APEX ACADEMIA PRESS
ELEGANCE ON EVERY RESEARCH

International Journal of Machine Learning, AI & Data Science Evolution
ISSN: 3068-5073/ Open Access / IJMLAIDSE_Y1F4A003
Apex Academia Press / https://doi.org/10.63665/ijmlaidse-y1f4a003

Original Article

Using Reinforcement Learning in AWS EC2 for Dynamic Resource Allocation

Kayode Soladeji

Ladoke Akintola University of Technology, ksoginni@student.lautech.edu.ng

Abstract

For the best performance at the best cost for cloud computing services like Amazon EC2, one has to dynamically allocate resources. Some of the traditional techniques for scaling depend on static thresholds or predictive models that cannot be updated according to changing workloads over time. We will show how one would use reinforcement learning to do real-time allocations of AWS EC2 resources. An RL agent uses an MDP for transforming the scaling problem of how to share resources into learning how to use system metrics and patterns in the workload for identifying the optimal instance type and scaling actions. We use the AWS SDK and CloudWatch metrics in implementing our system. It is able to be tried in both the lab and real-world setting. Tests have shown that our reinforcement learning agent can reduce costs without badly affecting service-level goals. Few people do this, but this approach offers superior automated scaling. Perhaps the method could do scaling of the cloud infrastructure independently.

Keywords

Reinforcement Learning (RL), Cloud Computing, AWS EC2, Dynamic Resource Allocation, Auto Scaling, Markov Decision Process (MDP), Cost Optimization, Cloud Infrastructure Management, Deep Q-Learning, Performance-Aware Scheduling.

Article History

Received: 29.09.2025

Accepted: 07.10.2025

Published: 20.10.2025

1. Introduction

A. Reasons to Optimize Cloud Resources

More people are using cloud computing these days, and that is why businesses have changed how they install and update their apps. Cloud platforms, such as AWS, grant people the ability to take computing resources at any instance in time. This allows them to build up their infrastructure to the size they need to complete their work. Even today, it is very challenging to utilize resources in a time- and cost-efficient manner. Insufficient resources provided to your apps could result in SLA breaches or slow down. Too many resources provided result in increased operating costs for you. We have reached a time when intelligent, automated resource utilization is of prime importance for increasing cloud adoptions. Only then will cloud resources be able to bring maximum economy, performance, reliability, and energy efficiency.

B. Dynamic Resource Allocation Difficulties

For many reasons, dynamic resource allocation in cloud environments like AWS EC2 is hard to achieve: workload patterns may change over time or in bursts that are hard for static rules to discern; there are many levers to pull, including where the instances will live, at what cost on demand, reserved, or spot, among others. Finally, some of the existing autoscaling solutions, such as AWS Auto Scaling, rely on very basic prediction models or set limits. None of this work well when things are hard or change in a rapid manner. The reason is that static rules find it hard to strike a proper balance between the goals of cost cutting and performance assurances. A whole raft of issues thus creates this need for smarter, more adaptive plans for managing our resources.

C. Why Reinforcement Learning and AWS EC2?

AWS EC2 is a great place to find infrastructure as a service. You can create virtual machines that have different settings, prices, and tools which will help you keep track of how they are going. Because so many businesses and researchers use it, it is the best place to test out smart automation strategies. Reinforcement learning is a type of machine learning that helps make decisions when you are not sure what comes next. The new

https://creativecommons.org/licenses/by/4.0/

paradigm of using resources is better compared to the old one. RL agents are perfect for systems that change over time, like cloud infrastructures, because, by interacting with the environment and receiving feedback, they learn how to perform better. By using reinforcement learning, along with the strong APIs and telemetry tools from EC2, you will be able to build an intelligent agent that can change how resources are used in real time. This will be helpful and save money.

B. The Paper's Contributions

It proposes a novel application of reinforcement learning to the dynamic allocation of resources in AWS EC2 environments. First, to get a better understanding of how the cloud infrastructure is doing at any moment in time and to understand what the agent can do, we model the allocation problem as a Markov Decision Process (MDP). We have developed a deep reinforcement learning agent that interfaces with AWS in real time to learn how to select and scale resources. The key contributions of our work are: (1) formalizing an MDP for managing EC2 resources; (2) integrating an RL agent with AWS via its CloudWatch APIs and SDKs; (3) a thorough comparison of the RL methodology against traditional autoscaling methods; and (4) insights into real-world problems and performance gains concerning RL-based allocation strategies. This work also sets the foundation for future research in autonomous infrastructure management and showcases the capability of reinforcement learning in optimizing the efficiency of cloud

2. Background and Related Work

A. Overview of Scaling Mechanisms and AWS EC2 Instance Types

Each of these instance types differs in the amount of CPU, memory, storage, and network available on them. Some of the types of instances that are offered include the General Purpose (t3, m5), Compute Optimized (c6g), and Memory Optimized (r5). Every instance can host websites, run high-performance computing, and a lot more. There are three ways EC2 can be paid for: on-demand instances, reserved instances, and spot instances. The extra space open on a spot instance is considered free space; you can, however, stop them. If you promise to use reserved instances, you save money. Elastic Load Balancing and Auto Scaling Groups (ASGs) are provided to help your business grow. These add or remove instances on their own based on rules that have been set. But these systems are not flexible enough for the complicated workloads because they usually react to changes and make use of linear forecasts or set thresholds

B. Conventional Techniques for Allocating Resources

Traditional cloud resource allocation algorithms are of three major types, namely: rule-based, threshold-based, and prediction-based. Rule-based systems are based on "if-then" logical statements such as "scale out if CPU > 70%." Threshold-based policies use measures like utilization of CPU or memory to optimize something but always behave the same and are often badly configured. Predictive auto-scaling applies statistical or machine-learning algorithms on historical data to predict the amount of work to be done in the near future. Predictive approaches may be slightly flexible but work fine when the amount of work and resources is unchanging. This does not apply if the amount of work changes fast or is not stable. Due to the above issues, old-fashioned ways of obtaining cheap elastic resources no longer work

C. Introduction to Reinforcement Learning (RL) Basics

Reinforcement learning can be described as a technique that enables an agent to learn decision-making through interaction with the environment. The agent observes what is currently happening, takes an action, and gets a reward for how good that action turned out to be. Finally, the agent generates a policy that maps the optimal actions for every state. RL does best when things are hard to understand and composed of many parts-most of which cannot be viewed simultaneously. Many such systems operate in the cloud. Popular reinforcement learning algorithms include PPO, DQN, Q-learning, and Policy Gradient. Reinforcement learning helps systems arrive at proper utilization and scaling of cloud resources through trial and error to determine the best approach. There doesn't necessarily need to be any hard-and-fast rules or pre-defined ways to do things

D. Previous Research ML/RL Applications in Cloud Computing

Researchers are beginning to actively investigate the role of machine learning and, more specifically, reinforcement learning for cloud resource management. Several have applied deep learning to predict resource utilization over time, and others have used supervised learning approaches to predict the load a piece of work will require. More recently, a spate of frameworks has been developed that uses RL to schedule and orchestrate containers and autonomously scale VMs in energy-efficient ways. CloudRL, AutoScaleRL, and several others are just a few recent examples. These approaches have shown promising economic and system efficiency benefits. On the other hand, few substantial works show how to connect directly to AWS EC2. Most of them work just with specific types of workload or simulation environments. We try to fill this gap by applying and testing an RL-based solution in a real AWS environment

3. Problem Formulation

A. Identify a Markov Decision Process (MDP) as the resource allocation problem.

We model the dynamic resource allocation problem as a Markov Decision Process to facilitate reinforcement learning. A Markov Decision Process consists of five components: S, A, P, R, and γ. S denotes the set of all states describing what the system is doing in the moment. A is the list of everything the agent can do. P is the function describing the probability of something happening. Rewards are given by a function called R. The discount rate on prices is given by γ. In an AWS EC2 environment, an agent decides on how the resources should be shared. So, the agent takes an action, say, instance type or instance size, and then observes the current state of the system: workload, instance type, and performance metrics-and it gets a reward, for instance, saving money, or punishment, for instance, SLA breach. This formalization allows RL algorithms to find, over time, the best possible policies. State Space: how often it has been used, what kinds of instances it has, how busy it is right now, and other things like that. It is due to the fact that the state space possesses all the information the RL agent needs for smart choices. This includes, but is not limited to, up-to-the-minute information like the number of instances of EC2 running, the type of instances running, the workload characterization (request rate), performance indicators from the past, and perhaps even spot instance costs. The state can even learn things from the SLA, such as response time or number of errors. The RL algorithm needs a state representation that should be small enough for it to work fast but large enough to provide the needed information for making good decisions.

B. Scale Up/Down, Instance Type Switch, Spot/Reserved/On-Demand Selection Action Space

The set of actions the RL agent can take at each decision point is defined by the action space. In the perspective of EC2, this comprises:

- Increasing or decreasing the quantity of occurrences
- Modifying instance types, such as c6g.large from t3. medium
- Altering pricing schemes, like choosing a spot instance rather than an on-demand one
- Using load forecasts to determine when to start or stop instances

The degree of control granularity determines the action space's complexity. A continuous action space might enable more subtle control, whereas a discrete action space might specify a fixed set of scaling alternatives. To reap long-term rewards, the agent must learn what to do in each state.

C. Reward Function: Performance Utilization, SLA Adherence, and Cost-Efficiency

The reward function is a great way of learning: it really tells the agent how good a group of actions and states are. In this paper, the designed reward function tries to balance three major objectives that involve cost reduction for running the business, meeting SLAs, and improving resource utilization. For example, an agent may receive a reward for using spot instances to save costs. However, if some performance metrics-such as a too high error rate or a too long response time-fall below SLA levels, the agent will get into trouble. The reward may also vary depending on how much CPU and memory you give. One needs to think hard about how to design a good reward function that teaches the agent to appreciate long-term performance and efficiency over short-term gains.

4. Reinforcement Learning Model

A. Selection of RL Algorithm (e.g., Deep Q-Learning, A2C, PPO, DQN, etc.)

To make our method work, the most important thing is picking the right reinforcement learning algorithm. We look at DRL algorithms that can be used under conditions of continuity, randomness, and delayed rewards. We also consider the hardness and high-dimensionality in allocating resources to the environment. Two of the most viable alternatives when frequent changes in settings are needed are PPO and A2C. If the number of choices is limited, DQN will be a good choice. PPO serves us well because it is very stable, can run across episodes, and fits well for any systems that are cloud-hosted. The clipped goal function of PPO prevents the policy updates from going too far, which means performance is less likely to drop tremendously during training. That is why PPO would serve very well for the frequently changing, highly variable-pattern EC2 workloads.

B. The Agent's Architecture

This paper proposes a reinforcement learning agent with two deep neural networks: the policy network and the value network. The value network will find out how much a certain state is worthwhile the policy network makes a list of possible actions and how likely they are to happen based on the current situation. The PPO method will enable both networks to learn at the same time. The agent takes in features such as memory and CPU usage, network throughput, instance settings, and prices in the spot market that have been normalized. These features go through fully connected layers using activation functions such as Tanh or ReLU to create hidden representations. The policy network would dictate the processes responsible for instance selection and scaling. By trial and error, the agent learns to make choices which will be better for the environment in the long term by saving money at the same time.

C. AWS Environment Integration (e.g., CloudWatch Metrics, boto3)

The system is hooked up to CloudWatch, which is AWS's way of monitoring, and the AWS SDK for Python, boto3. This will enable the RL agent to communicate with the AWS infrastructure in real time. CloudWatch gives you time-series data related to the health of your system, CPU utilization, network traffic, disk performance, and more. These figures are just what the agent needs to perceive how things are going. Also with Boto3, the agent will be able to select between spot and OnDemand pricing models. They will also start, stop, and modify EC2 instances and edit settings for groups that automatically scale. Such close integration means that the choices that the agent makes can actually be used and tested in a cloud environment ready for production. While training, we make use of IAM roles with limited privileges to keep things safe and in check. This keeps people from doing things that could be risky.

D. The Real-Time Environment vs Simulation

We build both real and fake places to test and practice in. The simulation environment accelerates training by employing fake cost and performance models and shows how AWS works by playing back workload traces. It allows for easier and quicker changes in and testing of settings, hyperparameters, and reward functions of an algorithm. No simulated system, however, can include all that is happening in reality, for example, instance startup time, instability of the spot market, and many others. We also deploy the trained agent in a real-time AWS testbed to be able to test it with real EC2 APIs, metrics, and infrastructure. Our approach employs real-world deployment together with simulation to make sure the final model will be strong and useful.

5. Implementation

A. Description of the Experimental Setup

There are several types of EC2 instances running in different locations and availability zones within the experiment; thus, you can create an AWS account just for this purpose if you want. We rely mostly on ASGs to make sure that things are easy to manage. We consider this from the point of view of our RL-based agent. There is a dedicated monitoring module in the environment that makes sure choices by the agent, system metrics, and cost-related information have a single location. Then you can go back and review it. An EC2 machine whose Docker and boto3 run, which occasionally polls CloudWatch, talks to the AWS environment on behalf of the agent. You will easily find every instance, each one tagged, so finding and learning about it is fairly trivial. Now, the policy network needs to learn to perform a certain amount of real or fake work in each part of the experiment. Every time step, the state changes, and rewards get written down.

B. Creating Workloads or Using Actual Traffic Traces

We will consider real-world traces of traffic and synthetic workload generators to test the system under a range of real-world situations. Synthetic workloads are employed as a driver for web applications to behave like real-world ones in emulating random changes, sudden spikes, and daily patterns occurring. In this way, one can learn what the RL agent can do in order to manage and adapt in difficult but controlled scenarios. We have cloud workload traces available from open benchmarks such as Google Cluster Data and Alibaba Cluster Traces. These show how well the learned policy applies to real-world situations by putting the agent into challenging, noisy, unstable conditions:

C. Libraries and Tools (such as AWS SDK, Stable Baselines3, and OpenAI Gym)

We use a lot of new libraries to make training and development faster. Modifications to the OpenAI Gym interface allow most reinforcement learning frameworks to work with our cloud environment. We perform our training with the library Stable Baselines3, which has already implemented many RL methods, including PPO. We visualize training with Tensor Board, while PyTorch allows us to implement our own models. The boto3 library helps in managing EC2 instances, fetching CloudWatch metrics, and sending commands to other AWS services such as the EC2 Pricing API, EC2 Spot Fleet, and Auto Scaling. With this set of tools, one can get started quickly, reproduce the same results over again, and connect to AWS with ease.

D. The Process of Training and Evaluation

It means that in training, this agent goes through the same episodes again and again. At the core, this agent learns from its mistakes, and based on what it thinks are good rewards, it changes the rules and interacts with the world. Every time it plays, it gets paths. There are two states, two actions for each one, two rewards, so with this, PPO can change the model and find policy gradients. We also apply experience normalization for stability and entropy regularization to cause people to try new things. The trained and frozen policy is tested both in real AWS environments and simulated ones which include unseen workloads. We check the RL agent against set standards by looking at the set of evaluation metrics. Changes in hyperparameters are done using a grid search, and we run the tests many times to ensure statistical validity.

6. Evaluation and Results

A. Metrics: CPU/Memory Utilization, Latency, SLA Violations, Cost Savings

We will use a range of important performance metrics to evaluate the method of reinforcement learning. By summing the total cost of EC2 usage coming from the policy of the RL agent and comparing that to the cost of setting things up by hand or auto-scaling, you can get an idea about how much money you can save. The better the decisions are that are made about provisioning, the higher the average CPU and memory usage will be, showing how well resources are used. We count SLA breaches when throughput, error rates, or application response times exceed or go below what is acceptable. We will also consider latency data such as average and 95th percentile response time to find out how people feel about the service based on how we make use of resources. All these signs work in collaboration to let you know if the RL agent is doing a good job and not costing you too much.

B. Baseline Comparisons: Predictive Models, Heuristic Rules, and AWS Auto Scaling

We compare the performance of our RL agent against several baseline methods to gain insight into the efficacy of our agent. One is AWS Auto Scaling, which uses step-scaling based either on static thresholds or goals being tracked. The second one is simple and involves rules and logic like "scale out if CPU > 70%" and "scale in if CPU < 30%." The third baseline comprises predictive models using time-series analysis, such as ARIMA or LSTM, to estimate how much to scale based on expected incoming work. That of course makes sense because all baselines were run in the same AWS environment and with the same workload traces. The RL agent always saves more money, better utilizes resources, and meets SLAs better than the baselines, especially when workloads change a lot.

C. Examining Observed Enhancements or Restrictions

Our test shows that the reinforcement learning agent makes it much easier to track cloud resources by finding a balance between cost and performance and can quickly adapt to changes in the workload. The agent independently learns to scale up when it looks like traffic will go up and to use spot instances when the risk is low. That's a far cry from methods that never change. But you do have to do some work. First, the initial training is extremely time and cost-intensive, especially if it occurs in production. The agent may also struggle with weird but important edge cases, such as when a type of instance suddenly becomes unavailable or a service goes down without warning. You will also have to know a lot about the subject matter to be able to tune the hyperparameters and the reward function. There are problems, but overall, performance improvements demonstrate RL as a good way to improve cloud infrastructure.

7. Challenges and Limitations

A. Cost of Computation and Training Time

The biggest challenge with the use of reinforcement learning in managing AWS EC2 resources is that training and performing the necessary mathematical computations take a lot of time and are financially burdensome. Generally, RL algorithms take a number of episodes to converge to an efficient and stable policy in very complex and high-dimensional environments, as is the case with cloud infrastructures. Training in an actual AWS environment is too expensive because one has to run the instances and get performance data immediately. Even in simulations, calculating the right prices, latency, and workload behavior is costly. And it takes longer and costs more money when you have to perform model selection and hyperparameter tuning. All these challenges show how important it is to save money by using good training methods such as transfer learning or pretraining in a simulation.

B. Managing Unexpected Increases in Workload

While reinforcement learning agents can learn to adapt to patterns over time, they have a hard time with sudden, unexpected increases in workload such as flash sales, viral content, or denial-of-service attacks. The RL agent may take more than one decision interval to make the right choice, especially for those who are trained on normal workloads. But these spikes need fast resource allocations. This latency would cause apps to run slowly or break SLAs temporarily. Reward shaping and putting people in fake spike situations during training might help, but being able to respond quickly in real life is still a problem. You almost always need backup systems that are fast-acting when you put something into production. Examples of such systems are emergency buffers and rule-based overrides.

C. Unpredictability of AWS Pricing (Spot Interruptions, etc.)

AWS prices fluctuate wildly, making it difficult to even guess what they might be at any particular point in time, especially across the spot instance market. This is a problem unto itself: You can shut down at-will spot instances and their prices increase or decrease according to demand versus supply. They cut expenses dramatically, yet they aren't always available-which can greatly slow things down. RL agents may learn, as part of cost-cutting efforts, to pay more attention to those spot instances. Unless properly handled, interruptions could cause an outage of your resources and applications. Since spot instances constantly change, it is hard to determine how they will behave during training. To some extent, RL agents can learn to balance cost with reliability; however, they are still in need of much more problem-solving ability for very critical applications.

D. Generalization and Scalability Across Workloads and Regions

Two important challenges for RL agents in large-scale diverse cloud environments are scalability and generalization: teaching an agent to handle a certain variety of application workload in one location may not generalize well to other workloads, instance types, or locations. If the latency, cost, or resource availability changes, the agent must be retrained or otherwise improved in order to remain useful. Increasing numbers of services, dependencies, and autoscaling groups increase the size of action and state spaces, which could make learning harder. Modular, hierarchical, or multi-agent systems may find it easier to address these challenges. However, generalization remains one of the thorny issues when the RL-based resource management system needs to work in various cloud settings.

8. Future Work

A. Using Multi-Tier Architectures with Multi-Agent RL

That Have More Than One Level Most of the cloud services are using multi-tier architecture these days. The typical examples may be caches, databases, web servers, and application servers. All these have varying resource requirements and may scale differently. On the other hand, multi-agent reinforcement learning is a domain that still has much room for research contributions. In MARL, different agents take responsibilities at various levels or services. They can collaborate to maintain the performance of a system. It helps you get stronger, stay focused, and improve. The agents may cooperate in making decisions in order to enhance the service from start to end without exceeding their budget. They may discuss the state or use the learned rules to do this.

B. Generalization through Federated or Meta-RL

Learning Poor generalization across diverse contexts may be handled in future studies by federated reinforcement learning or meta-reinforcement learning techniques. Federated RL enhances generalization while protecting privacy and operational constraints, enabling multiple agents to train concurrently on diverse workloads or in various locations. Subsequently, they typically make decisions without sharing any raw data between them. Meta-RL, on the other hand, aims to create agents that can adapt fast to new situations without having to learn everything from scratch-that is, to teach them "how to learn." These methods give you more leeway as far as how you can handle cloud workloads, which can change over time and from one technical or organizational setting to another.

C. Expanding to Edge or Hybrid Cloud Situations

It would be interesting to see whether the RL-based model for resource allocation would work for edge or hybrid cloud computing. A hybrid cloud computing environment is when private data centers and public cloud providers collaborate. This makes moving things around, staying compliant, and finding data much more difficult than it needs to be. There is an increasing demand for supporting real-time decisions with low power consumption in edge computing due to its proximity to the end users with only a limited number of resources. By equipping RL agents to better perform planning tasks with regard to where to place an entity and when to act in these contexts, the proposed system will become much more useful and powerful. You achieve this through the use of light agents, training in different locations, and the breaking up of already known information.

9. Conclusion

This paper proposes a novel reinforcement learning-based approach to dynamic resource allocation in AWS EC2 environments. We model the problem as a Markov Decision Process, allowing a reinforcement learning agent to identify an optimal scaling strategy and instance selection. Unlike typical rule-and-guess-based approaches, we tested our approach on both synthetic and real workload traces. It works seamlessly with AWS and uses boto3 and CloudWatch. Also, it leverages deep reinforcement learning methods such as PPO. The results clearly depict that the RL agent can save tremendous amounts while meeting SLAs and resources being properly utilized. Issues to be fixed now are those of generalization, training cost, and how to deal with edge cases, which may involve instances like a spot instance going down. On the other hand, performance improvements suggest that reinforcement learning may be applied to enable the cloud systems to act autonomously. This is the work that lets you build smart cloud systems that can self-correct if something goes wrong or changes.

10. References

[1] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347.

[2] Mao, H., Alizadeh, M., Menache, I., & Kandula, S. (2016). Resource Management with Deep Reinforcement Learning. *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets-XV)*.

[3] Xu, Y., Rao, L., & Liu, X. (2012). On balancing energy consumption and end-to-end delay in cloud-based data centers. *IEEE Transactions on Parallel and Distributed Systems*, 24(6), 1234–1244.

[4] Amazon Web Services. (2024). Amazon EC2 Auto Scaling Documentation**.** Retrieved from: https://docs.aws.amazon.com/autoscaling/

[5] Google Cloud Platform. (2011). Google Cluster Data Trace. Retrieved from: https://github.com/google/cluster-data

[6] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster Computing with Working Sets. *USENIX Conference on Hot Topics in Cloud Computing*.

[7] Chen, X., Ren, S., & Cheng, X. (2018). Performance-aware Virtual Machine Placement in Clouds: A Reinforcement Learning Approach**.** *IEEE Transactions on Cloud Computing*.

[8] OpenAI. (2020**)**. Gym: A Toolkit for Developing and Comparing Reinforcement Learning Algorithms. Retrieved from: https://github.com/openai/gym

[9] Liang, H., Lakshmanan, V., & Bhat, V. (2021). Autoscaling Kubernetes Applications Using Reinforcement Learning. *ACM Symposium on Cloud Computing (SoCC)*.

[10] Stable Baselines3. (2022). Reliable implementations of reinforcement learning algorithms in PyTorch. https://github.com/DLR-RM/stable-baselines3

[11] He, Y., Guo, Y., Li, D., & Xu, C. (2020). Learning-based Auto-scaling for Web Applications in Clouds. *Future Generation Computer Systems*, 107, 501–512.

[12] Jiang, J., Lan, T., Ha, S., & Chiang, M. (2012). Joint VM Placement and Routing for Data Center Traffic Engineering**.** *IEEE INFOCOM*.

[13] AWS SDK for Python (boto3). (2024). Documentation and Code Samples. https://boto3.amazonaws.com/