Tạp chí Khoa học Đại học Huế: Kỹ thuật và Công nghệ; pISSN 2588-1175 | eISSN 2615-9732

Tập 134, Số 2A, 2025, Tr. 143–163; DOI: 10.26459/hueunijtt.v134i2A.7972

ERDQ-LEARNING: MÔ HÌNH HỌC TĂNG CƯỜNG TÍCH HỢP HPA PHÂN CHIA TỶ LỆ POD SERVERLESS TRONG KUBERNETES

Đặng Thanh Chương[1]*, Nguyễn Duy Sơn[1], Nguyễn Đặng Duy Trinh[2]

[1] Trường Đại học Khoa học, Đại học Huế, 77 Nguyễn Huệ, Thành phố Huế, Việt Nam
[2] Trường Đại học Y Dược, Đại học Huế, 06 Ngô Quyền, Thành phố Huế, Việt Nam

Tóm tắt. Trong môi trường điện toán đám mây không máy chủ (serverless), việc duy trì hiệu suất và chất lượng dịch vụ trước sự biến động của lưu lượng truy cập là một thách thức quan trọng. Cơ chế tự động phân chia tỷ lệ tài nguyên truyền thống - Horizontal Pod Autoscaler (HPA) vẫn được xem là một cơ chế cơ bản và cần thiết trong Kubernetes. Tuy nhiên, HPA chỉ sử dụng ngưỡng CPU cố định để điều chỉnh số lượng Pod, điều này dẫn đến phản ứng chậm hoặc điều phối kém hiệu quả trong môi trường có tải động. Bài báo này đề xuất một mô hình học tăng cường ERDQ-learning trên cơ sở cải tiến thuật toán Q-learning kết hợp 2 kỹ thuật Experience Replay và Double Q-learning (ERDQ-learning) nhằm tối ưu hóa quá trình phân chia tỷ lệ Pod serverless được triển khai mô phỏng trên nền tảng Kubernetes. Mục tiêu của thuật toán ERDQ-learning là nhằm điều chỉnh ngưỡng kích hoạt CPU của HPA một cách linh hoạt theo thời gian thực, dựa trên các trạng thái hệ thống bao gồm mức sử dụng CPU, số lượng pod hiện tại, độ trễ và ngưỡng đang áp dụng. Kết quả thực nghiệm trên môi trường mô phỏng Kubernetes với lưu lượng biến đổi cho thấy mô hình ERDQ-learning cải thiện đáng kể khả năng thích ứng của hệ thống, giảm độ trễ dịch vụ và tối ưu hóa mức sử dụng tài nguyên so với cơ chế HPA truyền thống. Phương pháp đề xuất thể hiện tính khả thi và hiệu quả cao trong việc điều phối tài nguyên động cho các hệ thống serverless hiện đại.

Từ khóa: Serverless Cloud, HPA, Pod, Reinforcement Learning, Double Q-learning

ERDQ-learning: Reinforcement learning model integrating HPA for scaling serverless pods in Kubernetes

Dang Thanh Chuong[1]*, Nguyen Duy Son[1], Nguyen Dang Duy Trinh[2]

[1] University of Science, Hue University, 77 Nguyen Hue Street, Hue City, Vietnam
[2] University of Medicine and Pharmacy, Hue University, 06 Ngo Quyen Street, Hue City, Vietnam

Abstract. In serverless cloud computing environments, maintaining performance and Quality of Service (QoS) under fluctuating traffic conditions remains a significant challenge. The traditional Horizontal Pod Autoscaler (HPA) in Kubernetes, while fundamental, relies solely

** Liên hệ: dtchuong@hueuni.edu.vn*

Ngày nhận bài: 02–07–2025; Ngày nhận đăng: 16–07–2025

on fixed CPU utilization thresholds to scale Pods, often resulting in delayed responses or inefficient orchestration in dynamic load scenarios. This paper proposes an enhanced reinforcement learning approach, ERDQ-learning, which integrates Experience Replay and Double Q-learning to optimize the scaling behavior of serverless Pods. The ERDQ-learning model dynamically adjusts HPA's CPU activation threshold in real time, using system state parameters such as CPU utilization, current Pod count, response latency, and the existing threshold. Experimental evaluations in a simulated Kubernetes environment with varying traffic patterns demonstrate that ERDQ-learning significantly enhances system adaptability, reduces latency, and improves resource efficiency compared to the traditional HPA. These results highlight the feasibility and effectiveness of the proposed model for intelligent resource orchestration in modern serverless systems.

Keywords: Serverless Cloud, HPA, Pod, Reinforcement Learning, Double Q-learning

1 Giới thiệu

Trong những năm gần đây, kiến trúc điện toán đám mây không máy chủ *(serverless computing)* đã nổi lên như một mô hình triển khai ứng dụng linh hoạt, cho phép các nhà phát triển tập trung vào logic chức năng mà không cần quản lý trực tiếp hạ tầng máy chủ. Mô hình này hỗ trợ thực thi theo sự kiện (*event-driven*) và tự động phân bổ tài nguyên dựa trên nhu cầu thực tế, góp phần tối ưu chi phí vận hành và nâng cao khả năng mở rộng dịch vụ [1] [2]. Khác với các mô hình điện toán đám mây truyền thống, nơi tài nguyên được phân bổ cố định, đám mây không máy chủ khai thác khả năng phân bổ tài nguyên động, cho phép giảm thiểu số lượng phiên bản (instance) hoạt động khi không có nhu cầu. Kiến trúc này thường được hiện thực hóa dưới dạng *Functions-as-a-Service* (FaaS), trong đó các chức năng ứng dụng được triển khai dưới dạng các hàm độc lập, vận hành trong môi trường *container* hóa [3]. Thay vì triển khai toàn bộ một nền tảng hoặc ứng dụng phức tạp, FaaS chỉ cần triển khai các hàm đơn lẻ – các thành phần cốt lõi của ứng dụng, được kích hoạt theo yêu cầu. Các hàm này được tải động vào các *containers*, có thể mở rộng song song một cách hiệu quả, và không yêu cầu can thiệp ở cấp hệ điều hành. Hạ tầng thực thi được điều phối tự động bởi các nhà cung cấp dịch vụ đám mây nhằm tối ưu hóa hiệu suất sử dụng tài nguyên. Nhờ đặc tính này, *serverless* đã trở thành một mô hình triển khai tiềm năng trong các môi trường điện toán biên không máy chủ (*serverless edge computing*). Mô hình mang lại nhiều lợi ích cho cả nhà phát triển và doanh nghiệp, đặc biệt trong việc giảm chi phí, tự động hóa quy trình triển khai, và tối ưu tài nguyên. Một trong những lợi ích quan trọng nhất là khả năng tự động phân chia tỷ lệ (*auto-scaling*), cho phép ứng dụng tự động mở rộng hoặc thu hẹp quy mô theo tải thực tế mà không cần cấu hình thủ công, đảm bảo hiệu năng ổn định ngay cả khi lưu lượng tăng đột biến [4].

Kubernetes (K8s) là một nền tảng mã nguồn mở được thiết kế để tự động hóa việc triển khai, quản lý và mở rộng các ứng dụng container hóa. Được phát triển bởi Google và hiện nay được duy trì bởi *Cloud Native Computing Foundation* (CNCF), Kubernetes đã trở thành nền tảng tiêu chuẩn trong quản lý hạ tầng *container* nhờ vào tính linh hoạt và khả năng vận hành mạnh mẽ

trong môi trường phân tán. Trong hệ sinh thái Kubernetes, *Pod* là đơn vị triển khai cơ bản nhất, chứa một hoặc nhiều *container* hoạt động cùng nhau và chia sẻ tài nguyên hệ thống. Mỗi *Pod* đại diện cho một phiên bản ứng dụng đang chạy trong cụm, là đơn vị mà Kubernetes dùng để thực hiện các chức năng như mở rộng, phục hồi và cân bằng tải. Mô hình serverless triển khai trên Kubernetes bản chất là quá trình tự động hóa triển khai và điều chỉnh quy mô của các hàm chức năng (*function*), được hiện thực hóa thông qua kiến trúc FaaS (*Function-as-a-Service*), tức là mỗi hàm được đóng gói thành một container và triển khai dưới dạng một Pod độc lập trong cụm Kubernetes (chúng tôi gọi là FaaP (Function-as-a-Pod), kết hợp với các cơ chế autoscaler và thành phần điều phối sự kiện, định tuyến [5] [6]. Cách tiếp cận này tận dụng hệ sinh thái phong phú của Kubernetes để hiện thực hóa tính năng serverless trong môi trường container hóa linh hoạt và kiểm soát được.

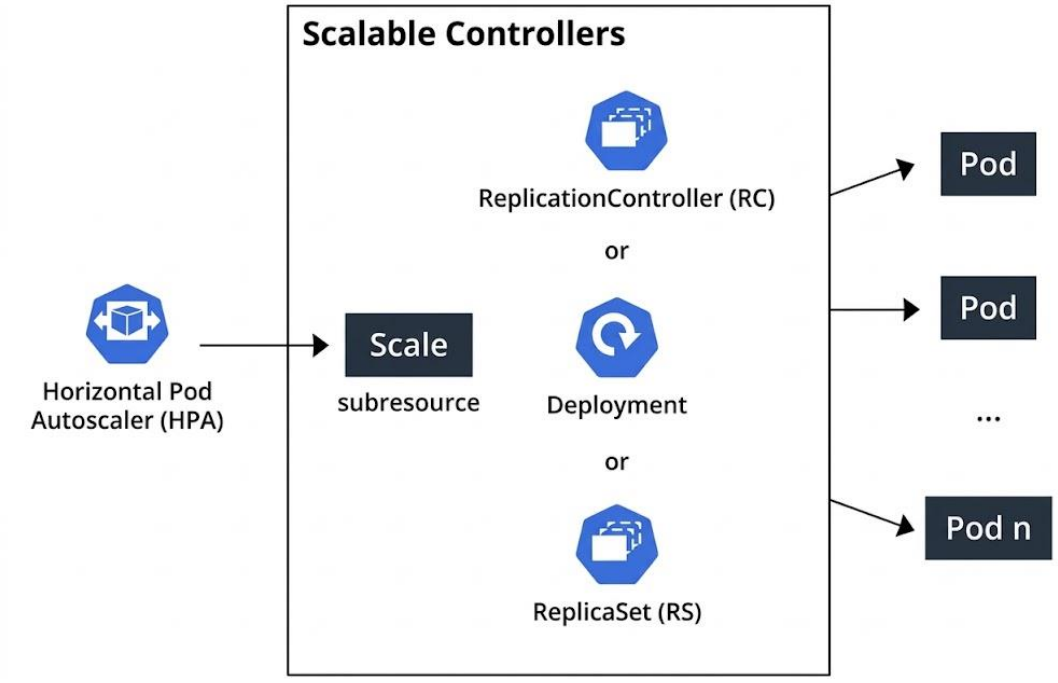
Auto-scaling là một kỹ thuật trong điện toán đám mây và quản lý tài nguyên, được hiểu là quá trình tự động phân chia tỉ lệ tài nguyên (như máy chủ, container hoặc Pod) của hệ thống để đáp ứng nhu cầu thực tế của ứng dụng hoặc dịch vụ. Mục tiêu chính của auto-scaling là đảm bảo hệ thống đáp ứng tốt với các thay đổi về tải và tối ưu hóa việc sử dụng tài nguyên nhằm giảm chi phí vận hành. Auto-scaling là một thành phần quan trọng trong các kiến trúc hiện đại như microservices, containerization và serverless computing. Kubernetes Autoscaler là một thành phần quan trọng trong hệ sinh thái Kubernetes, giúp tự động phân chia tỉ lệ hoặc thu nhỏ các tài nguyên như pod, node để tối ưu hóa hiệu suất và chi phí vận hành. Đây là một cơ chế cốt lõi để đảm bảo rằng các ứng dụng container hóa trong Kubernetes có thể đáp ứng tải biến động mà không cần can thiệp thủ công.

Horizontal Pod Autoscaler (HPA) là một cơ chế phân chia tỉ lệ ngang trong Kubernetes, cho phép tự động điều chỉnh số lượng Pod của một workload (ví dụ: Deployment, ReplicaSet, hoặc StatefulSet) dựa trên các chỉ số quan sát được (observed metrics), như được chỉ ra trong Hình 1 [8]. Thuật toán HPA hoạt động như một vòng lặp điều khiển (control loop), nhằm duy trì hiệu suất ứng dụng ổn định trước sự thay đổi của tải hệ thống, đồng thời tối ưu hóa việc sử dụng tài nguyên [7]. Dựa trên các chỉ số quan sát được từ hệ thống, cơ chế chia tỷ lệ sẽ tính toán số lượng Pod mong muốn $d_{\text{desired}}(t)$ bằng cách áp dụng giải thuật HPA theo phương trình [7-8]:

$$d_{\text{desired}}(t) = \left\lceil d_{\text{on}}(t) \frac{\bar{\rho}(t)}{\rho_{\text{target}}} \right\rceil \quad (1)$$

Trong mỗi vòng lặp điều khiển, HPA sử dụng dữ liệu đầu vào bao gồm mức sử dụng CPU trung bình $\bar{\rho}(t)$, số Pod hiện tại $d_{\text{on}}(t)$ cùng với ngưỡng mục tiêu ρ_{target}, để xác định giá trị $d_{\text{desired}}(t)$. Sau đó, thuật toán áp dụng hàm ràng buộc để giới hạn số Pod trong khoảng từ M (số Pod được đặt trước tối thiểu) đến L (số Pod tối đa của hệ thống), kết quả thu được số lượng Pod mong muốn được khởi tạo. Giá trị $d_{\text{desired}}(t)$ sẽ được so sánh với trạng thái hiện tại để quyết định việc phân chia tỉ lệ (scale out/scale in) số lượng Pod. Mặc định, vòng điều khiển này thực thi mỗi

15 giây; tuy nhiên, khoảng thời gian này có thể được điều chỉnh linh hoạt thông qua tham số τ trong thuật toán nhằm phù hợp với đặc thù tải và mục tiêu vận hành của hệ thống serverless.



Hình 1. Minh họa cơ chế HPA hoạt động trong Kubernetes [7]

Thuật toán 1 – Điều chỉnh số Pod bằng HPA mặc định [7-8]
Đầu vào: ρ_{target}, M (Số Pod tối thiểu), L (Số Pod tối đa), T(thời gian mô phỏng), τ (chu kỳ lấy mẫu) **Đầu ra:** $d_{\text{desired}}(t)$

Ý tưởng:
1. $t \leftarrow 0$
2. While ($t \leq T$) do
{
3. $\bar{\rho}(t) \leftarrow Metrics.Get(CPU\_average)$
4. $d\_on(t) \leftarrow Metrics.Count(Pod)$
5. $d_{\text{desired}}(t) \leftarrow [\, d_{on(t)} \times \frac{\bar{\rho}(t)}{\rho_{target}}]$
6. $d_{\text{desired}}(t) \leftarrow min(\, max(\, d_{\text{desired}}(t), M\,), L\,)$
7. $Controller.Scale(d\_desired(t))$
8. $Wait(\tau)$
9. $t \leftarrow t + \tau$
}

Khác với cách tiếp cận thay thế hoàn toàn cơ chế điều phối tài nguyên, nghiên cứu này đề xuất một mô hình phối hợp giữa học tăng cường và HPA. Cụ thể, tác tử học tăng cường (RL agent) không trực tiếp quyết định số lượng Pod, mà điều chỉnh động ngưỡng sử dụng CPU mục tiêu (threshold) của HPA. Việc điều chỉnh này tác động đến quyết định phân chia tỉ lệ số lượng Pod của HPA, tức là điều chỉnh số lượng Pod được khởi tạo hay kết thúc. Trong mô hình đề xuất ở bài báo này, HPA vẫn giữ vai trò thực thi điều phối quy mô tài nguyên, đảm bảo an toàn và ổn định, trong khi RL đóng vai trò chiến lược, học từ dữ liệu quan sát được để tối ưu chính sách phân chia tỉ lệ theo thời gian thực. Cách tiếp cận này tận dụng ưu điểm của cả hai cơ chế: sự ổn định của HPA và khả năng thích nghi linh hoạt của RL trong môi trường serverless có tải biến động.

Trong bài báo này, chúng tôi mô hình hóa bài toán phân chia tỉ lệ các Pod serverless trong Kubernetes thông qua tích hợp thuật toán học tăng cường, nhằm tối ưu hóa việc điều chỉnh ngưỡng HPA theo thời gian thực một cách linh hoạt và thích nghi với tải động. Mô hình đề xuất được trình bày chi tiết trong phần III của bài báo.

Những đóng góp chính của bài báo gồm:

(1). Đề xuất mô hình tích hợp học tăng cường vào HPA K8s trên cơ sở thuật toán ERDQ-learning (được cải tiến từ thuật toán Q-learning truyền thống) nhằm tự động phân chia tỉ lệ các Pod serverless dựa trên Kubernetes, cho phép điều khiển động các ngưỡng cố định trong HPA. Mô hình cho phép tác tử học thích nghi với các đặc trưng thay đổi của hệ thống như: kích thước workload, quy mô triển khai, tốc độ phản hồi, độ phức tạp dịch vụ và các ngưỡng sử dụng tài nguyên hiện hành, từ đó đánh giá và đưa ra quyết định điều phối tài nguyên nhằm tối ưu hiệu suất và đảm bảo QoS.

(2). Thực nghiệm mô phỏng trong môi trường có lưu lượng biến động phức tạp được sinh bằng công cụ Locust, nhằm kiểm chứng hiệu quả của mô hình đề xuất so với HPA truyền thống và HPA kết hợp Q-learning. Kết quả được phân tích và trình bày trong phần IV, cho thấy mô hình ERDQ-learning đạt được sự cân bằng tốt giữa độ trễ, tài nguyên sử dụng và tính ổn định chính sách.

Nội dung tiếp theo của bài báo bao gồm: Phần II trình bày tổng quan các nghiên cứu liên quan về phân chia tỉ lệ tài nguyên trong môi trường serverless và các hướng tiếp cận học tăng cường. Mô tả chi tiết thiết kế mô hình và ERDQ-learning và phân tích bài toán phân chia tỉ lệ pod dựa ngưỡng được trình bày trong Phần III; Phần IV trình bày quá trình thực nghiệm và phân tích kết quả nhằm đánh giá hiệu quả của mô hình đề xuất. Cuối cùng là phần kết luận và hướng phát triển của bài báo.

2 Các công trình nghiên cứu liên quan

Nghiên cứu của Zhang và cộng sự [9] đã triển khai thuật toán Q-learning kết hợp cả phân chia tỉ lệ theo chiều ngang (Horizontal Scaling) và phân chia tỉ lệ theo chiều dọc (Vertical Scaling). Thử nghiệm trên một ứng dụng thương mại điện tử cho thấy phương pháp này giảm chi phí tài nguyên mà vẫn duy trì chất lượng dịch vụ (QoS). Đặc biệt, việc kết hợp hai loại phân chia tỉ lệ này giúp hệ thống đạt được sự cân bằng giữa hiệu suất và chi phí vận hành. Tương tự, Qiu và cộng sự [10] sử dụng thuật toán Proximal Policy Optimization (PPO) để phân chia tỉ lệ tài nguyên. Nghiên cứu của họ thực hiện trên cả dữ liệu tổng hợp và thực tế, nhấn mạnh vào thời gian hội tụ và độ chính xác trong việc dự đoán tải. Kết quả cho thấy PPO vượt trội hơn các thuật toán truyền thống nhờ khả năng thích nghi nhanh chóng với các biến động lớn trong tải.

Nghiên cứu của Benedetti và cộng sự [3] đề xuất một tác tử dựa trên Q-learning để điều chỉnh ngưỡng CPU trong Horizontal Pod Autoscaler (HPA) của Kubernetes và tiến hành một loạt các thử nghiệm tải trên hệ thống với các ngưỡn CPU cố định khác nhau. Họ so sánh bộ tự động điều chỉnh Q-learning và HPA dựa trên trên tài nguyên với ngưỡng cố định trong độ trễ của các ứng dụng. Kết quả thực nghiệm chỉ ra rằng RL có thể cải thiện đáng kể mức sử dụng tài nguyên trong khi vẫn duy trì độ trễ gần với điều kiện lý tưởng. Nghiên cứu của Rossi và cộng sự [11] triển khai một mạng Deep Q-Network (DQN) để tự động điều chỉnh tài nguyên, bao gồm mức sử dụng CPU, bộ nhớ của cơ sở hạ tầng và thời gian phản hồi của ứng dụng. Kết quả cho thấy rằng DQN có thể cân bằng tốt giữa hiệu suất và chi phí, mặc dù cần thêm nghiên cứu để giảm độ trễ trong các hệ thống có tải lớn.

Ngược lại với công trình của Rossi và cộng sự, Khaleq và cộng sự [12] chỉ tập trung vào các giải pháp dựa trên học tăng cường (Reinforcement Learning - RL) và thực hiện một so sánh chi tiết giữa các loại thuật toán RL khác nhau, được triển khai nhằm thay đổi ngưỡng sử dụng tài nguyên của HPA trong môi trường Kubeless. Các thí nghiệm được thực hiện với cả các ứng dụng tiêu tốn bộ nhớ (Memory-intensive) và tiêu tốn CPU (CPU-intensive) để làm rõ sự khác biệt giữa các thuật toán RL trong các loại ứng dụng khác nhau. Kết quả của nghiên cứu cho thấy hệ thống tự động phân chia tỉ lệ thông minh được đề xuất có thể cải thiện thời gian phản hồi của các microservices lên đến 20% so với các phương pháp tự động phân chia tỉ lệ truyền thống. Hệ thống cũng duy trì chất lượng dịch vụ (QoS) và đáp ứng các yêu cầu của ứng dụng thời gian thực, trong khi giảm thiểu sự can thiệp của người dùng. Điều này chứng minh rằng việc sử dụng RL có thể tối ưu hóa việc điều chỉnh quy mô tài nguyên trong các ứng dụng thời gian thực trên nền tảng đám mây.

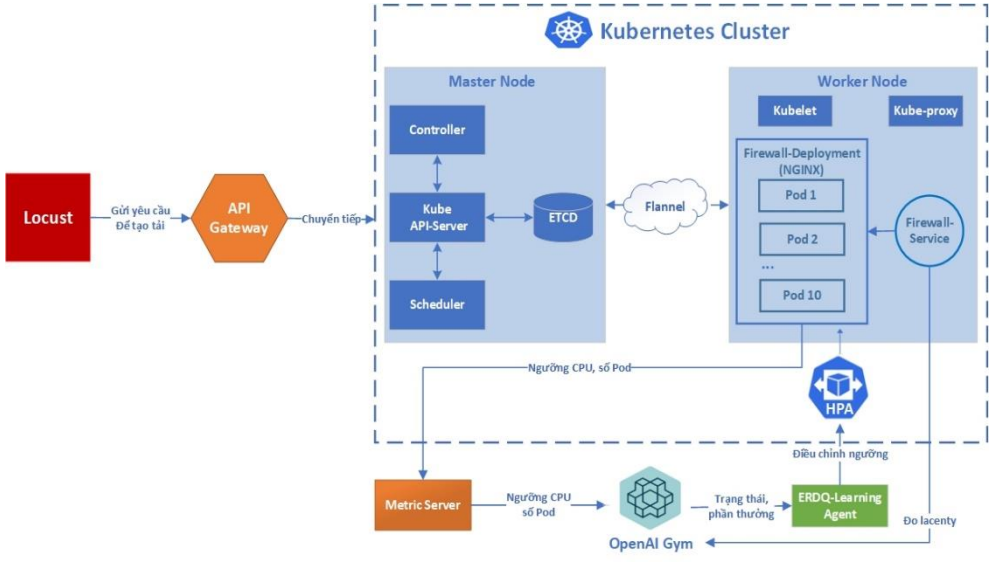
Học tăng cường đang nổi lên như một trong những phương pháp đầy hứa hẹn trong việc giải quyết các bài toán tối ưu hóa tài nguyên có tính động cao, đặc biệt trong môi trường điện toán đám mây serverless – nơi hiệu năng, khả năng thích ứng và chi phí vận hành là những yếu tố then chốt. Trong bối cảnh đó, các cơ chế quản lý tài nguyên truyền thống như HPA của

Kubernetes vẫn còn tồn tại nhiều hạn chế. Cụ thể, HPA thường dựa vào ngưỡng cố định (ví dụ: mức sử dụng CPU) để điều chỉnh số lượng Pod, điều này khiến hệ thống phản ứng chậm trước những biến động bất thường của tải công việc. Khi tải tăng đột ngột hoặc thay đổi không đều theo thời gian, việc duy trì một chính sách cố định dễ dẫn đến hai hệ quả tiêu cực: thừa tài nguyên gây lãng phí, hoặc thiếu tài nguyên dẫn đến suy giảm chất lượng dịch vụ (QoS), đặc biệt là độ trễ phản hồi. Những thách thức trên càng trở nên rõ nét hơn trong hệ thống serverless, nơi các hàm chức năng (functions) được kích hoạt theo yêu cầu và đòi hỏi hệ thống phản ứng trong thời gian gần như tức thời, các cơ chế quản lý tài nguyên trong serverless vì vậy không chỉ phải chính xác mà còn phải linh hoạt, thích nghi nhanh với các thay đổi của tải và môi trường thực thi. Nghiên cứu của Lizhong trong [13] đã đề xuất một cơ chế phân bổ tài nguyên trong điện toán đám mây không máy chủ dựa trên thuật toán chia tỷ lệ theo ngưỡng để quản lý tài nguyên một cách hợp lý. Ý tưởng phân chia tỉ lệ theo ngưỡng trong [13] cũng sẽ được chúng tôi áp dụng vào mô hình trong bài báo này.

3 Mô hình học tăng cường ERDQ-learning phân chia tỉ lệ Pod serverless dựa trên HPA Kubernetes

3.1 Mô hình hóa hệ thống phân chia tỉ lệ Pod serverless dựa trên ngưỡng

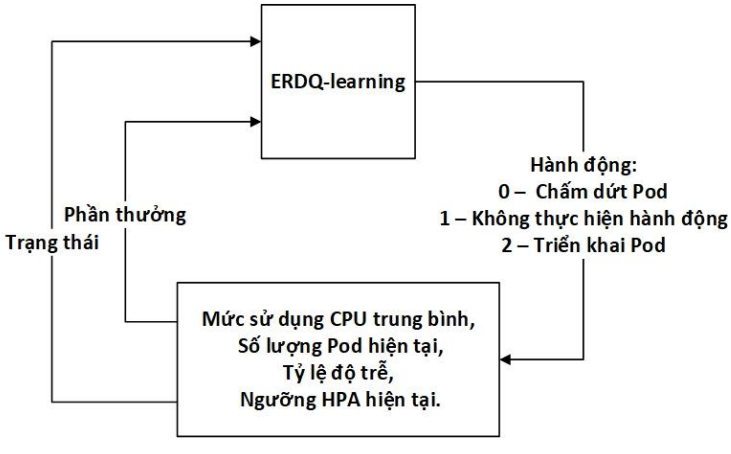
Mô hình tích hợp học tăng cường cho bài toán phân chia tỉ lệ Pod trong môi trường Serverless được chỉ ra trong Hình 2.



Hình 2. Mô hình tích hợp học tăng cường cho bài toán phân chia tỉ lệ Pod trong môi trường Serverless Cloud

Trong mô hình được xây dựng, các hàm chức năng serverless được đóng gói thành các Pod trong Kubernetes dưới dạng Deployment, như firewall-deployment trong kiến trúc hệ thống. Mỗi Pod đảm nhận việc xử lý một đơn vị chức năng serverless, phản hồi các yêu cầu được gửi đến từ người dùng thông qua API Gateway. HPA sẽ giám sát các chỉ số như mức sử dụng CPU của Pod để quyết định việc phân chia tỉ lệ (scale-out/scale-in) số lượng Pod theo thời gian thực. Trong hệ thống được triển khai, HPA được tích hợp với thuật toán học tăng cường ERDQ-learning nhằm điều chỉnh ngưỡng kích hoạt phân chia tỉ lệ một cách chủ động, thay vì sử dụng giá trị ngưỡng cố định như HPA truyền thống. Môi trường huấn luyện được xây dựng dựa trên nền tảng OpenAI Gym, một thư viện mã nguồn mở tiêu chuẩn cho phát triển và đánh giá các thuật toán học tăng cường [14]. Trong mô hình này, Gym được mở rộng để tương tác trực tiếp với cụm Kubernetes, thông qua các thành phần như Metrics Server (thu thập chỉ số CPU, số lượng Pod) và hệ thống đo độ trễ dịch vụ. Dựa trên trạng thái hệ thống hiện tại, tác tử thực hiện hành động (điều chỉnh ngưỡng HPA) và nhận về phần thưởng (reward) đánh giá hiệu quả hành động đó, với mục tiêu đạt được sự cân bằng tối ưu giữa hai yếu tố: giảm thiểu lãng phí tài nguyên và duy trì chất lượng dịch vụ (QoS).

Cấu trúc tổng thể của mô hình bao gồm bốn thành phần chính: không gian hành động (Action Space), không gian trạng thái (State Space), hàm phần thưởng (Reward Function) và tác tử học tăng cường (Reinforcement Learning Agent). Trong đó, ba thành phần đầu đóng vai trò xác định môi trường và quy tắc học tập cho tác tử, còn tác tử chịu trách nhiệm học và ra quyết định dựa trên kinh nghiệm tích lũy. Mô hình sử dụng thuật toán Q-learning kết hợp với hai kỹ thuật tăng cường là Experience Replay [15] và Double Q-learning [16] nhằm cải thiện hiệu suất hội tụ và độ tin cậy của chính sách học được. Sơ đồ huấn luyện học tăng cường theo thuật toán ERDQ-learning trong mô hình phân chia tỉ lệ Pod serverless bằng HPA trong Kubernetes được biểu diễn trong Hình 3. Chi tiết cơ chế hoạt động của mô hình trong Hình 3 sẽ được chúng tôi trình bày ở các phần tiếp theo của bài báo này.



Hình 3. Sơ đồ huấn luyện ERDQ-learning trong mô hình phân chia tỉ lệ Pod serverless bằng HPA trong Kurbernetes

3.2 Mô hình hóa bài toán dưới dạng Markov Decision Process (MDP)

Bài toán tích hợp học tăng cường nhằm điều chỉnh ngưỡng HPA trong hệ thống serverless có thể được mô hình hóa dưới dạng quá trình quyết định Markov (Markov Decision Process – MDP), bao gồm một bộ 5 thành phần chính $< S, A, P, r, \gamma >$ được định nghĩa như sau:

- ***S*: Không gian trạng thái (State Space).** Phản ánh tình trạng hiện tại của hệ thống, cung cấp thông tin cần thiết cho tác tử đưa ra quyết định. Gồm bốn tham số chính:

- CPU utilization: Đo mức sử dụng CPU của firewall-deployment, rời rạc thành 6 mức: 0%-30%, 30%-60%, 60%-90%, 90%-120%, 120%-150%, >150%. Giá trị tối đa >150% phản ánh khả năng vượt quá giới hạn CPU trong trường hợp tải cao.

- Số lượng Pod: từ 1 đến 10 (giới hạn bởi HPA), được rời rạc thành 5 mức: 1-2, 3-4, 5-6, 7-8, 9-10.

- Tỷ lệ latency/SLA: Tỷ lệ giữa độ trễ thực tế (latency) và ngưỡng độ trễ tối đa chấp nhận được theo thỏa thuận mức độ dịch vụ SLA (1 giây), rời rạc thành 6 mức: 0%-30%, 30%-60%, 60%-90%, 90%-120%, 120%-150%, >150%.

- Ngưỡng HPA hiện tại: Giá trị ngưỡng CPU target của HPA, rời rạc thành 5 mức: 20%-40%, 40%-60%, 60%-80%, 80%-100%.

Tổng số trạng thái có thể là 6×5×6×5=900 được lưu trữ trong hai bảng Q-Table. Thiết kế này đảm bảo rằng tác tử có cái nhìn toàn diện về tài nguyên (CPU, số pod) và QoS (latency), từ đó đưa ra hành động phù hợp.

- ***A*: Không gian hành động (Action Space).** Không gian hành động định nghĩa tập hợp các hành vi mà tác tử có thể lựa chọn tại mỗi bước tương tác với môi trường. Trong mô hình đề xuất, không gian này được rời rạc hóa thành ba hành động mà tác tử có thể quyết định, bao gồm:

- (0) Scale-in: Chấm dứt Pod serverless.

- (1) Do nothing: Không thực hiện hành động nào.

- (2) Scale-out**:** Triển khai Pod serverless.

- ***P*: Hàm chuyển đổi trạng thái (Transition Function).** Là một hàm xác suất mô tả cách thức môi trường chuyển từ trạng thái s sang trạng thái kế tiếp s' khi tác tử thực hiện hành động a. Hàm này được định nghĩa là $P(s' \mid s, a)$, biểu thị xác suất đạt đến trạng thái s' khi ở trạng thái s và thực hiện hành động a.

- ***γ*: Hệ số chiết khấu (Discount Factor).** Ký hiệu là γ, là một giá trị nằm trong khoảng [0,1]. Nó xác định tầm quan trọng tương đối của các phần thưởng trong tương lai so với phần thưởng tức thì:

- Nếu $\gamma = 0$ tác tử chỉ quan tâm đến phần thưởng trước mắt (short-term rewards).

- Nếu $\gamma \approx 1$ tác tử quan tâm đến tổng phần thưởng dài hạn (long-term rewards), không chiết khấu giá trị của các phần thưởng tương lai.

- ***r*: Hàm phần thưởng (Reward Function).** Hàm phần thưởng được thiết kế nhằm định hướng tác tử học tăng cường ra quyết định phù hợp với hai mục tiêu chính trong môi trường serverless: tiết kiệm tài nguyên (giảm chi phí) và duy trì chất lượng dịch vụ (giảm độ trễ). Cụ thể, phần thưởng tại mỗi bước huấn luyện được tính từ ba thành phần sau:

- Phần thưởng hiệu suất sử dụng tài nguyên (r_{perf_res}): Phần thưởng này nhằm khuyến khích sử dụng ít Pod hơn nhưng vẫn đảm bảo QoS, từ đó tiết kiệm tài nguyên hệ thống. Công thức được định nghĩa như sau [10]:

$$\left(r_{perf\_res} \right) = \frac{10 - số\ pod}{10 - 1} \times 10 \quad (2)$$

Giá trị tối đa của phần thưởng tài nguyên là 10 (khi số Pod = 1) và tối thiểu là 0 (khi số Pod = 10).

- Phần thưởng độ trễ ($r_{latency}$): Phần thưởng này phản ánh mức độ thỏa mãn yêu cầu về độ trễ (latency) của dịch vụ. Công thức được định nghĩa dựa trên hàm mũ suy giảm [10]:

$$\left(r_{latency} \right) = e^{-p \times w_{lat} \times \frac{latency}{SLA}} \times 10 \quad (3)$$

trong đó:

- p = 0.1: hệ số điều chỉnh tốc độ suy giảm
- w_{lat} là hệ số phạt độ trễ, xác định như sau: $w_{lat} = 0.3$ nếu latency < SLA (dịch vụ tốt), $w_{lat} = 10$ nếu latency ≥ SLA (dịch vụ vi phạm QoS).
- $SLA = 1$ (giây): ngưỡng độ trễ tối đa chấp nhận được

- Hình phạt dư Pod (Penalty): Một hình phạt bổ sung được áp dụng nhằm tránh hệ thống sử dụng số lượng Pod vượt quá mức cần thiết, được bật khi mà hệ thống đã đảm bảo hiệu năng tốt (latency_ratio < 90%). Khi đó, nếu số lượng Pod vượt quá giá trị lý tưởng $ideal\_pod$ = 6, hình phạt được tính như sau:

$$penalty = \begin{cases} \tanh\left(\frac{số\ pod - 6}{4}\right), & \text{nếu latency\_ratio} < 90\% \\ 0, & \text{ngược lại} \end{cases} \quad (4)$$

Hàm tanh (hyperbolic tangent) làm mượt mức phạt, giúp thay đổi đột ngột khi số Pod chỉ chênh lệch nhỏ so với mức lý tưởng. Việc bổ sung thành phần penalty được xem là một cải tiến của hàm phần thưởng truyền thống, nhằm giảm thiểu số lượng Pod rỗi không cần thiết, từ đó khuyến khích tác tử học chính sách tiết kiệm tài nguyên hơn. Cách tiếp cận này tuân theo nguyên

tắc reward shaping (định hình phần thưởng) trong học tăng cường, nhằm dẫn dắt tác tử học chính sách hiệu quả và ổn định hơn [17].

- **Tổng phần thưởng:** Trong các nghiên cứu về học tăng cường, đặc biệt trong bối cảnh bài toán đa tác tử, Grunitzki [16] đã đề xuất một phương pháp linh hoạt để thiết kế hàm phần thưởng theo hướng đa mục tiêu (multi-objective reward design), trong đó nhiều tiêu chí đánh giá như mức sử dụng tài nguyên và độ trễ phản hồi có thể được kết hợp thành một hàm mục tiêu duy nhất thông qua các trọng số phù hợp. Dựa trên hướng tiếp cận này, luận văn xây dựng hàm phần thưởng tổng tại mỗi bước theo công thức:

$$r_{all} = \alpha * r_{perf\_res} + \beta * r_{latency} - \text{penalty} \quad (5)$$

Hai tham số α và β được sử dụng để điều chỉnh mức độ quan trọng tương đối của hai yếu tố trong hàm phần thưởng: hiệu suất sử dụng tài nguyên và độ trễ dịch vụ. Tổng của hai hệ số này được cố định ở mức $\alpha + \beta = 2.5$ phù hợp với định hướng thiết kế được đề xuất bởi Grunitzki [16], nhằm giới hạn giá trị phần thưởng trong một phạm vi ổn định. Thiết lập này giúp kiểm soát biên độ dao động của phần thưởng, từ đó hỗ trợ tác tử học tăng cường hội tụ nhanh hơn và ổn định hơn trong quá trình huấn luyện. Trong bài báo này, hai trọng số được lựa chọn là $\alpha = \beta = 1.25$, với mục tiêu đảm bảo sự cân bằng giữa hai mục tiêu: tiết kiệm tài nguyên hệ thống và duy trì chất lượng dịch vụ ở mức chấp nhận được. Với phần thưởng tổng có giá trị tối đa về lý thuyết là: $1.25 \times 10 + 1.25 \times 10 = 25$. Đây là một thiết kế hợp lý trong các bài toán tối ưu đa mục tiêu, giúp tác tử điều hướng hành vi theo hướng cân bằng giữa chi phí triển khai và chất lượng phục vụ.

Trong quá trình quyết định Markov, tác tử học cách tối ưu hóa chính sách $\pi(s)$ để tối đa hóa tổng phần thưởng kỳ vọng. Tuy nhiên, trong quá trình học, tác tử phải cân bằng giữa:

- Khai thác (Exploitation): Chọn hành động tốt nhất dựa trên kiến thức hiện có.
- Khám phá (Exploration): Thử nghiệm các hành động mới để tìm chiến lược tối ưu hơn.

Vấn đề đặt ra nếu tác tử chỉ khai thác, nó có thể bị mắc kẹt trong một chính sách kém tối ưu, nếu chỉ khám phá, nó có thể tốn quá nhiều thời gian để tìm chính sách tối ưu. Để giải quyết vấn đề này, một chiến lược phổ biến được áp dụng là $\varepsilon -$ greedy. Theo đó, tại mỗi bước, tác tử sẽ chọn hành động có giá trị Q cao nhất (khai thác) với xác suất $1 - \varepsilon$ và với xác suất ε, tác tử sẽ chọn ngẫu nhiên một hành động bất kỳ (khám phá). Cụ thể, tại mỗi trạng thái s, tác tử chọn hành động a theo quy tắc:

$$a_t = \begin{cases} \text{argmax}_{a_t} Q(s_t, a_t) & \textit{với xác suất } (1-\varepsilon) \\ \textit{chọn hành động ngẫu nhiên} & \textit{với xác suất } e \end{cases} \quad (6)$$

trong đó:

- ε là tỷ lệ khám phá, thường nằm trong khoảng. Nếu ε cao, tác tử sẽ khám phá nhiều hơn.
- Nếu ε thấp, tác tử sẽ ưu tiên khai thác.

Ban đầu, ε được khởi tạo với giá trị cao để tăng cường khả năng khám phá trong giai đoạn đầu khi tác tử chưa có nhiều kinh nghiệm về môi trường. Sau đó, ε được điều chỉnh giảm dần sau mỗi episode theo công thức hàm mũ (exponential decay):

$$\varepsilon \leftarrow \max(\varepsilon_{min}, \varepsilon * \varepsilon_{decay}) \quad (7)$$

Trong đó, ε_{decay} ($0 < \varepsilon_{decay} < 1$) là hệ số điều chỉnh tốc độ suy giảm, còn ε_{min} là ngưỡng tối thiểu nhằm duy trì một mức độ khám phá nhất định. Việc giảm ε theo thời gian giúp tác tử từng bước chuyển dịch từ giai đoạn khám phá sang khai thác, tập trung vào việc tối ưu hóa chính sách hành động dựa trên những gì đã học được. Đồng thời, việc kết hợp sử dụng bộ nhớ kinh nghiệm (replay buffer) và huấn luyện trên minibatch giúp tăng tính ổn định và khả năng tổng quát của chính sách học được.

Chiến lược $\varepsilon-$ greedy vì vậy giữ vai trò quan trọng trong việc ngăn tác tử hội tụ sớm vào các chính sách hành động cục bộ chưa tối ưu, từ đó góp phần đảm bảo quá trình hội tụ ổn định của thuật toán ERDQ-learning trong điều chỉnh ngưỡng HPA một cách thích nghi và hiệu quả, phù hợp với tính biến động theo thời gian thực của môi trường Kubernetes

3.3 Thuật toán học tăng cường ERDQ-learning phân chia tỉ lệ Pod dựa trên ngưỡng

Mô hình đề xuất sử dụng thuật toán ERDQ-learning với việc tích hợp thêm hai kỹ thuật nâng cao của học tăng cường là Experience Replay và Double Q-learning vào Q-learning truyền thống (được mô tả trong Hình 4), nhằm khắc phục các hạn chế của Q-learning truyền thống như chậm hội tụ, sai lệch trong đánh giá giá trị hành động và hiệu quả khám phá kém. Cụ thể:

Experience Replay [15]: Kỹ thuật này sử dụng một bộ nhớ đệm (buffer) để lưu trữ các kinh nghiệm (state, action, reward, next_state) trong quá trình tác tử tương tác với môi trường. Tại mỗi bước huấn luyện, một minibatch gồm các mẫu được chọn ngẫu nhiên từ bộ đệm sẽ được sử dụng để cập nhật bảng Q. Điều này không chỉ giúp giảm phương sai trong quá trình học mà còn cho phép tái sử dụng dữ liệu, từ đó cải thiện tính ổn định và hiệu quả hội tụ của thuật toán.

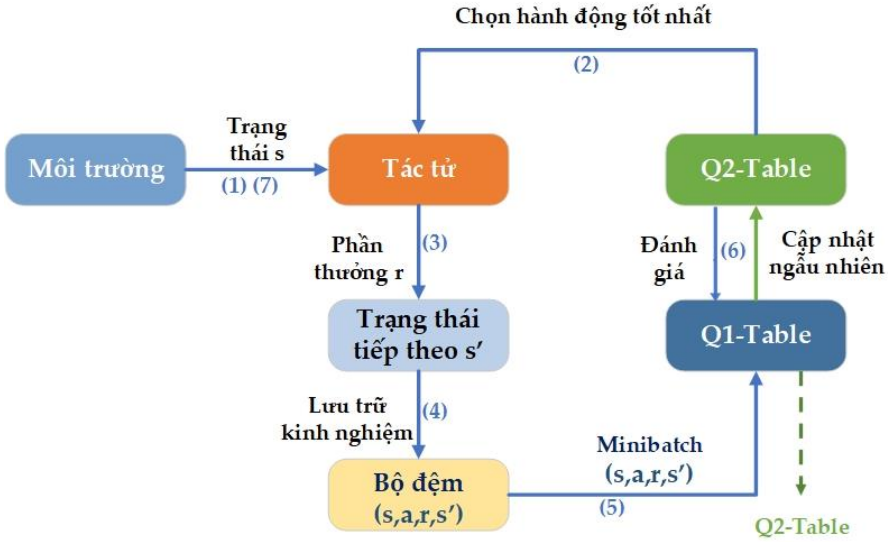
Double Q-learning [16]: Nhằm khắc phục sai lệch đánh giá quá mức giá trị hành động (overestimation bias) thường gặp trong Q-learning truyền thống, phương pháp này duy trì đồng thời hai bảng Q: Q_1 và Q_2, tại mỗi bước cập nhật, thuật toán sẽ chọn ngẫu nhiên một bảng $Q_i \in$

$\{Q_1, Q_2\}$ để cập nhật, bảng còn lại là Q_j (với $i \neq j$) được sử dụng để ước lượng giá trị hành động tốt nhất tại trạng thái tiếp theo. Công thức cập nhật được thực hiện như sau:

$$Q_i(s,a) \leftarrow Q_i(s,a) + \alpha * [r + \gamma * Q_j(s', argmax_{a'} Q_i(s',a')) - Q_i(s,a)] \quad (8)$$

Trong đó:

- s: trạng thái hiện tại
- a: hành động đã thực hiện
- α (learning rate): tốc độ học.
- r (reward): phần thưởng nhận được sau mỗi hành động.
- γ (discount factor): hệ số chiết khấu phần thưởng tương lai.
- $argmax_{a'} Q_i(s',a')$: hành động tốt nhất ở trạng thái tiếp theo theo bảng Q_i



Hình 4. Tích hợp hai kỹ thuật Experience Replay và Double Q-learning trong mô hình đề xuất

Hình 4 minh họa cơ chế hoạt động của tác tử trong mô hình đề xuất, khi kết hợp đồng thời hai kỹ thuật Experience Replay và Double Q-learning nhằm nâng cao độ ổn định và hiệu quả trong quá trình huấn luyện. Quy trình này bao gồm các bước chính sau:

(1) Tác tử nhận trạng thái hiện tại s từ môi trường.

(2) Tác tử chọn hành động a theo chính sách ε-greedy: sử dụng một trong hai bảng Q ($Q1$ hoặc $Q2$) để chọn hành động có giá trị tối ưu tại trạng thái hiện tại, trong khi bảng còn lại được để đánh giá giá trị hành động đó.

(3) Tác tử thực hiện hành động a trong môi trường, đồng thời nhận về phần thưởng r và trạng thái tiếp theo s'.

(4) Kinh nghiệm (s, a, r, s') được lưu vào bộ đệm kinh nghiệm để sử dụng trong quá trình huấn luyện.

(5) Khi bộ đệm đủ lớn, một minibatch ngẫu nhiên từ được lấy từ bộ đệm kinh nghiệm để cập nhật giá trị.

(6) Một trong hai bảng Q ($Q1$ hoặc $Q2$) được chọn ngẫu nhiên để cập nhật theo công thức (8), bảng còn lại được sử dụng để đánh giá giá trị hành động tại trạng thái kế tiếp.

(7) Cập nhật trạng thái hiện tại $s \leftarrow s'$, lặp lại chu trình từ Bước (1).

Quá trình thực thi ERDQ-learning được mô tả trong Thuật toán 2 dưới đây:

Thuật toán 2 – Thuật toán ERDQ-learning
Input: Trạng thái $S = \{s_1, s_2, \ldots, s_n\}$, Hành động $A = \{scale\_in, scale\_out, no\_change\}$, Hai bảng giá trị $Q_1(s,a)$ và $Q_2(s,a)$. Tham số: $\alpha, \gamma, \varepsilon, \varepsilon_{min}, \varepsilon_{decay}, batch\_size$. **Output:** Hai bảng $Q1$ (s,a) và $Q2$ (s,a) đã cập nhật.
Khởi tạo $Q_1(s,a) \leftarrow 0$ và $Q_2(s,a) \leftarrow 0$. Khởi tạo bộ nhớ kinh nghiệm $D \leftarrow \emptyset$. Lặp (cho từng episode): Quan sát trạng thái hiện tại s từ môi trường. Lặp (cho mỗi bước của episode): Chọn hành động a_t từ trạng thái s theo ε-greedy Thực hiện $a_t \rightarrow$ nhận r , s' Lưu (s, a_t, r, s') vào D. Nếu $\lvert D \rvert \geq batch\_size$ Lấy minibatch B gồm các mẫu ngẫu nhiên từ D Với mỗi $(s, a, r, s') \in B$: Chọn ngẫu nhiên $i \in \{1,2\}$; $j = 3 - i$ $Q_i(s,a) \leftarrow Q_i(s,a) + \alpha * [r + \gamma * Q_j(s', argmax_{a'} Q_i(s', a')) - Q_i(s,a)]$ Cập nhật: $s \leftarrow s'$, $\varepsilon \leftarrow \max(\varepsilon_{min}, \varepsilon * \varepsilon_{decay})$ Lặp mỗi episode đến khi hội tụ.

4 Phân tích kết quả

Trong phần này, chúng tôi tiến hành đánh giá hiệu quả huấn luyện của mô hình học tăng cường và so sánh với cơ chế Horizontal Pod Autoscaler (HPA) truyền thống trong Kubernetes. Mục tiêu là làm rõ khả năng của mô hình ERDQ-learning (được đề xuất trong bài báo) trong việc

cân bằng giữa chất lượng dịch vụ (QoS) và hiệu quả sử dụng tài nguyên trong môi trường serverless có tải biến thiên. Mô hình được huấn luyện trong tổng cộng 159 episode, mỗi episode gồm 25 bước (tương đương 260 giây). Thời gian huấn luyện thực tế đạt 41.340 giây (~11,48 giờ). Các tham số kỹ thuật của mô hình được trình bày trong Bảng 1.

Bảng 1. Các giá trị tham số trong mô phỏng

Tham số	Ký hiệu	Giá trị
Số lượng nút master		1
Số lượng nút worker		1
Số Pod tối thiểu được khởi tạo trước	M	1
Số Pod tối đa của hệ thống	L	10
Hiệu suất CPU mục tiêu (%)	ρ_{target}	59
Chu kỳ lấy mẫu (giây)	τ	10
Độ trễ SLA (giây)	L_{SLA}	1
Thời gian huấn luyện (episode)	T	159
Tốc độ học	α	0.3
Hệ số chiết khấu	γ	0.9
Xác suất chọn hành động ngẫu nhiên	ε	1.0
Giá trị ε tối thiểu	ε_{min}	0.05
Tốc độ giảm ε sau mỗi episode	ε_{decay}	0.995
Kích thước bộ nhớ kinh nghiệm	N_{buffer}	3000
Số lượng mẫu	N_{batch}	64

Trong suốt quá trình thực nghiệm, cơ chế HPA mặc định của Kubernetes vẫn được duy trì hoạt động song song. Tác tử học tăng cường thực hiện điều chỉnh ngưỡng mục tiêu sử dụng CPU của HPA theo thời gian thực, dựa trên trạng thái hệ thống quan sát được. Cách tiếp cận này cho phép HPA tiếp tục vận hành như một cơ chế điều phối ổn định ở lớp hạ tầng, trong khi RL đóng vai trò chiến lược – học và điều chỉnh chính sách chia tỉ lệ một cách linh hoạt nhằm thích nghi hiệu quả hơn với các biến động của tải công việc. Cụ thể, khi tác tử RL chọn hành động "tăng", "giảm" hoặc "giữ nguyên" ngưỡng HPA, các điều chỉnh này sẽ ảnh hưởng trực tiếp đến quyết định phân chia tỉ lệ số lượng Pod trong các vòng điều khiển tiếp theo của HPA. Thiết kế này bảo đảm mối quan hệ phối hợp giữa RL và HPA: tác tử RL không hoàn toàn thay thế HPA mà tối ưu hóa quá trình hoạt động của nó bằng cách điều chỉnh tham số đầu vào quan trọng nhất. Cách tiếp cận này vừa duy trì được độ tin cậy và tính ổn định vốn có của HPA, vừa khai thác được khả năng học thích nghi theo thời gian của RL trong môi trường serverless có tải động.

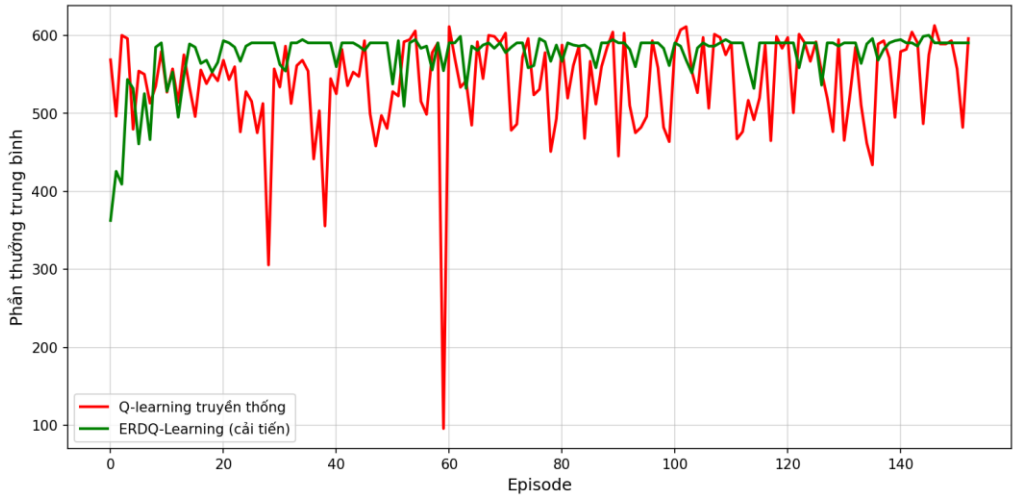
Để đánh giá hiệu quả điều phối tài nguyên giữa ba cơ chế : HPA truyền thống, HPA kết hợp với Q-learning và HPA kết hợp với ERDQ-learning, chúng tôi thực hiện các thực nghiệm mô phỏng trong cùng điều kiện tải và thời gian tương ứng với quá trình huấn luyện của ERDQ-learning. Kết quả thực nghiệm được tổng hợp và trình bày qua các biểu đồ và bảng so sánh, bao gồm:

- Biểu đồ so sánh phần thưởng trung bình theo từng chu kỳ (episode) giữa Q-learning truyền thống và ERDQ-learning,

- Biểu đồ đánh giá độ trễ trung bình và số lượng Pod trung bình giữa ba cơ chế: HPA, HPA + Q-learning và HPA + ERDQ-learning.

- Bảng so sánh số Pod trung bình được triển khai và độ trễ trung bình giữa các cơ chế.

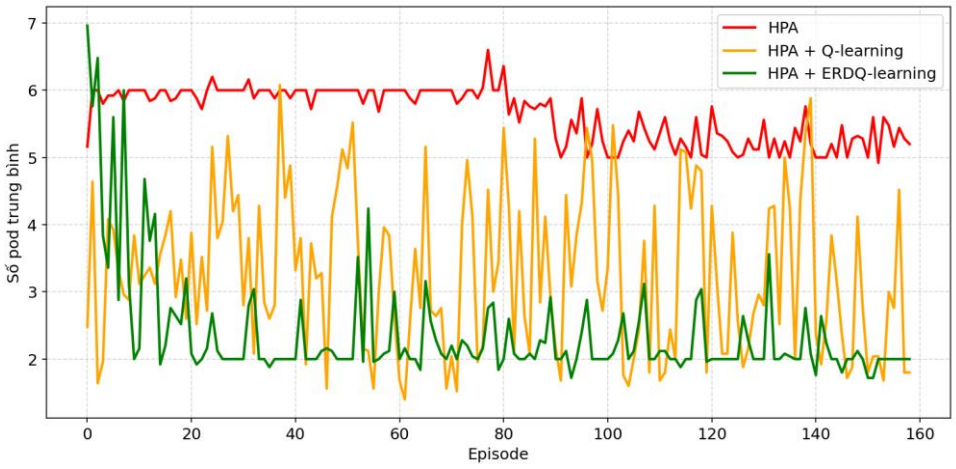
Biểu đồ trong Hình 5 thể hiện sự thay đổi phần thưởng trung bình theo từng episode trong quá trình huấn luyện của hai mô hình: ERDQ-learning (cải tiến) và Q-learning truyền thống.



Hình 5. Biểu đồ phần thưởng trung bình theo episode

Trong giai đoạn đầu, cả hai đều có phần thưởng dao động mạnh do chính sách chưa ổn định. Tuy nhiên, Q-learning cho thấy sự bất ổn rõ rệt, đặc biệt ở các episode 58–60, phản ánh việc tác tử gặp khó khăn trong việc tìm ra chiến lược điều phối hiệu quả, dẫn đến tình trạng thừa hoặc thiếu Pod. Trong khi đó, ERDQ-learning thể hiện xu hướng hội tụ nhanh hơn và ổn định hơn đáng kể. Từ episode 70 trở đi, phần thưởng của ERDQ-learning duy trì ổn định quanh mức 590–600 mà không có hiện tượng suy giảm đột ngột như Q-learning. Kết quả này cho thấy việc kết hợp Experience Replay và Double Q-learning trong ERDQ-learning giúp tăng tốc độ hội tụ và ổn định chính sách học được, góp phần nâng cao hiệu quả điều phối tài nguyên, giảm độ trễ và đảm bảo QoS trong môi trường serverless có tải biến thiên.

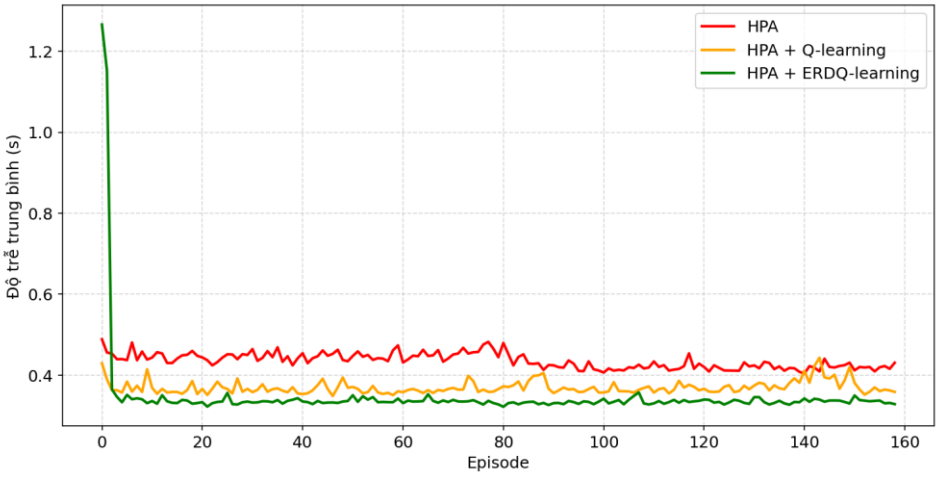
Hình 6 minh họa sự thay đổi của số lượng Pod trung bình theo từng episode đối với ba cơ chế điều phối: HPA truyền thống, HPA kết hợp với Q-learning, và HPA kết hợp với ERDQ-learning. Trong đó, HPA truyền thống (đường màu đỏ) duy trì số lượng Pod khá ổn định, dao động quanh mức 5–6 Pod trong suốt quá trình, phản ánh một chiến lược điều phối mang tính thận trọng, ít thích nghi với biến động tải và dễ dẫn đến khả năng dư thừa tài nguyên khi không cần thiết. Ngược lại, cơ chế sử dụng Q-learning (màu cam) thể hiện sự dao động mạnh về số lượng Pod, lên xuống liên tục từ 1 đến 6 Pod, đặc biệt rõ rệt trong giai đoạn giữa của huấn luyện (từ episode 40 đến 140). Điều này cho thấy tác tử Q-learning truyền thống vẫn chưa học được chính sách cân bằng hiệu quả giữa độ trễ dịch vụ và chi phí tài nguyên, dẫn đến phản ứng không ổn định với tải thay đổi. Trong khi đó, ERDQ-learning (đường màu xanh lá) thể hiện khả năng điều phối hiệu quả hơn khi duy trì số lượng Pod dao động ổn định trong khoảng 2–3 Pod. Mô hình này cho thấy tác tử đã học được chính sách phù hợp, tiết kiệm tài nguyên nhưng vẫn đảm bảo chất lượng dịch vụ. Tổng thể, ERDQ-learning vượt trội hơn cả về hiệu quả sử dụng tài nguyên và tính ổn định trong suốt quá trình huấn luyện.



Hình 6. Biểu đồ minh họa sự thay đổi của số lượng Pod trung bình qua từng episode

Hình 7 thể hiện độ trễ trung bình của dịch vụ theo từng episode đối với ba cơ chế điều phối: HPA truyền thống, HPA kết hợp Q-learning và HPA kết hợp ERDQ-learning. Theo đó, HPA truyền thống (đường màu đỏ) duy trì độ trễ trung bình ổn định trong khoảng 0.43–0.46 giây, tuy ít biến động nhưng không có dấu hiệu cải thiện rõ rệt trong suốt quá trình huấn luyện, cho thấy khả năng phản ứng còn hạn chế trước các thay đổi của tải. Cơ chế Q-learning (màu cam) giúp cải thiện độ trễ xuống khoảng 0.36–0.37 giây, cho thấy hiệu quả cao hơn so với HPA thông thường. Tuy nhiên, vẫn xuất hiện các pha dao động nhẹ về độ trễ, phản ánh chính sách điều phối chưa hoàn toàn ổn định. Trong khi đó, ERDQ-learning (đường màu xanh lá) đạt được độ trễ thấp nhất, dao động ổn định quanh mức 0.33–0.35 giây và duy trì tính ổn định đó xuyên suốt quá trình huấn luyện. Như vậy ERDQ-learning không chỉ tiết kiệm tài nguyên mà còn đảm bảo chất lượng

dịch vụ tốt hơn, nhờ khả năng điều phối linh hoạt và thích ứng với môi trường tải biến thiên. Kết quả thực nghiệm được trình bày trong Bảng 2 tiếp tục củng cố kết luận này khi thể hiện rõ ràng rằng mô hình kết hợp ERDQ-learning mang lại hiệu quả vượt trội trong việc tối ưu hóa tài nguyên và giảm độ trễ dịch vụ so với cả hai cơ chế còn lại là HPA truyền thống và HPA kết hợp Q-learning.



Hình 7. Biểu đồ minh họa độ trễ trung bình qua từng episode

Bảng 2. Kết quả so sánh số Pod, độ trễ và phần thưởng giữa các cơ chế

	Số Pod trung bình	Độ trễ trung bình (s)	Phần thưởng hội tụ
HPA	5.64	0.4352	Không áp dụng
HPA với Q-learning	3.28	0.3692	Dao động, không ổn định
HPA với ERDQ-learning	2.38	0.3460	Ổn định, hội tụ sớm

Trong Bảng 2, số lượng Pod trung bình mà HPA truyền thống triển khai là 5.64 Pod, trong khi HPA kết hợp với Q-learning truyền thống giúp giảm xuống còn 3.28 Pod (giảm 41.8%). Tuy nhiên, mô hình HPA kết hợp ERDQ-learning tiếp tục cải thiện hơn nữa khi chỉ sử dụng trung bình 2.38 Pod, tương ứng với mức tiết kiệm khoảng 57.8% so với HPA, và 27.4% so với Q-learning truyền thống.

Về độ trễ trung bình, HPA duy trì ở mức 0.4352 giây, trong khi Q-learning giúp cải thiện xuống còn 0.3692 giây. Mô hình ERDQ-learning tiếp tục nâng cao hiệu quả này với độ trễ trung bình 0.3460 giây, tương ứng với mức cải thiện khoảng 20.45% so với HPA, và 6.29% so với Q-learning.

Đáng chú ý, trong khi Q-learning truyền thống có phần thưởng dao động và không ổn định, thì ERDQ-learning cho thấy phần thưởng hội tụ sớm và duy trì ổn định qua các episode.

Điều này phản ánh khả năng học chính sách hiệu quả hơn của ERDQ-learning, giúp tác tử đưa ra quyết định điều phối tài nguyên tốt hơn theo thời gian

Sự cải thiện rõ rệt này có thể được lý giải bởi khả năng học hỏi và thích ứng linh hoạt của mô hình ERDQ-learning. Khác với HPA truyền thống vốn chỉ dựa trên một ngưỡng CPU cố định và không có khả năng điều chỉnh theo thời gian thực, cũng như Q-learning truyền thống thường gặp phải hiện tượng sai lệch đánh giá quá mức giá trị hành động (overestimation bias) do sử dụng một bảng Q duy nhất và không có cơ chế lưu trữ kinh nghiệm, ERDQ-learning đã khắc phục đồng thời cả hai điểm yếu này nhờ kết hợp hai kỹ thuật: Experience Replay giúp lưu trữ và tái sử dụng kinh nghiệm tương tác với môi trường nhằm tăng tính ổn định và đa dạng dữ liệu huấn luyện và Double Q-learning sử dụng hai bảng Q độc lập để tách biệt quá trình lựa chọn và đánh giá hành động, từ đó giảm thiểu sai lệch trong quá trình cập nhật giá trị.

Trong nghiên cứu này, tải hệ thống được tạo ra bởi công cụ Locust, bao gồm nhiều đặc điểm biến thiên theo thời gian thực như: xu hướng dài hạn, dao động ngắn hạn, các pha burst ngẫu nhiên và xu hướng tăng dần theo thời gian. Việc đánh giá mô hình trong một môi trường có tính biến động cao như vậy càng khẳng định khả năng thích nghi và hiệu quả điều phối của ERDQ-learning trong điều kiện vận hành thực tế. Do đó, ERDQ-learning được xem là một hướng tiếp cận khả thi và hiệu quả cho việc điều chỉnh ngưỡng HPA trong các hệ thống serverless hiện đại.

5 Kết luận

Kết quả thực nghiệm cho thấy mô hình học tăng cường với thuật toán ERDQ-learning vượt trội so với phương pháp HPA truyền thống, giảm 47.81% số pod được triển khai, đảm bảo độ trễ thấp và ổn định, cải thiện QoS với latency trung bình 0.3460 giây (thấp hơn 10.22% so với 0.3854 giây của HPA cố định). Khả năng thích nghi của mô hình được thể hiện qua việc điều chỉnh linh hoạt ngưỡng HPA (từ 20% đến 100%), giúp hệ thống phản ứng hiệu quả với tải công việc biến động, đồng thời duy trì số pod ở mức tối ưu. Từ các kết quả trên, có thể khẳng định rằng ERDQ-learning là một bước cải tiến rõ rệt so với Q-learning truyền thống trong bài toán điều phối tài nguyên cho môi trường serverless. Mô hình không chỉ giúp tác tử học nhanh và ổn định hơn, mà còn hình thành chính sách điều phối hiệu quả, giảm thiểu đáng kể độ trễ dịch vụ và số lượng pod dao động. Nhờ đó, ERDQ-learning đạt được sự cân bằng tốt giữa hiệu quả tài nguyên và chất lượng dịch vụ (QoS) trong điều kiện tải biến thiên. Mô hình có thể được mở rộng với việc xem xét đa mục tiêu với hàm thưởng cũng như áp dụng các thuật toán học tăng cường sâu nhằm tối ưu hơn hiệu năng của hệ thống.

Tài liệu tham khảo

1. Wen, J., Liu, Y., Chen, Z., Chen, J., & Ma, Y., "Characterizing commodity serverless computing platforms," Journal of Software: Evolution and Process, vol. 35, no. 10, p. 2394, 2023.
2. Wen, J., Chen, Z., Jin, X., & Liu, X., "Rise of the planet of serverless computing: A systematic review," CM Transactions on Software Engineering and Methodology, vol. 32, no. 5, pp. 1-61, 2023.
3. Priscilla Benedetti, Mauro Femminella, Gianluca Reali, and Kris Steenhaut, "Reinforcement learning applicability for resource-based auto-scaling in serverless edge applications," In 2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), p. 674–679, IEEE, 2022.
4. Tari, M., Ghobaei-Arani, M., Pouramini, J., & Ghorbian, M., "Auto-scaling mechanisms in serverless computing: A comprehensive review," Computer Science Review, vol. 53, pp. 53,100650, 2024.
5. J. Spillner, C. Mateos and D. A. Monge, "FaaSification of applications – From monolithic to serverless container-based architectures," in Cloud Computing and Services Science, Springer, 2019, p. 131–151.
6. K. Team, "Request flow in Knative Serving," 06 2025. [Online]. Available: https://knative.dev/docs/serving/request-flow/.
7. R. Srilakshmi, "Scaling Applications using HPA in Azure DevOps and Kubernetes," 2024. [Online]. Available: https://www.linkedin.com/pulse/scaling-applications-using-hpa-azure-devops-kubernetes-srilakshmi-r-upzfc.
8. "Kubernetes horizontal pod autoscaler," https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/, [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/. [Accessed 02 2025].
9. Zhiyu Zhang, Tao Wang, An Li, and Wenbo Zhang, "Adaptive auto-scaling of delaysensitive serverless services with reinforcement learning," In 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), p. 866–871, IEEE, 2022.
10. W. M. A. P. C. W. H. F. Z. T. K. T. B. a. R. K. I. H. Qiu, "Reinforcement learning for resource management in multi-tenant serverless platforms," in Systems, Proc. of the 2nd European Workshop on Machine Learning and, 2022, p. 20–28.
11. M. J. a. R. H. F. Rossi, "Efficient resource allocation using deep reinforcement learning," in Proc. 20th Int. Conf. on Machine Learning and Applications (ICMLA), 2021, pp. 811-816.
12. A. A. K. a. I. Ra, "Intelligent autoscaling of microservices in the cloud for real-time applications," IEEE Access, vol. 9, p. 35464–35476, 2021.
13. L. Zhong, "Reinforcement Learning Based Resource Allocation Mechanisms in Serverless Clouds," Amsterdam, 2023.
14. J. P. P. dos Santos, T. Wauters, B. Volckaert and F. De Turck, "Gym-HPA: Efficient Auto-Scaling via Reinforcement Learning for Complex Microservice-based Applications in Kubernetes," Miami, FL, USA, 2023.
15. K. K. D. S. A. G. I. A. D. W. a. M. R. V. Mnih, "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, p. 529–533, 2015.

16. H. v. Hasselt, "Double Q-learning," in Advances in Neural Information Processing Systems, vol. 23, 2010, p. 2613–2621.

17. Barto, R. S. Sutton and A. G., Reinforcement Learning: An Introduction, Cambridge, MA: MIT Press, 2018.

18. R. Grunitzki, "A Flexible Approach for Designing Optimal Reward Functions in Multi-Agent Reinforcement Learning Problems," Porto Alegre, 2018.