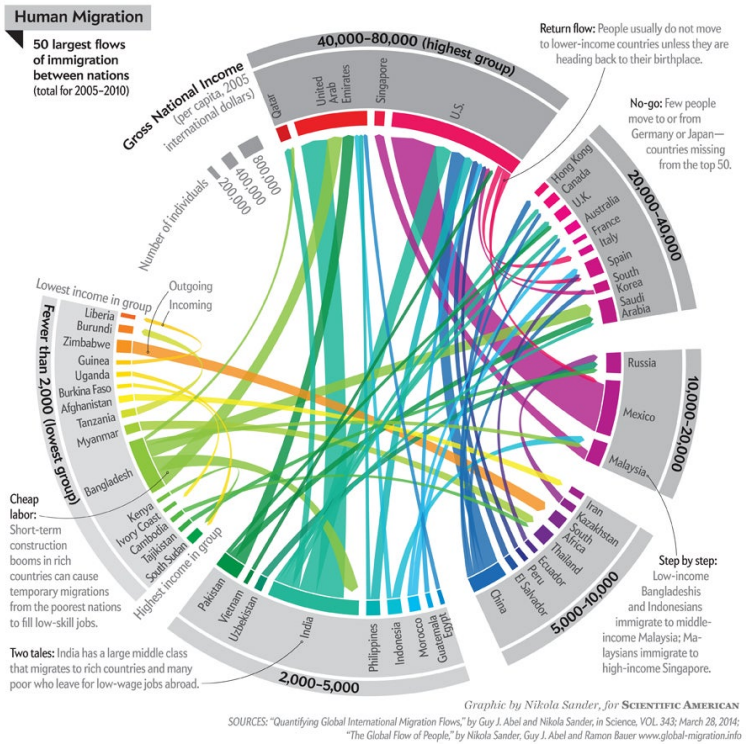
Anomaly Tolerant Resource Management in Cloud-Edge Continuum Using Learning-Based Techniques

submitted in partial fulfillment for the degree of master of science

Efstathios Trigkas
14765667

Master Information Studies
data science
Faculty of Science
University of Amsterdam

Submitted on 28.06.2024



	UvA Supervisor
Title, Name	Hongyun Liu
Affiliation	
Email	h.liu@uva.nl

Amsterdam Data Science

The Awesome Company ™

ABSTRACT

Anomaly-tolerant resource management has been gaining attention for various optimization approaches, particularly within the context of the Cloud Edge Continuum (CEC). These concepts have been explored in the literature over the years, but typically as isolated topics. Our research aims to bridge the gap between resource management and fault tolerance by proposing a deep reinforcement learning-based model called DeepRM-AS-AT. This model effectively manages resources within a computational system while simultaneously addressing anomalies in the scheduling process by proactively migrating them to a scalable cloud machine. Compared to the state-of-the-art algorithm DeepRM, which was trained with a 10% anomaly rate, our "expensive" model variation reduces the percentage of unexecuted jobs from 14.2% to 1.9% and decreases the average job slowdown from 9.4 to 1.2. Further research is proposed to utilize a real training dataset instead of a synthetic one to better evaluate the anomaly tolerance of the system.

KEYWORDS

cloud-edge continUum, anomaly tolerance, deep reinforcement learning, resource managment, job scheduler

GITHUB REPOSITORY

https://github.com/stathis99/Anomaly-Tolerant-DeepRM

1 INTRODUCTION

In recent years, Cloud-Edge continuum (CEC) has emerged as a hybrid architecture that leverages the strengths of both cloud and edge computing [11, 25]. By bringing computation and storage closer to the data sources and end-users, cloud-edge computing contributes to reduced processing and response times, network load, and overall system performance.

Traditionally, in computer systems the way to tackle scheduling of resources like CPU, Memory and I/O devices etc. is by designing heuristic algorithms that prioritize finding good solutions quickly [1, 8, 15, 16, 23, 27, 31]. However, the solution found may not be the optimal one among all possible solutions to the problem, or it may merely approximate the exact solution.[3] To effectively manage the complexities inherent in the cloud-edge continuum, there is an increasing need to transition from traditional heuristic approaches to more advanced and sophisticated methodologies.

Further research has demonstrated that machine learning offers a feasible alternative to conventional resource management heuristics [7, 19, 35]. These studies have shown that deep reinforcement learning (DRL) can be applied to resource management problems and perform comparably to state-of-the-art heuristics. A notable attempt, that of H. Mao et al. [19], demonstrated the potential of DRL in their algorithm DeepRM which outperformed popular scheduling heuristics such as Shortest-Job-First.

Moreover, the increasing complexity of cloud-edge continuum has exposed the systems to several anomalies, e.g. hardware failures, system dynamics or anomalous workloads involved in the resource management process[4, 5, 13] making the task of scheduling on such systems to be considered as an NP-hard problem. [30, 32]. For those unforeseen anomalies, an anomaly-tolerant orchestrator becomes an ideal solution. In accordance with the research conducted by A. Ledmi et al. [17], distributed systems like Cloud-Edge are susceptible to various types of failures, underscoring the significance of system tolerance. Another study conducted by S. Tuli et al. [28] demonstrated the ability of a machine learning model utilizing a Generative Adversarial Network (GAN) to predict preemptive migration decisions for proactive fault-tolerance in containerized edge deployments outperformed state-of-the-art baselines.

Although DRL-based scheduling algorithms have demonstrated advantages in the reviewed literature, they have only been tested in laboratory simulation experiments where tasks do not reflect the real life scenarios while anomaly tolerance has not been examined. [26, 35] . The existing research indicates that these algorithms perform well for specific tasks; however, their efficacy in handling abnormalities remains unexplored.

With this research, we aim to bridge this research gap by developing a system that addresses the challenges of resource management and anomaly tolerance simultaneously. More specifically, we propose a DRL based model denoted as DeepRL-AS-AD that manages to efficiently schedule workloads comparably to the state-of-the-art machine learning-based job schedulers while concurrently treating anomalies proactively.

In this paper, we ask: *To what extent can a learning- based resource scheduler become anomaly-tolerant in order to make the scheduling performance of the Cloud-Edge Continuum more stable?*

The following research can be further divided into the following sub-questions:

- To what extent can a deep reinforcement learning-based resource management algorithm reduce the percentage of unexecuted jobs in the scheduling process, by proactively migrating anomalous workloads to a cloud machine?
- To what extent can the agent of a deep reinforcement learning-based resource management system simultaneously achieve the dual objectives of minimizing the percentage of unexecuted jobs and job slowdown, while gradually optimizing its policy?
- To what extent can the agent control the cost related to cloud machine usage while simultaneously attaining the objectives defined in the previous questions?

2 RELATED WORK

In this section we review the problem of resource scheduling based of machine learning methods and suggested solutions in the literature. Subsequently, we introduce scientific work related to false tolerance in computational systems. Lastly, we conclude by summarizing the relevant research in the overlap of these two areas and we introduce the relevant research gap.

2.1 Resource management in computational systems

In distributed systems, the scheduling problem is NP-complete generally. [30, 32]. This entails the strategy followed by a computational system for job allocation across executing resources, aiming to optimize a relevant metric such as minimizing energy consumption, minimizing makespan, minimizing delay time, reducing response time, load balancing multi-objectives etc. The above objectives can be summarized as two aspects, reducing cost and increasing profit of Cloud providers or improving quality of services (Qos). [6, 8, 16, 20, 29]

Some of the primitive research about resource management on distributed systems include Meta-Heuristic Models [21] ,minimal-length pre-emptive schedule [22], zero-one integer programming approach [2], Branch-and-bound techniques [18] etc. Nevertheless, growing complexity of massive distributed systems such as cloud-edge continuum, is limiting the application of these approaches.

Recent studies have demonstrated the effectiveness of Machine Learning based methods in resource scheduling of Cloud computing. [7, 19, 33, 35]. Deep reinforcement learning (DRL), a combination of deep learning (DL) and reinforcement learning (RL), is one branch of the machine learning and has a considerable prospect in scheduling problems.

A paper that summarizes this field is that of G. Zhou [35]. In their research they surveyed methods of resource scheduling with focus on DRL-based scheduling approaches in Cloud computing, reviewed the application of DRL as well as discussed challenges and future directions of DRL in scheduling of Cloud computing. This research concludes that DRL is one of effective methods to solve the dynamic resource scheduling of large-scale Cloud computing. However, the paper states several challenges that these algorithms encounter, one of them being that they are only tested at the laboratory simulation experiments where tasks do not represent the complexity in real Cloud environments. Furthermore, real scheduling relies on predicting dynamic tasks without pre-emptive or prior knowledge. DRL-based methods fail to adequately address this challenge.

Another paper that is considered a state-of-the-art in the field of DRL-based scheduling is that of Mao [19]. In his research, he proposed an algorithm named DeepRL that is comparable and sometimes better than ad-hoc heuristics for a multi-resource cluster scheduling problems. At the core of his research is an agent that learns from experience to allocate incoming jobs to a computational system with the objective to minimize job slowdown. His method manages to outperform popular scheduling algorithms such as shortest-job-first and Packer. The research concludes that if the model would work in a practical context, this could offer a real alternative to current heuristic based approaches.

Their have been several studies based on Maos' work most notably the research of Yufei Ye. [9] and Wenxia Guo [33]. Their work is based on DeepRM but their solutions have faster convergence speed and better scheduling efficiency with regarding to average slowdown time. This is achieved through the implementation of imitating learning techniques to accelerate convergence of the model and the use of convolution neural networks to better capture the state of the environment.

2.2 Fault Tolerance in Distributed Systems

As the number of connected devices in the cloud-edge continuum continues to expand, its complexity has grown proportionally. This increased complexity has rendered the continuum more susceptible to anomalies, thereby reducing its fault tolerance[4, 5, 13]. Fault Tolerance is defined as the ability of the system to function properly even in the presence of any failure. It is a measure of the system's resilience against disruptions and its capability to ensure consistent and reliable performance.

As shown in previous research [17] the design of distributed systems places significant emphasis on fault tolerance. In the event of a hardware or software failure within the system, termed as a fault, the system's functionality is disrupted. To ensure uninterrupted operation despite these faults, techniques are employed to tolerate failures. The techniques examined in the paper aim to detect and correct errors, enabling the system to maintain its functionalities. According to this research, there are two main directions for achieving false tolerance. Reactive fault tolerance techniques are used to reduce the impact of failures on a system when the failures have occurred and Proactive fault tolerance expects the faults proactively and places healthy (working) components in place of the faulty components, to prevent recovery from faults. Similarly, another research that of Arshad A. [10] provides state of the art fault tolerant techniques in distributed systems to ensure replication, high redundancy, and high availability of distributed services. He recognizes pre-emptive migration and load balancing as a key proactive method to ensure fault tolerance.

Further research conducted by M.Kirti [14] addresses the problem of fault tolerance through pre-emptive migration in heterogeneous cloud environments. With the proposed task allocations algorithm (FTTA), they manage to reduce task failures through pre-emptive migration in heterogeneous cloud environments. FTTA prioritizes tasks to minimize execution time , selects suitable virtual machines, and assigns tasks to optimize execution within deadline limits. During the task allocation process, the algorithm adopts fault-tolerant strategy that includes pre-emptive migration if necessary, which allows the migration of tasks to identify the best suitable virtual machine.Experimental results show that FTTA outperforms existing algorithms like FCFS, Priority-based, SJF, Dy max min, and RADL in reducing rejected tasks, make span, and improving speed and efficiency. Similarly, another study conducted by S. Tuli [28] demonstrated the ability of a machine learning model utilizing a Generative Adversarial Network (GAN) to predict preemptive migration decisions for proactive fault-tolerance in containerized edge deployments while outperforming state-of-the-art baselines.

2.3 RL based anomaly tolerant resource management

An overlap between the areas of learning-based resource management and fault tolerance can be observed in the research work of S. Moghaddam [12]. The proposed solution utilizes an anomaly detection module to detect persistent performance problems in the system, triggering the decision-making module of reinforcement learning (RL) to initiate a scaling action for correcting the issue. This solution combines the two concepts; however, the agent is

triggered only when an anomaly is detected to correct the problem and does not serve as the main job scheduler.

In summary, most research in these domains has investigated the concepts of resource management and fault tolerance in isolation. However, minimal research has been conducted to bridge the gap between these two concepts. Our research will focus on addressing this gap and propose a solution that tackles these problems simultaneously.

3 PROBLEM FORMULATION

3.1 Policy-Based Deep Reinforcement Learning and DeepRM

In this section we give a brief overview of Policy-based deep reinforcement learning methods, as it is a fundamental aspect of our research. Readers can find a more detailed description in the literature [24, 34].

A general description of a DRL problem can be established in Figure1 where an agent interacts with an environment. At each time step t the agent observes the state s of the environment and picks an action a. According to the action, the environment transitions to state $s[t+1]$ and the agent receives a reward r. The agent learns by repeating this interaction, while the goal of the agent is to maximize the expected cumulative discounted reward :

$$\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right],$$

where $\gamma \in (0, 1]$ is a factor discounting future rewards.

It is important to note that the next state of the environment and the rewards follow the Markov Decision property. This means that they depend only on the current state s and action a taken by the agent and not on the full history of previous actions or states.

The agent picks actions based on a *policy* π defined as a probability distribution over actions $\pi : \pi(s, a) \rightarrow [0, 1]$. In the case of DeepRM, due to the nature of the problem, there can be millions of different $(state, action)$ pairs making it impossible to solve this with traditional RL methods. This problem has been solved by employing a function approximator.

A function approximator makes sure that similar states output similar actions. DeepRM uses a deep neural network with parameters θ to represent the function approximator. The idea behind this is to have a more manageable numbers of parameters, referred as policy parameters in order for the agent to match states with actions.

In order for the agent to maximize the cumulative discounted reward, it performs gradient-decent on the parameters θ. The gradient of this objective is given by :

$$\nabla_\theta \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right] = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)\right].$$

$Q^{\pi_\theta}(s, a)$ represents the expected cumulative discounted reward when following the policy π_θ. A crucial step in this method is to observe different trajectories of executions using simple Monte Carlo Method and estimate the gradients. Finally, the policy parameters are updated using the following equation:

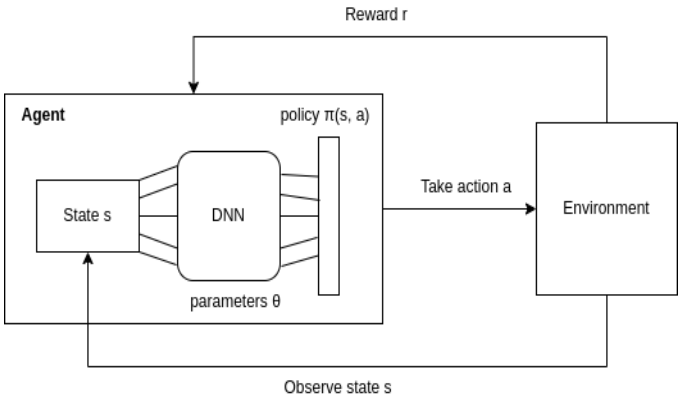


Figure 1: DRL overview

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta \log \pi_\theta(s_t, a_t) v_t.$$

In this formula, a is the step size for the gradient descent. The term ∇_θ adjusts the policy parameters to increase or decrease the likelihood of selecting action a in state s based on the gradient direction. The magnitude of each update step is influenced by the return v_t , meaning that the parameter θ is adjusted to enhance the probability of actions that lead to higher returns.

4 METHODOLOGY

This study aims to bridge the gap in the literature by providing a consistent job scheduling algorithm that efficiently schedules jobs with the objective of minimizing slowdown while simultaneously is able to cope with anomalies in the scheduling process, thereby increasing its anomaly tolerance.

The first aspect of the study builds on the work of Mao's DeepRM [19], which is considered state-of-the-art in DRL based job scheduling. Utilizing the source code provided by the original researcher, we will implement modifications to critical components particularly in the reward system of the agent, the output layer of the DNN and the Environment that the agent interacts with in order to address our research questions.

At the same time, we will address anomaly tolerance by leveraging the cloud-edge architecture. Specifically, we will integrate a cloud computing cluster into the existing system, enabling the agent to proactively isolate and manage anomalous workloads [17]. Additionally, we will introduce a cost associated with cloud cluster usage to evaluate the agent's ability to control job migration efficiently.

The combination of these approaches will be implemented in the proposed method, DeepRM-AT-AS. This approach is investigated in the rest of the paper.

4.1 System Design

This section outlines the architecture of the computational system simulation. The system, depicted in Figure 2, shares similarities with DeepRM in its utilization of comparable components. However, it distinguishes itself through the incorporation of anomalous

coping mechanisms, which are integral aspects of our approach.The objective of this section is to provide a high-level overview of the entire system, including the interconnections between these key components.

At its core, our architecture consists of several main components. At the centre of Figure 2 (green part) is the Environment, which the Agent interacts with. This Environment includes two primary subcomponents: the Job Queue on the left and the execution machines on the right. Jobs flow from the Job Queue to one of the execution machines upon the agent's request. In the top left of Figure 2 (yellow part) is the Job Generator, which is responsible for generating and sending jobs to the Job Queue at each time step. The bottom section of Figure 2 (blue part) represents the Agent and its interconnection with the Environment. As illustrated, the Agent observes the state of the Environment at each time step. This state is provided as input to the neural network, which then outputs an action for the Environment to follow. The action involves selecting which job from the Queue will be executed in the next time step.

Further details about each component are elaborated below:

- **Job Generator:** The job generator is responsible for creating synthetic data called jobs. This dataset is utilized for training and validating the model. Details about the dataset and the job generation mechanism is discussed in subsection 4.2.
- **Job Queue:** This component functions as a holding area for generated jobs. Jobs remain in the queue until they are allocated to one of the two available machines for execution.
- **Agent:** The agent is responsible for making allocation decisions. At each time step t, it selects a job from the queue and determines the most suitable machine for its execution (further details on the agent's decision-making process will be discussed in section 4.3).
- **Edge Machine:** The edge machine is a fixed-size computing resource specifically designed for executing small and normal-sized jobs.
- **Cloud Machine:** The cloud machine, in contrast, is a scalable resource capable of executing all job types, including large and anomalous ones. However, its utilization is constrained by two specific parameters:
  (1) **C.O.U** (Cost Of Usage): This parameter is designed to introduce a cost element associated with the computational resources utilized by the Cloud Machine. By incorporating this cost into the agent's reward function, the system discourages or encourages the allocation of normal jobs to the Cloud Machine.
  (2) **N.D** (Network Delay): This parameter serves to emulate the network latency inherent in the cloud-edge continuum. By incorporating this delay into the agent's decision-making process, the system introduces a time penalty for jobs allocated to the Cloud Machine. This approach encourages the agent to prioritize the Edge Machine for non-anomalous workloads.

4.2 Training Data

As previously established, the job generator is essential for creating synthetic data for model training. The utilization of synthetic data offers several advantages over real-world data in this specific context.

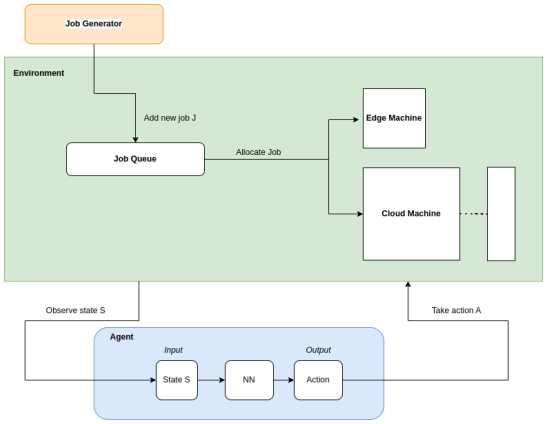


Figure 2: System Design

Firstly, synthetic data generation facilitates the creation of sufficiently large datasets, a crucial requirement for effective DRL training. Secondly, synthetic data gives us the ability to manipulate parameters and simulate diverse training scenarios.

Upon initialization of the environment, the job generator creates a scenario of length t , corresponding to the time steps of the simulation. For each time step, the generator, draws samples from a predetermined probability distribution in order to decide whether to generate a job, thereby simulating the arrival rate of jobs.

For each job J , assuming that the machine's host d types of resources (e.g. CPU and Memory) is represented by the following parameters:

- The job ID: We denote the job ID as j_i, where $j \in J$ is the set of all jobs, and i is the unique ID of job j.
- The job resource requirement:
  Denoted as $r_j = \{[r_{1j1}, r_{2j1}], \ldots, [r_{1jn}, r_{2jn}]\}$, where r represents the amount per type of resource required by job j.
- The job length: Represented as j_L, the length of job j is in the range of Lower Boundary to Upper Boundary: $L_n = \{L \in \mathbb{N} \mid LB \leq L \leq UB\}$.

The DeepRM job generator utilized a uniform distribution to simulate jobs, resulting in fixed boundaries same probability of occurrence and an absence of anomalies. To enhance the realism of job generation, our approach involves modifying the probability distribution to draw samples from a log-normal distribution. As can be obtained from Figure 3 this adjustment ensures the occurrence of outliers (anomalies) due to the heavy tail of the log-normal distribution, and the Gaussian approximation of this distribution renders jobs more realistic. In our code, we modify the mean and standard deviation of the function in order to simulate different anomaly rates for different scenarios.

Furthermore, the arrival rate of DeepRM's job generator follows a Bernoulli distribution with a fixed probability of success. To make the arrival rate more realistic, we employ the sine trigonometric function to simulate fluctuations in traffic, such as those occurring

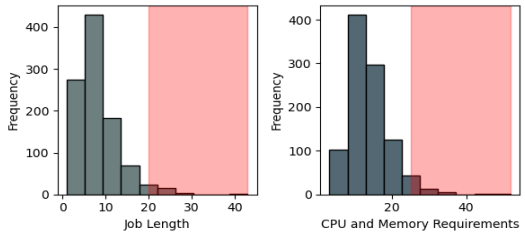


Figure 3: Job Distribution Profile : The red part of the graph represents the jobs that are considered anomalous

during day and night, and the Poisson distribution to model random spikes. These random spikes simulate extreme traffic in arrival rates and are considered anomalous.

4.3 Model

The core investigatory part of this paper is the model used for job scheduling. This constitutes a deep reinforcement learning based model that effectively performs allocation of incoming jobs to the two machines available, with the primary objectives of minimizing job slowdown and detecting and migrating anomalies to the Cloud Machine.

Our proposed model, DeepRM-AT-AS, builds upon the original DeepRM but introduces key modifications, particularly in the reward system and the output layer of the DNN, to support our established multi-objective goal. The subsequent sections provide a detailed description of the various dimensions of our proposed model.

Agent : The agent comprises a deep neural network architecture consisting of three layers. The initial layer receives input in the form of image representations depicting the state of the Environment, including the Job Queue and the Machines, along with the status of each job within them. Subsequently, the next layer processes the input data and produces an action as output, which instructs the environment on which job to select next for execution.

State Space : The state space is given to the input layer of the agent in the form of an image. As illustrated in Figure 4, the state of the environment is composed of two components:

(1) State of Job Queue: This represents the waiting queue where jobs await resource allocation. The horizontal length of each array indicates the demand of a job for a specific resource, while the vertical length corresponds to the duration of the job. In the example depicted in Figure 4, the first row of job slots represents the CPU requirements, and the second row represents the memory requirements.

(2) State of Machines: Represents the current resource allocation in the Edge and Cloud Machines. As illustrated in Figure 4, different colours denote various jobs that have been allocated. Notably, the Edge Machine has a fixed allocation capacity, whereas the Cloud Machine is dynamic, capable of executing a significantly larger number of jobs.

Action Space : The original model produced two distinct actions: "Allocate" and "MoveOn." In our methodology, we introduced a supplementary action denoted as "CloudAllocate." This addition

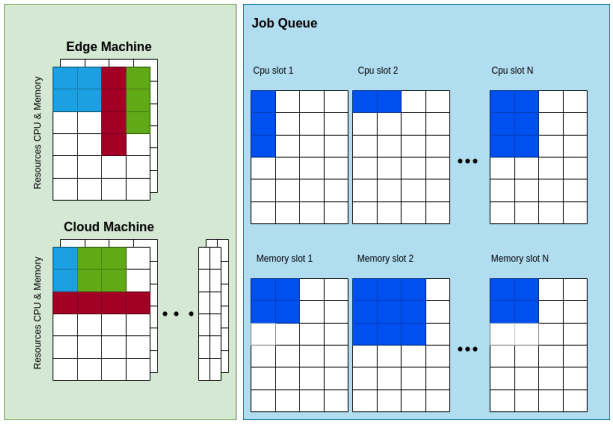


Figure 4: Env Image

was devised to effectively address anomalies within the system by allocating jobs to the Cloud Machine when necessary.

- Allocate : Allocates job i from the job queue on the Edge Machine.
- CloudAllocate : Allocates job i from the job queue on the Cloud Machine.
- MoveOn : Move on the next time step $t + 1$.

Reward System As described in Section 3, the reward system serves as a feedback signal from the environment to the agent. The main purpose is to evaluate how well the agent performs the current action in the current state. The agent uses this signal as a guidance towards the optimal solution.

In our approach, we address a multi-objective problem where the agent must solve two different tasks simultaneously. Firstly, it needs to reduce job slowdown. Secondly, it must detect and migrate anomalous jobs to the Cloud Machine. Additionally, we set an additional goal for the agent to control cloud usage effectively. In the following section, we will describe how these three objectives are represented mathematically.

The part of the reward function that solves the first problem is:

$$R1 = \sum_{j \in J_M} \frac{-DelP}{j_L} + \sum_{j \in J_Q} \frac{-HolP}{j_L} + \sum_{j \in J_B} \frac{-DisP}{j_L} \tag{1}$$

The J_M , J_Q, J_B represent the jobs that exist in each time step in the execution machine, waiting queue and the backlog accordingly while j_L represents the length of each job. For each job we compute penalty divided by the job length and sum all quantities. This part of the function guides the agent to assign jobs as quickly as possible and is based on the reward system of DeepRM.

The part of the reward function that isolates anomalous workloads to the Cloud Machine is the following:

$$R2 = \sum_{j \in CM} \Big[I(\max(\vec{j_r}) \geq RT \vee j_l \geq LT) \times AlR + I(\max(\vec{j_r}) < RT \wedge j_L < LT) - 1 \times AlP \Big] \tag{2}$$

In this part, our objective is to incentivize the agent to allocate anomalous jobs on the cloud while penalizing it for allocating

normal jobs on it. The RT and LT variables serve as thresholds for anomalous jobs in resource and length dimensions, while AlR and AlP are the allocation reward and penalty.

Finally, we sum the two parts and return the overall reward for the two objectives to the agent.

$$R = R1 + R2 \tag{3}$$

4.4 Model Training

As described in Section 3 , during the training phase, the model learns the optimal policy through trial and error. The agent observes the environment and takes actions based on a current policy π_θ . Following each action, the state transitions to the next state, and the agent receives a reward v_t. After a set number of time steps, the current policy is updated by performing gradient descent on the parameters, with the objective of maximizing the expected cumulative reward in the next trial. This loop continues for a predetermined number of steps, called an *epoch*.

In this section we will briefly describe the training algorithm of DeepRm-AS-At. For more informations we refer readers to the original DeepRm paper [19].

Upon initialization of the training process, we generate a predetermined number of job sequences, referred to as $jobsets$. These distinct jobsets serve as training examples for the model, facilitating its ability to generalize across various scenarios. We define a number of training iterations, which indicate the number of consecutive training loops we intend to use for training the model.

Each iteration, referred to as an epoch, involves simulating n episodes for each jobset. An episode is a scenario with a fixed duration, during which the agent allocates the incoming jobs from the jobset based on the current policy. We simulate multiple episodes, rather than a single one, to enable the agent to explore the probabilistic space of potential actions by introducing an element of randomness into the chosen actions. This approach allows for a more comprehensive exploration of the action space.

After the completion of all episodes, we utilize the resulting data (state, action, rewards) to calculate the discounted cumulative reward v_t at each timestep t of each episode. Based on the calculated discounted cumulative rewards, we estimate the policy gradient, which is then used at the end of each epoch to update the parameters of the policy. It is important to note that, due to the high variance of the policy gradients, the original deep reinforcement learning algorithms subtract a baseline value from each reward v_t to reduce this variance.

4.5 Proposed Model Variations

In this paper, we propose a methodology to train two variations of the DeepRM-AS-AT model. The first variation, called DeepRM-AS-AT-expensive, aims to optimize job allocation for speed, without considering cloud usage costs. The second variation, named DeepRM-AS-AT-cost-efficient, focuses on minimizing cloud usage costs while maintaining job slowdown comparable to our baselines. Both models are designed to tolerate incoming anomalies. The different objectives and the relevant metrics will be examined in the Evaluation section.

Both variations have been trained by modifying the reward system of the agent. Specifically, we have adjusted the variable ALP at the core of the reward system. As described earlier, this variable represents the penalty for allocating non-anomalous jobs to the cloud cluster. The parameters that the model variations have been trained can be seen in Table 1

DeepRM-AS-AT-expensive : For the "Expensive" model, we set the ALP parameter to -0.01 and the rest of the parameters to -1. This ensures that during the training process, the agent is penalized for high job slowdown, thereby incentivizing the agent to use the cloud for allocating both normal and anomalous jobs. The positive reward for anomalous job allocation also ensures that anomalous jobs are allocated to the cloud machine as quickly as possible.

DeepRM-AS-AT-cost-efficient : For the "Cost-Efficient" model, we set the ALP parameter to -5. This ensures that during the training process, the agent is penalized for allocation of normal jobs on the cloud machine, thereby incentivizing the agent to use the cloud for allocating only anomalous jobs. With this variations less jobs will be sent on the cloud machine resulting to less cost of usage.

Algorithm 1: Pseudo-code for training algorithm

```
jobsets ← GenerateJobs();
for each epoch do
    for each jobset do
        ∇_θ ← 0;
        for each episode do
            Run episode following current π_θ and return
            reward v_t
        end
        for each timestep do
            compute baseline b_t
            for each episode do
                ∇_θ ← ∇_θ + α∇_θ log π_θ(s_t, a_t)(v_t − b_t)
            end
        end
    end
    θ ← θ + ∇_θ
end
```

Table 1: Reward Function Parameters

	DelP	HolP	DisP	AlR	AlP
Expensive	-1	-1	-1	1	-0.01
Cost Efficient	-1	-1	-1	1	-5

4.6 Evaluation

To evaluate our proposed method, we will compare two variations of DeepRM-AS-AT with two variations of the original DeepRM model. Specifically, our baseline will consist of the DeepRM model trained without anomalous workloads in the job set and the DeepRM model trained with a 10% anomalie rate.

In each case, the settings of the experiment and the environment are the same. We train the four models with 10 job sets containing 50 jobs each. In each iteration, each job set explores 10 different trajectories (episodes). We also enforce an artificial terminal of length 200 for each episode to execute.

The four different models will be evaluated based on three performance dimensions: job slowdown, anomaly tolerance, and the cost of cloud usage.

Job slowdown: The average slowdown is a metric used to evaluate how efficiently the agent allocates jobs on the execution machines, with the objective of minimizing slowdown. This metric, used by the original DeepRM paper to evaluate performance, is calculated as follows:

$$\text{Job Slowdown} = \frac{1}{N} \sum_{i=1}^{N} \frac{\text{Job Completion Time}_i - \text{Job Arrival Time}_i}{\text{Job Execution Time}_i}$$

where N is the total number of jobs.

Anomaly tolerance: The anomaly tolerance aspect of the model will be evaluated by calculating the average number of jobs that fail to get executed after the end of each episode. This may occur due to anomalies interfering with the allocation process. This metric is calculated by dividing the number of jobs that are in the job queue or the backlog after the end of the episode by the total number of jobs in the job set.

Cost of Cloud Usage: The cost of cloud usage is a metric responsible for measuring the usage of the Cloud machine. Our objective, especially with DeepRM-AS-AT-cost-efficient, is to achieve a balance between the Cloud Machine and the Edge Machine usage. It is calculated by multiplying the total number of jobs allocated on the cloud machine by the length of each job and then averaging this value for each episode in every training iteration.

4.7 Experimental Setup

The training of the models will be conducted on the Snellius National Supercomputer. The Snellius cluster is ideal for deep neural network training due to its high-performance computing hardware. Specifically, the partition that will be used is called "gpu" and consists of a Platinum 8360Y (2.4GHz) processor with 72 cores, 480GB RAM, and 4x NVIDIA A100 GPUs with 40GB HBM2 each.

Different versions of our code, along with the parameters of our experiments, will be stored on the version control system GitHub, ensuring reproducibility of our experiments. To collect the various metrics necessary for the evaluation of our methodology, we have added scripts to collect and store the metrics in CSV files and the data can be found in a seperate folder named "Experiments".

5 RESULTS

The results for each tested model are presented in this section. The models tested were DeepRM-AS-AT-expensive and DeepRM-AS-AT-cost-efficient, with two variations of the original DeepRM model used as benchmarks. Each model was trained with 10 job sets consisting of 50 jobs each for 1000 iterations. In each iteration, the models explored 10 different trajectories for each job set. Finally, we evaluated the models based on three different metrics: job slowdown, anomaly tolerance, and cost of cloud usage. Figure 8 demonstrates each model's ability to train by gradually optimizing their policy, maximizing the reward received from the environment throughout the iterations.

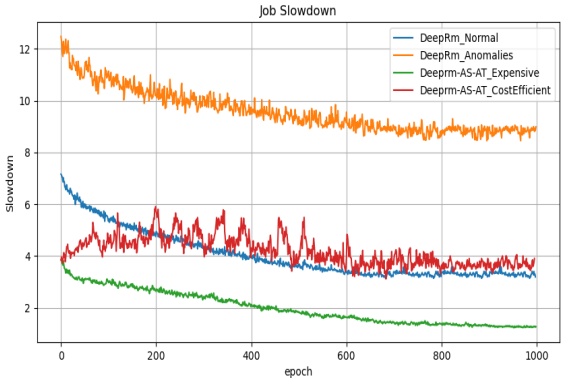


Figure 5: Average Job Slowdown comparison for all models

5.1 Job slowdown

As described in subsection 4.6, job slowdown as a metric aims to evaluate how effectively a scheduler's strategy reduces the average slowdown of jobs waiting to be allocated in a computational system. The primary objective of the DeepRM model was to reduce this metric comparing to different heuristic algorithms. In our approach, while this objective is considered important, it is not the only one. We also aim to improve the pre-existing method for anomaly tolerance simultaneously.

As observed in Figure 5, at the 1000th iteration of training DeepRM-anomalous (the original model with 10% anomalies) performs significantly worse than the other models, reaching a mean slowdown of 9.4. Figure 6 and a later subsection will discuss that the high slowdown is caused by the original model's inability to handle anomalous jobs in the scheduling process, resulting in an average of 14.2% of jobs not being executed within the set timeframe limit of 200 time steps. Following this is DeepRM-AS-AT-expensive (high cloud usage), which reached a mean of 3.9, DeepRM-Normal (the original model) with a mean of 3.1, and finally, DeepRM-AS-AT-cost-eff (low cloud usage) with a mean of 1.2.

5.2 Anomaly Tolerance

The anomaly tolerance aspect of our approach is evaluated by the ability of the job scheduler to schedule all incoming jobs in a limited timeframe, which is in our settings 200 timesteps. Figure 6 shows the average percentage of not executed jobs during each episode in each iteration.

DeepRM-Anomalous has a significantly larger percentage of non-executing jobs, indicating the model's inability to execute an average of 14.2% of the assigned jobs, with a variance of 38.4. This inability is sometimes caused by anomalous jobs requiring extremely large amounts of resources, which the machine lacks the capacity to allocate. In other cases, the job queue gets blocked by these anomalous jobs, resulting in an exceptionally high percentage of non-executing jobs, reaching up to 27%

Our approach, for both trained models, has a significantly reduced percentage of non-executing jobs, demonstrating the capability of our approach to tolerate anomalies within the scheduling

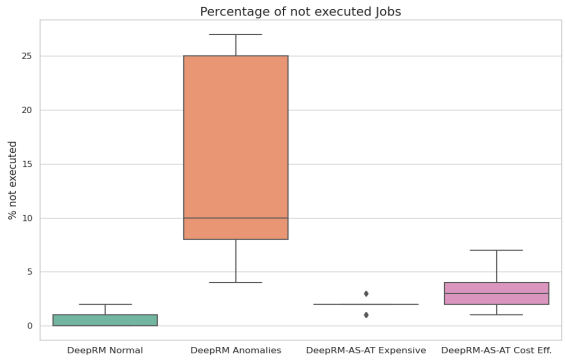


Figure 6: Anomaly Tolerance : average percentage of unexecuted jobs in each iteration

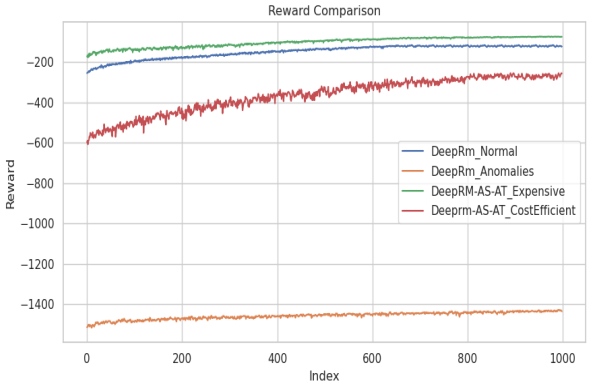


Figure 8: Mean reward comparison for all models

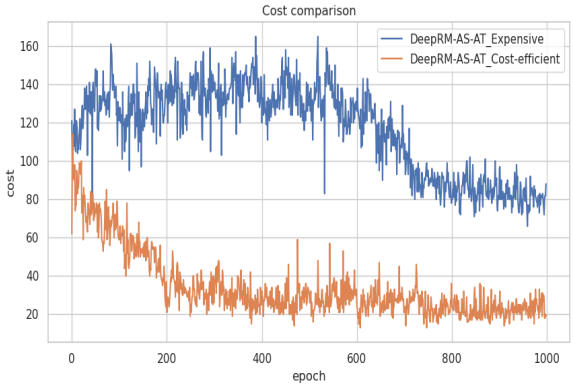


Figure 7: Average cost comparison in each iteration for DeepRM-AS-AT

process. Specifically, DeepRM-AS-AT-expensive achieves an average percentage of 1.9% with a variance of 0.2, while DeepRM-AS-AT-cost-eff has an average of 3% with a variance of 1.3.

5.3 Cost of Cloud Usage

The cost of cloud usage is an additional metric we introduced to compare the two variations of the DeepRM-AS-AT model in terms of each agent's ability to control the usage of cloud resources. While not directly associated with our primary focus on job slowdown and anomaly tolerance, it serves as a secondary dimension of the initial problem.

As discussed in subsection 4.6, the cost of cloud usage reflects the average utilization of cloud resources. In Figure 7, we compare the expensive and cost-efficient models relative to this metric. As observed, in the 1000th iteration, the expensive model has an average cost of 84, while the cost-efficient model has an average of 19, demonstrating the ability of the latter model to reduce the usage of the cloud machine.

6 DISCUSSION

6.1 Reflection

In this study, we propose an algorithm named DeepRM-AS-AT that is responsible for effective resource management within a simulation of a job scheduler. Compared to the state-of-the-art algorithm DeepRM, which has been trained with a single objective of reducing job slowdown, our algorithm simultaneously addresses anomalies in the scheduling process by proactively migrating them to a cloud cluster and learns from experience to control the cost associated with cloud usage. We compare our approach with the state-of-the-art and evaluate the effectiveness of the two methodologies based on three key metrics: job slowdown, percentage of unexecuted jobs and cost of cloud usage.

Our results indicate that Deep-RM-AS-AT-Expensive achieves a lower job slowdown compared to DeepRM. It is important to note that this reduction is partly attributable to the increased computational capacity provided by the addition of the cloud cluster. Moreover, both of our model variations achieve a reduction in the percentage of unexecuted jobs compared with DeepRM trained with a percentage of anomalies. This improvement is due to the anomaly capping mechanism introduced in our research.

Another distinction of our model from the state-of-the-art is that we provide an additional objective: the ability to control cloud machine usage. The two variations of our model serve this purpose, and when compared with each other, it can be established that the cloud usage is effectively controlled by the agent while simultaneously the main objectives of anomaly tolerance and job slowdown reduction are achieved.

Overall, in this project, we have introduced an innovative approach to optimizing the cloud-edge continuum through a learning-based, anomaly-tolerant resource management method. Our method leverages deep reinforcement learning (DRL) to manage resources and address anomalies simultaneously.

6.2 Limitations

In all conducted experiments, the training datasets and environment sizes were fixed to ensure that model training would converge to a solution within a reasonable timeframe. Although specific settings

were used to ensure reproducibility, this approach may potentially hinder scalability, as we have not tested whether our approach remains effective and efficient on a larger scale. Previous research [9] has addressed this issue by employing convolutional neural networks (CNNs) to better capture state observations, which can lead to faster training times and convergence towards an optimal reward. The increasing complexity can be managed through various methods, such as load balancing the training of the deep reinforcement learning (DRL) model across a cluster of multiple instances.

Furthermore, another limitation pertains to the internal validity of the anomaly tolerance aspect of the model. While our results indicate that DeepRM-AS-AT effectively manages anomalies, using the percentage of unexecuted jobs as a metric does not accurately validate whether the model identifies anomalous jobs and treats them separately from normal jobs. A potential solution for further research could involve implementing a tracking mechanism to monitor the status of each job at every time step and analyzing the differences between normal and anomalous jobs.

Finally, the generalizability of the model may be limited due to the use of synthetic data for training instead of real-world data. While synthetic data provides the flexibility to manipulate parameters and simulate diverse training scenarios, it can also result in model overfitting. Additionally, the deep reinforcement learning models presented in this paper were trained with a specific set of hyperparameters, without extensive or automatic tuning. Variations in hyperparameter values could lead to different performance outcomes, especially in comparison to studies that have thoroughly optimized their hyperparameters. These limitations highlight the need for future research to incorporate real-world data and conduct comprehensive hyperparameter tuning

7 CONCLUSION

In this paper, we aim to bridge the gap in the literature regarding learning-based resource management and anomaly tolerance in the cloud-edge continuum. We investigate the extent to which a learning-based resource scheduler can become anomaly-tolerant to enhance the stability of scheduling performance within the Cloud-Edge Continuum.

To address this question, we propose an algorithm named DeepRM-AS-AT. This algorithm effectively manages resources within a computational system while simultaneously addressing anomalies in the scheduling process by proactively migrating tasks to scalable cloud machines. By comparing our results with the state-of-the-art resource management algorithm DeepRM, we demonstrate that our algorithm successfully achieves the multi-objective goal of anomaly tolerance and effective resource management. Our results indicate that the methodology effectively reduces the percentage of unexecuted jobs and performs better in terms of job slowdown. Additionally, we introduce another dimension to the problem, enabling the algorithm to efficiently control cloud machine usage, as evidenced by the differences between the two variations of our model.

Our results highlight the potential of deep reinforcement learning (DRL)-based resource management in complex and anomaly-prone systems. However, our approach is subject to several limitations, as discussed in Subsection 6.2, most notably regarding the generalizability of the model and the scalability of the entire system. To better evaluate the potential of our approach, we propose further research in the following directions: the use of a real-world dataset that incorporates the actual scenarios a job scheduler might encounter, and hyperparameter tuning of the model to identify the optimal parameters. This will help in grasping the complexities inherent in the data and could potentially demonstrate the real-world applicability of the approach studied in this paper.

REFERENCES

[1] Zahra Beheshti and Siti Mariyam Hj Shamsuddin. 2013. A review of population-based meta-heuristic algorithms. *Int. j. adv. soft comput. appl* 5, 1 (2013), 1–35.

[2] Wesley Chu, Leslie Holloway, Min-Tsung Lan, and Kemal Efe. 1980. Task Allocation in Distributed Data Processing. *Computer* 13 (12 1980), 57 – 69. https://doi.org/10.1109/MC.1980.1653419

[3] Sachin Desale, Akhtar Rasool, Sushil Andhale, and Priti Rane. 2015. Heuristic and meta-heuristic algorithms and their relevance to the real world: a survey. *Int. J. Comput. Eng. Res. Trends* 351, 5 (2015), 2349–7084.

[4] Manoj Devare. 2019. *Challenges and Opportunities in High Performance Cloud Computing*. https://doi.org/10.4018/978-1-7998-5339-8.ch096

[5] Faraz Fatemi Moghaddam, Mohammad Ahmadi, Samira Sarvari, Mohammad Eslami, and Ali Golkar. 2015. Cloud computing challenges and opportunities: A survey. In *2015 1st International Conference on Telematics and Future Generation Networks (TAFGEN)*. 34–38. https://doi.org/10.1109/TAFGEN.2015.7289571

[6] Sukhpal Singh Gill, Inderveer Chana, Maninder Singh, and Rajkumar Buyya. 2018. CHOPPER: an intelligent QoS-aware autonomic resource management approach for cloud computing. *Cluster Computing* 21 (06 2018). https://doi.org/10.1007/s10586-017-1040-z

[7] Sepideh Goodarzy, Maziyar Nazari, Richard Han, Eric Keller, and Eric Rozner. 2020. Resource Management in Cloud Computing Using Machine Learning: A Survey. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 811–816. https://doi.org/10.1109/ICMLA51294.2020.00132

[8] Songtao Guo, Jiadi Liu, Yuanyuan Yang, Bin Xiao, and Zhetao Li. 2018. Energy-Efficient Dynamic Computation Offloading and Cooperative Task Scheduling in Mobile Cloud Computing. *IEEE Transactions on Mobile Computing* PP (04 2018), 1–1. https://doi.org/10.1109/TMC.2018.2831230

[9] Wenxia Guo, Wenhong Tian, Yufei Ye, Lingxiao Xu, and Kui Wu. 2021. Cloud Resource Scheduling With Deep Reinforcement Learning and Imitation Learning. *IEEE Internet of Things Journal* 8, 5 (2021), 3576–3586. https://doi.org/10.1109/JIOT.2020.3025015

[10] Arshad A. Hussein, Adel Al-zebari, Naaman M Omar, Karwan Jameel Merceedi, Abdulraheem Jamil Ahmed, Nareen O. M. Salim, Sheren Sadiq Hasan, Shakir Fattah Kak, Ibrahim M. Ibrahim, Hajar Maseeh Yasin, and Azar Abid Salih. 2021. State of Art Survey for Fault Tolerance Feasibility in Distributed Systems. *Asian Journal of Research in Computer Science* (2021). https://api.semanticscholar.org/CorpusID:239059264

[11] Mohammad Manzurul Islam, Sarwar Morshed, and Parijat Goswami. 2013. Cloud computing: A survey on its limitations and potential solutions. *International Journal of Computer Science Issues (IJCSI)* 10, 4 (2013), 159.

[12] Sara Kardani-Moghaddam, Rajkumar Buyya, and Kotagiri Ramamohanarao. 2021. ADRL: A Hybrid Anomaly-Aware Deep Reinforcement Learning-Based Resource Scaling in Clouds. *IEEE Transactions on Parallel and Distributed Systems* 32, 3 (2021), 514–526. https://doi.org/10.1109/TPDS.2020.3025914

[13] Sandeep Kelkar. 2015. Challenges and Opportunities with Cloud Computing. *International Journal of Innovative Research in Computer and Communication Engineering* 3 (04 2015), 2719–2724. https://doi.org/10.15680/ijircce.2015.0304007

[14] Medha Kirti, Ashish Kumar Maurya, and Rama Yadav. 2024. Fault-tolerant allocation of deadline-constrained tasks through preemptive migration in heterogeneous cloud environments. *Cluster Computing* (05 2024), 1–28. https://doi.org/10.1007/s10586-024-04538-9

[15] Natallia Kokash. 2005. An introduction to heuristic algorithms. *Department of Informatics and Telecommunications* (2005), 1–8.

[16] Yuanjun Laili, Sisi Lin, and Diyin Tang. 2020. Multi-phase integrated scheduling of hybrid tasks in cloud manufacturing environment. *Robotics and Computer-Integrated Manufacturing* 61 (02 2020), 101850. https://doi.org/10.1016/j.rcim.2019.101850

[17] Abdeldjalil Ledmi, Hakim Bendjenna, and Sofiane Mounine Hemam. 2018. Fault Tolerance in Distributed Systems: A Survey. In *2018 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS)*. 1–5. https://doi.org/10.1109/PAIS.2018.8598484

[18] Perng-Yi Richard Ma, Lee, and Tsuchiya. 1982. A Task Allocation Model for Distributed Computing Systems. *IEEE Trans. Comput.* C-31, 1 (1982), 41–47. https://doi.org/10.1109/TC.1982.1675884

[19] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource Management with Deep Reinforcement Learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks* (Atlanta, GA, USA) *(HotNets '16)*. Association for Computing Machinery, New York, NY, USA, 50–56. https://doi.org/10.1145/3005745.3005750

[20] Sadip Midya, Asmita Roy, Koushik Majumder, and Santanu Phadikar. 2018. Multi-objective optimization technique for resource allocation and task scheduling in vehicular cloud architecture: A hybrid adaptive nature inspired approach. *Journal of Network and Computer Applications* 103 (2018), 58–84. https://doi.org/10.1016/j.jnca.2017.11.016

[21] Richard R. Muntz and Edward G. Coffman. 1970. Preemptive Scheduling of Real-Time Tasks on Multiprocessor Systems. *J. ACM* 17 (1970), 324–338. https://api.semanticscholar.org/CorpusID:18740570

[22] Camille C. Price. 1982. Task allocation in distributed systems: A survey of practical strategies. In *Proceedings of the ACM '82 Conference (ACM '82)*. Association for Computing Machinery, New York, NY, USA, 176–181. https://doi.org/10.1145/800174.809792

[23] Rakhmat Purnomo and Tri Dharma Putra. 2024. Comparative Study: Preemptive Shortest Job First and Round Robin Algorithms. *Sinkron* (2024). https://api.semanticscholar.org/CorpusID:268836588

[24] Sahil Sharma, A. Srinivas, and Balaraman Ravindran. 2023. Deep Reinforcement Learning. *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)* (2023), 1–7. https://api.semanticscholar.org/CorpusID:15294098

[25] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646. https://doi.org/10.1109/JIOT.2016.2579198

[26] Umarani Srikanth and Geetha Rg. 2023. Effectiveness Review of the Machine Learning Algorithms for Scheduling in Cloud Environment. *Archives of Computational Methods in Engineering* 30 (03 2023). https://doi.org/10.1007/s11831-023-09921-0

[27] Karan S. Sukhija, Naveen Aggarwal, and Manish Kumar Jindal. 2015. An Optimized Approach to CPU Scheduling Algorithm. https://api.semanticscholar.org/CorpusID:18368980

[28] Shreshth Tuli, Giuliano Casale, and Nicholas R. Jennings. 2021. PreGAN: Preemptive Migration Prediction Network for Proactive Fault-Tolerant Edge Computing. arXiv:2112.02292 [cs.DC]

[29] Shreshth Tuli, Shashikant Ilager, Kotagiri Ramamohanarao, and Rajkumar Buyya. 2022. Dynamic Scheduling for Stochastic Edge-Cloud Computing Environments Using A3C Learning and Residual Recurrent Neural Networks. *IEEE Transactions on Mobile Computing* 21, 3 (2022), 940–954. https://doi.org/10.1109/TMC.2020.3017079

[30] J.D. Ullman. 1975. NP-complete scheduling problems. *J. Comput. System Sci.* 10, 3 (1975), 384–393. https://doi.org/10.1016/S0022-0000(75)80008-0

[31] Moomina Waheed, Nadeem Javaid, Aisha Fatima, Tooba Nazar, Komal Tehreem, and Kainat Ansar. 2018. Shortest Job First Load Balancing Algorithm for Efficient Resource Management in Cloud. In *Broadband and Wireless Computing, Communication and Applications*. https://api.semanticscholar.org/CorpusID:69855507

[32] Meng Xu, Lizhen Cui, Haiyang Wang, and Yanbing Bi. 2009. A Multiple QoS Constrained Scheduling Strategy of Multiple Workflows for Cloud Computing. *Proceedings - 2009 IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA 2009*, 629 – 634. https://doi.org/10.1109/ISPA.2009.95

[33] Yufei Ye, Xiaoqin Ren, Jin Wang, Lingxiao Xu, Wenxia Guo, Wenqiang Huang, and Wenhong Tian. 2018. A New Approach for Resource Scheduling with Deep Reinforcement Learning. *CoRR* abs/1806.08122 (2018). arXiv:1806.08122 http://arxiv.org/abs/1806.08122

[34] Junjie Zhang, Congdi Zhang, and Wei-Che Chien. 2021. Overview of Deep Reinforcement Learning Improvements and Applications. https://api.semanticscholar.org/CorpusID:233766844

[35] Guangyao Zhou, Wenhong Tian, and Rajkumar Buyya. 2021. Deep Reinforcement Learning-based Methods for Resource Scheduling in Cloud Computing: A Review and Future Directions. *CoRR* abs/2105.04086 (2021). arXiv:2105.04086 https://arxiv.org/abs/2105.04086