ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Αυτόνομη κλιμάκωση εφαρμογών υπολογιστικού νέφους χωρίς διακομιστή

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Νικόλαος Φιλίνης

Επιβλέπων : Συμεών Παπαβασιλείου
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2021

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Αυτόνομη κλιμάκωση εφαρμογών υπολογιστικού νέφους χωρίς διακομιστή

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Νικόλαος Φιλίνης

Επιβλέπων : Συμεών Παπαβασιλείου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την

.....	.....	.....
Συμεών Παπαβασιλείου	Ιωάννα Ρουσσάκη	Θεοδώρα Βαρβαρίγου
Καθηγητής Ε.Μ.Π.	Επίκουρη Καθηγήτρια Ε.Μ.Π.	Καθηγήτρια Ε.Μ.Π.

Αθήνα, Ιούνιος 2021

.....

Νικόλαος Φιλίνης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Το μοντέλο της παροχής εφαρμογών υπολογιστικού νέφους χωρίς διακομιστή (serverless computing) επιτρέπει τη γρήγορη και απρόσκοπτη ανάπτυξη κώδικα στο υπολογιστικό νέφος με τη μορφή συναρτήσεων γλωσσών προγραμματισμού. Η υποδομή αποκρύπτεται από τον χρήστη και οι πόροι δεσμεύονται δυναμικά ανάλογα με τη ζήτηση. Αυτό το μοντέλο έχει πολλαπλά οφέλη για τον πάροχο υπηρεσιών υπολογιστικού νέφους και τον πελάτη αφού υπάρχουν χρεώσεις μόνο όσο χρησιμοποιούνται οι πόροι και δεσμεύονται μόνο όσοι πόροι χρειάζονται. Η δυναμική δέσμευση και αποδέσμευση των πόρων με βάση τη ζήτηση των υπηρεσιών ονομάζεται ελαστικότητα. Η ρύθμιση και διαχείριση των πόρων πραγματοποιείται από μια serverless πλατφόρμα που τρέχει πάνω από ένα ενορχηστρωτή υπηρεσιών (π.χ., Kubernetes). Η παρούσα εργασία εστιάζει στην καλύτερη διαχείριση της ελαστικότητας ώστε οι εφαρμογές να χρησιμοποιούν με βέλτιστο τρόπο τους διαθέσιμους πόρους και να ελαχιστοποιείται το κόστος χωρίς να υπάρξει πτώση στην απόδοση των εφαρμογών. Στο πλαίσιο της εργασίας χρησιμοποιήθηκε η serverless πλατφόρμα Kubeless και δημιουργήθηκαν ευφυείς πράκτορες ενισχυτικής μάθησης οι οποίοι διαχειρίζονται την κλιμάκωση των παρεχόμενων υπηρεσιών. Για την εκπαίδευση του πράκτορα στα πλαίσια της ενισχυτικής μάθησης αναπτύχθηκαν διάφορες μοντελοποιήσεις του περιβάλλοντος και εφαρμόστηκαν διαφορετικοί αλγόριθμοι ενισχυτικής μάθησης. Στις διακριτές μοντελοποιήσεις του περιβάλλοντος χρησιμοποιήθηκαν αλγόριθμοι όπως ο Q-learning και ο DynaQ+ ενώ στα συνεχή περιβάλλοντα αξιοποιήθηκαν νευρωνικά δίκτυα και εφαρμόστηκε ο αλγόριθμος Deep Q-learning. Η αξιολόγηση της επίδοσής τους έγινε με βάση την απόδοση της εφαρμογής και τη χρησιμοποίηση των πόρων.

Λέξεις Κλειδιά

Υπολογιστικό νέφος, Serverless computing, Ενισχυτική μάθηση, Νευρωνικά δίκτυα, Ελαστικότητα, Κλιμάκωση, Kubernetes, Kubeless, Q-learning, DynaQ+, DQN, Deep Q-learning

Abstract

The serverless computing model enables fast and seamless deployment of code in the cloud in the form of programming language functions. The infrastructure is abstracted and hidden from the user and resources are dynamically allocated according to demand. This model has multiple benefits for the cloud computing provider and the customer, since the customer is charged only for the time that the resources were used and only the necessary resources are provisioned. The scaling up and down of resources is called elasticity. Resource configuration and management is performed by a serverless platform running on top of an orchestration platform (e.g., Kubernetes). This paper focuses on how to better manage elasticity so that applications use the optimal amount of resources to minimize costs without a drop in application performance. In the context of this work, the serverless computing platform Kubeless was used and intelligent reinforcement learning agents were created which manage scalability. In order to train the reinforcement learning agents, different configurations of the environment were developed and different reinforcement learning algorithms were tested. In the discrete versions of our environment algorithms such as Q-learning and DynaQ+ were used while in the continuous environments neural networks were utilized and the Deep Q-learning algorithm was applied. The evaluation of their efficiency was based on the application performance and resource utilization.

Keywords

Cloud computing, Serverless computing, Reinforcement learning, Neural networks, Elasticity, Scaling, Kubernetes, Kubeless, Q-learning, DynaQ+, DQN, Deep Q-learning

Περιεχόμενα

Περίληψη **4**

Abstract **5**

Περιεχόμενα **6**

1. Εισαγωγή **8**

1.1 Στόχος της εργασίας 8

1.2 Δομή της εργασίας 9

2. Παροχή υπολογιστικών υπηρεσιών χωρίς διακομιστή (Serverless Computing) **10**

2.1 Ανάλυση μοντέλων παροχής υπηρεσιών υπολογιστικού νέφους 10

2.2 Ορισμός του serverless computing 11

2.3 Προκλήσεις για παροχή υπηρεσιών με serverless τρόπο 13

2.3.1 Ψυχρή εκκίνηση (Cold start) 13

2.3.2 Αυτόματη κλιμάκωση (Elasticity management) 14

2.4 Βασικά εργαλεία ενορχήστρωσης για serverless computing 15

2.4.1 Kubernetes 15

2.4.2 Πλατφόρμες serverless computing 17

2.4.3 Πίνακας συγκρίσεων serverless πλατφορμών 18

2.5 Μηχανισμοί διαχείρισης κλιμακωσιμότητας serverless εφαρμογών 19

3. Μηχανισμοί ενορχήστρωσης με αξιοποίηση τεχνικών ενισχυτικής μάθησης **21**

3.1 Εισαγωγή στην ενισχυτική μάθηση 21

3.1.1 Ορισμός 21

3.2.2 Συναρτήσεις αξίας 22

3.2 Αλγόριθμοι ενισχυτικής μάθησης 23

3.2.1 Βασικά χαρακτηριστικά αλγορίθμων 23

3.2.2 Δυναμικός προγραμματισμός 24

3.2.3 Μέθοδοι Monte Carlo 24

3.2.4 Μέθοδοι Temporal difference 24

3.2.4.1 Q-learning 25

3.2.4.2 DynaQ και DynaQ+ 26

3.2.4.3 Deep Q-learning 27

3.3 Αξιοποίηση τεχνικών ενισχυτικής μάθησης για διαχείριση της κλιμακωσιμότητας serverless εφαρμογών 28

3.4 Βασικά εργαλεία μοντελοποίησης περιβαλλόντων και πρακτόρων ενισχυτικής μάθησης 29

3.4.1 OpenAI Gym 29

3.4.2 Tensorflow και Tensorflow Agents 30

3.5 Βασικά εργαλεία ανάπτυξης υποδομής 30
3.5.1 Kubeless 30
3.5.2 Metrics server 30
3.5.3 NGINX Ingress Controller 30
3.5.4 Prometheus 30
3.5.5 Vegeta 30
3.6 Υποδομή δοκιμών και αρχιτεκτονική 31

4. Ανάπτυξη περιβαλλόντων ενισχυτικής μάθησης 32
4.1 Σύντομη επισκόπηση των περιβαλλόντων ενισχυτικής μάθησης 32
4.2 Περιβάλλοντα διακριτού χώρου καταστάσεων και διακριτού χώρου κινήσεων 34
4.2.1 Πραγματικό περιβάλλον 34
4.2.2 Προσομοίωση περιβάλλοντος 39
4.3 Περιβάλλοντα συνεχή χώρου καταστάσεων και διακριτού χώρου κινήσεων 43
4.3.1 Πραγματικό περιβάλλον 43
4.3.2 Προσομοίωση περιβάλλοντος 47
4.4 Σύνοψη αποτελεσμάτων 49

5. Συμπεράσματα και μελλοντικές επεκτάσεις 50

Βιβλιογραφία 51

Παραρτήματα 56
Γλωσσάριο 56
Ανοικτό Αποθετήριο Κώδικα 57

1. Εισαγωγή

1.1 Στόχος της εργασίας

Τα τελευταία χρόνια η χρήση τεχνολογιών υπολογιστικού νέφους έχει αυξηθεί σε σημαντικό βαθμό. Η παροχή υπηρεσιών που απαιτούν υπολογιστική ισχύ μετατοπίζεται από τη λογική αγοράς ενός προϊόντος και προσεγγίζει την έννοια της υπηρεσίας που παραδίδεται στους πελάτες μέσω του διαδικτύου από υπολογιστικά κέντρα ή αλλιώς νέφη [1].

Το υπολογιστικό νέφος (cloud computing) έχει τη δυνατότητα να προσφέρει υπηρεσίες σε πελάτες με διάφορα επίπεδα αφαίρεσης. Συγκεκριμένα, οι πάροχοι μπορούν να προσφέρουν από εικονικές μηχανές μέχρι ολοκληρωμένες υπηρεσίες για τους τελικούς χρήστες. Αυτό το μοντέλο παροχής υπηρεσιών ονομάζεται Everything as a Service (EaaS ή aaS ή XaaS) [2]. Με αυτόν τον τρόπο, ανάλογα με το ποιο μοντέλο εφαρμόζεται, ο πελάτης μπορεί να έχει στη διάθεση του μια εικονική μηχανή για να αναπτύξει τον κώδικα που επιθυμεί ή ενδεχομένως μια υπηρεσία που εκτελεί μια συγκεκριμένη εργασία και φιλοξενείται σε υποδομή υπολογιστικού νέφους.

Μια τάση που παρατηρείται είναι η μετάβαση εφαρμογών από ένα μονολιθικό μοντέλο σε ένα μοντέλο που βασίζεται σε μικρά τμήματα εφαρμογών (microservices) όπου κάθε τμήμα μιας εφαρμογής αποτελεί ξεχωριστή οντότητα. Αυτό επιτρέπει την ταχύτερη ανάπτυξη κώδικα αφού πλέον κάθε τμήμα είναι ανεξάρτητο και τα τμήματα επικοινωνούν μεταξύ τους συνήθως μέσω δικτυακών διεπαφών. Σε αυτό βοηθά και η υιοθέτηση της τεχνολογίας των container που αποτελούν πιο ελαφριά εναλλακτική όσον αφορά τις ανάγκες τους σε υπολογιστικούς πόρους σε σχέση με τις εικονικές μηχανές.

Αποτέλεσμα των παραπάνω τεχνολογιών είναι η στροφή προς εφαρμογές που φιλοξενούνται στο νέφος και πλέον μιλάμε για Cloud-native computing. Ένα μέρος αυτής της προσέγγισης για ανάπτυξη containerized εφαρμογών σε συνδυασμό με τα microservices αποτελεί και η ανάπτυξη υπηρεσιών υπολογιστικού νέφους χωρίς διακομιστή (serverless computing). Στόχος του serverless computing είναι η απόκρυψη της υποδομής πίσω από τους παρεχόμενους πόρους και η ευκολία στην ανάπτυξη κώδικα για εφαρμογές υπολογιστικού νέφους.

Ένα από τα βασικά πλεονεκτήματα του serverless computing μοντέλου είναι η ευκολία στην κλιμάκωση των πόρων ώστε να ανταποκρίνεται καλύτερα στη ζήτηση και στις απαιτήσεις των χρηστών. Συγκεκριμένα, λόγω της ευκολίας δημιουργίας και αποδέσμευσης των containers, αυτό το μοντέλο εφαρμογής μπορεί να οδηγήσει σε χαμηλότερα κόστη. Ωστόσο, υπάρχουν διάφορες προκλήσεις που προκύπτουν σχετικά με τη βέλτιστη διαχείριση των πόρων.

Αντικείμενο της εργασίας είναι η ανάπτυξη μηχανισμών που υποστηρίζουν την αυτόματη κλιμάκωση εφαρμογών σε υπολογιστικά νέφη χωρίς διακομιστή (serverless). Για την επίτευξη αυτού του στόχου αξιοποιήθηκαν τεχνικές ενισχυτικής

μάθησης (reinforcement learning) που αποτελεί κλάδο της μηχανικής μάθησης. Κωδικοποιώντας ένα περιβάλλον με serverless αρχιτεκτονική, δημιουργήθηκαν ευφυείς πράκτορες οι οποίοι εκπαιδεύονται ώστε να παίρνουν καλές αποφάσεις κλιμάκωσης. Βασικό κριτήριο για την αξιολόγηση των αποφάσεων του πράκτορα αποτελούν η ελαχιστοποίηση των χρησιμοποιούμενων πόρων με άμεσο αποτέλεσμα τη μείωση του κόστους χωρίς όμως να έχουμε αρνητικό αντίκτυπο στην ποιότητα της παρεχόμενης υπηρεσίας (Quality of Service - QoS).

1.2 Δομή της εργασίας

Η εργασία είναι χωρισμένη σε 5 κεφάλαια. Το παρόν κεφάλαιο αποτελεί μια εισαγωγή στο θέμα της εργασίας με μια σύντομη περιγραφή του στόχου και της συνεισφοράς της. Το υπόλοιπο μέρος της εργασίας είναι οργανωμένο στα εξής κεφάλαια:

- Στο κεφάλαιο 2 αναλύεται το μοντέλο υπολογιστικού νέφους χωρίς διακομιστή (serverless computing) όπου εξηγούνται η λειτουργία, οι προκλήσεις και παρουσιάζονται μερικές υλοποιήσεις του. Συνοψίζεται επίσης η υπάρχουσα βιβλιογραφία σχετικά με τη διαχείριση της κλιμακωσιμότητας σε serverless περιβάλλοντα.
- Στο κεφάλαιο 3 γίνεται μια εισαγωγή στην ενισχυτική μάθηση και παρουσιάζονται κάποιοι βασικοί αλγόριθμοι που εφαρμόστηκαν στο πλαίσιο της εργασίας. Επεξηγείται το πώς μπορεί να εφαρμοστεί η ενισχυτική μάθηση για να λύσουμε το ζητούμενο πρόβλημα και ποια προσέγγιση ακολουθήθηκε για τη μοντελοποίηση του περιβάλλοντος.
- Στο κεφάλαιο 4 γίνεται μια επισκόπηση των διαφορετικών περιβαλλόντων που δοκιμάστηκαν με τους εκάστοτε αλγόριθμους που εφαρμόστηκαν και παρουσιάζονται αποτελέσματα σχετικά με την απόδοση τους.
- Στο κεφάλαιο 5 αξιολογούνται τα αποτελέσματα και τα ευρήματα της εργασίας, παρουσιάζονται τα βασικά συμπεράσματα και σχολιάζονται πιθανές προεκτάσεις.

2. Παροχή υπολογιστικών υπηρεσιών χωρίς διακομιστή (Serverless Computing)

Σε αυτό το κεφάλαιο γίνεται μια σύντομη επισκόπηση στα διαφορετικά μοντέλα παροχής υπηρεσιών με χρήση τεχνολογιών υπολογιστικού νέφους. Στη συνέχεια δίνεται ο ορισμός του serverless computing και παρουσιάζονται τα βασικά εργαλεία ενορχήστρωσης σε serverless περιβάλλοντα.

2.1 Ανάλυση μοντέλων παροχής υπηρεσιών υπολογιστικού νέφους

Το υπολογιστικό νέφος αναφέρεται στην παροχή υπολογιστικής ισχύος σε πελάτες ως υπηρεσία. Ο πάροχος της υπηρεσίας δεσμεύει τους κατάλληλους πόρους που ζητήθηκαν και τους παρέχει στους πελάτες μέσω του διαδικτύου. Υπάρχουν διάφορες υλοποιήσεις για την παροχή υπηρεσιών υπολογιστικού νέφους ανάλογα με το μοντέλο και το είδος της παρεχόμενης υπηρεσίας. Βασικά μοντέλα παροχής υπηρεσιών υπολογιστικού νέφους αποτελούν τα: IaaS, PaaS και SaaS [3].

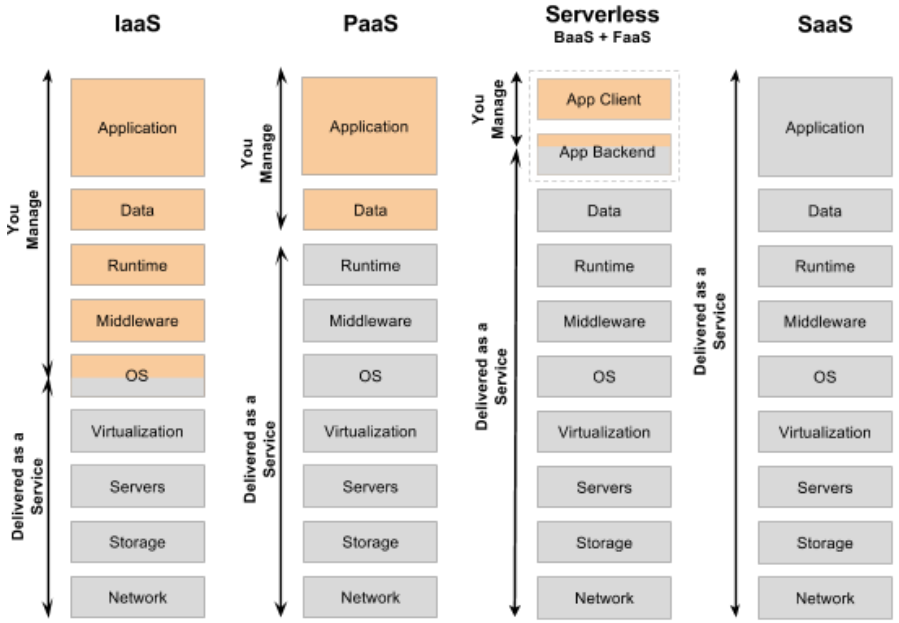
- **IaaS (Infrastructure as a Service/Υποδομή ως υπηρεσία)**: Σε αυτό το μοντέλο παροχής υπηρεσιών, οι παρεχόμενοι πόροι είναι συνήθως εικονικές μηχανές (Virtual Machines), διακομιστές (servers), εξισορροπητές φόρτου (load balancers), δίκτυα ή containers. Τα βασικότερα προϊόντα αποτελούν οι εικονικές μηχανές και τα containers τα οποία είναι μια πιο ελαφριά εναλλακτική των εικονικών μηχανών. Στον πελάτη επομένως προσφέρεται η απαραίτητη υποδομή για την ανάπτυξη κώδικα ανεξαρτήτως λειτουργικού συστήματος και γλώσσας προγραμματισμού. Ταυτόχρονα η δέσμευση των πόρων αποκρύπτεται από τον πελάτη. Το μοντέλο IaaS παρέχει μεγάλη ευελιξία αφού ο πελάτης επιλέγει ελεύθερα πως θα χρησιμοποιήσει την υποδομή αλλά επιβαρύνεται με το στήσιμο και την προετοιμασία του περιβάλλοντος με το οποίο επιθυμεί να εργαστεί.
- **PaaS (Platform as a Service/Πλατφόρμα ως Υπηρεσία)**: Σε αυτό το μοντέλο παροχής υπηρεσιών, προσφέρονται στον πελάτη έτοιμα περιβάλλοντα εκτέλεσης προγραμματιστικών γλωσσών, βιβλιοθήκες κώδικα και άλλα εργαλεία. Βρίσκεται σε ένα υψηλότερο επίπεδο αφαίρεσης σε σχέση με το IaaS αφού πλέον ο πελάτης δεν ασχολείται με τη δημιουργία και οργάνωση δικτύων, λειτουργικών συστημάτων και περιβαλλόντων εκτέλεσης. Ο πελάτης λοιπόν περιορίζεται από τα προσφερόμενα περιβάλλοντα εκτέλεσης του παρόχου αλλά επιταχύνεται και διευκολύνεται η ανάπτυξη κώδικα.
- **SaaS (Software as a Service/Λογισμικό ως Υπηρεσία)**: Σε αυτό το μοντέλο παροχής υπηρεσιών, προσφέρεται στον πελάτη μια εφαρμογή του παρόχου που

τρέχει στο υπολογιστικό νέφος. Αποτελεί το μεγαλύτερο επίπεδο αφαίρεσης επειδή ο πελάτης δεν έχει πρόσβαση στην υποδομή που φιλοξενεί την εφαρμογή. Συνήθως πρόκειται για λογισμικό που χρεώνεται με βάση τη χρήση του ή με κάποια συνδρομή. Παραδείγματα του μοντέλου SaaS είναι υπηρεσίες αποθηκευτικού χώρου στο νέφος και σουίτες εφαρμογών γραφείου που είναι διαθέσιμες στο διαδίκτυο. Στο μοντέλο SaaS ο πελάτης δεν μπορεί να αναπτύξει δικό του κώδικα και η υλοποίηση της εφαρμογής εξαρτάται μόνο από τον πάροχο.

Άλλες παραλλαγές των παραπάνω μοντέλων αποτελούν το BaaS (Backend as a Service/Backend ως Υπηρεσία), το CaaS (Containers as a Service/Containers ως Υπηρεσία), το FaaS (Function as a Service/Συνάρτηση ως Υπηρεσία) και το Serverless computing.

2.2 Ορισμός του serverless computing

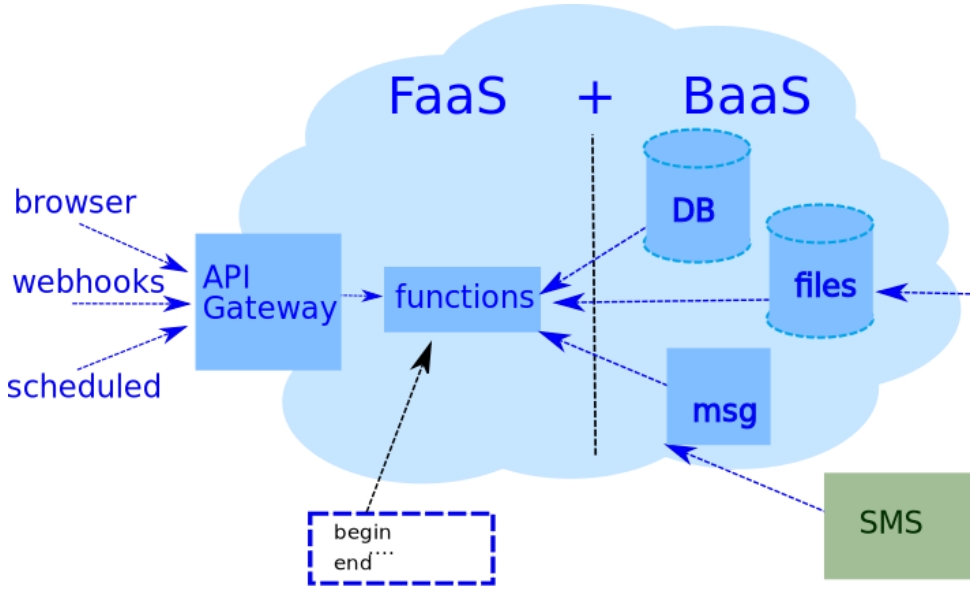
Το serverless computing ορίζεται ως μια πλατφόρμα η οποία αποκρύπτει τη χρήση του server από τον προγραμματιστή (Εικόνα 2.1). Η εκτέλεση του κώδικα γίνεται ανάλογα με την υπάρχουσα ζήτηση όταν πυροδοτούνται ορισμένα γεγονότα (triggers) και ο χρήστης της υπηρεσίας χρεώνεται μόνο για τους πόρους που καταναλώθηκαν. Οι πόροι κλιμακώνονται προς τα πάνω και προς τα κάτω αυτόματα για να ανταποκριθούν στα εισερχόμενα αιτήματα [4].



Εικόνα 2.1: Συγκριτικός πίνακας μοντέλων υπολογιστικού νέφους [5]

Μια serverless πλατφόρμα μπορεί να προσφέρει ένα ή και τα δύο από τα μοντέλα Function as a Service (Faas) και Backend as a Service (BaaS) (Εικόνα 2.2) [6]. Στο μοντέλο Function as a Service (FaaS) ο πάροχος του υπολογιστικού νέφους

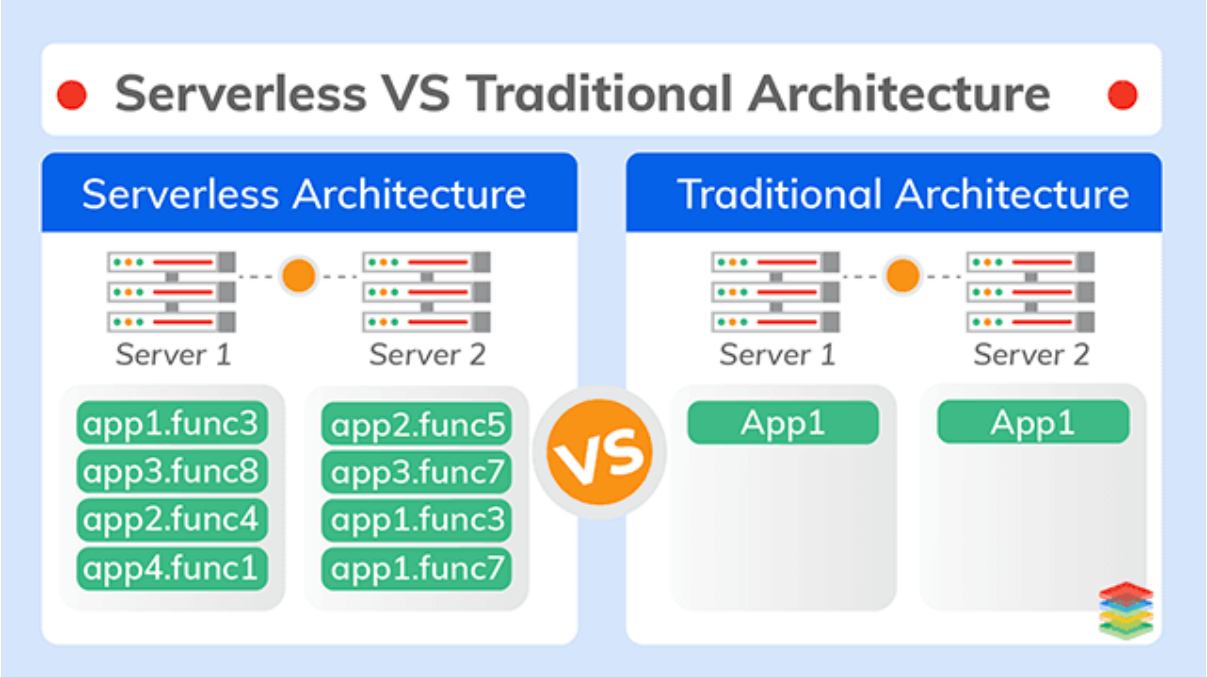
προσφέρει την απαραίτητη υποδομή για την ανάπτυξη κώδικα με τη μορφή συναρτήσεων που τρέχουν πάνω σε έτοιμα περιβάλλοντα εκτέλεσης προγραμματιστικών γλωσσών (π.χ. Python, Go, Java) [7]. Μερικές υλοποιήσεις αυτού του μοντέλου αποτελούν τα AWS Lambda, Google Functions, Azure Functions και IBM Cloud Functions. Στο μοντέλο Backend as a Service (Baas) προσφέρονται από τον πάροχο υπηρεσίες που βασίζονται σε API και είναι ικανές να υποκαταστήσουν βασικά κομμάτια μιας εφαρμογές που σχετίζονται με το back-end μιας εφαρμογής και συγκεκριμένα τις βάσεις δεδομένων, τον χώρο αποθήκευσης στο cloud κ.τ.λ.



Εικόνα 2.2: Αρχιτεκτονική serverless μοντέλου [8]

Η υλοποίηση του serverless computing βασίζεται στην τεχνολογία των containers τα οποία αποτελούν μια εναλλακτική στην τεχνολογία των εικονικών μηχανών (Virtual Machines). Οι εφαρμογές περικλείονται μαζί με τις εξαρτήσεις τους σε container images τα οποία στη συνέχεια είναι έτοιμα για εκτέλεση. Τα containers αποτελούν βελτίωση σε σύγκριση με τις εικονικές μηχανές αφού έχουν μικρότερο μέγεθος image, μικρότερο χρόνο εκκίνησης και επιτρέπουν τη δημιουργία πολλαπλών containers στο ίδιο λειτουργικό σύστημα.

Στην Εικόνα 2.3 αναδεικνύονται οι διαφορές μεταξύ μιας κλασικής αρχιτεκτονικής και του serverless μοντέλου. Αντί για μονολιθικές εφαρμογές έχουμε τμηματικές συναρτήσεις που εξυπηρετούν διαφορετικούς ρόλους και αλληλεπιδρούν μεταξύ τους για τη δημιουργία μιας υπηρεσίας.



Εικόνα 2.3: Σύγκριση Serverless και παραδοσιακής αρχιτεκτονικής [9]

Ένα από τα βασικά πλεονεκτήματα του serverless computing είναι η χρέωση με βάση τους πόρους που χρησιμοποιήθηκαν. Συγκεκριμένα ο αδρανής χρόνος της εφαρμογής δεν προσμετράται στην τελική χρέωση [10]. Κάποιες υλοποιήσεις του μοντέλου επιτρέπουν μάλιστα την πλήρη αποδέσμευση των πόρων (scale to zero) όταν περιέλθει ένα προκαθορισμένο χρονικό διάστημα αδράνειας. Συνεπώς, ένα ισχυρό προτέρημα είναι και η κλιμάκωση των εφαρμογών σε όσα αντίγραφα χρειάζονται με αυτόματο τρόπο.

Το serverless computing μπορεί να χρησιμοποιηθεί σε διάφορους τομείς αλλά η δομή του ευνοεί ιδιαίτερα τις εφαρμογές που έχουν βοηθητικό χαρακτήρα και εκτελούν μεμονωμένες εργασίες όπως κωδικοποίηση βίντεο και δημιουργία αντιγράφων ασφαλείας [11]. Επίσης, η ασύγχρονη και χωρίς κατάσταση (stateless) φύση του, το καθιστά χρήσιμο σε περιπτώσεις εφαρμογών που έχουν σπάνιες και υψηλές αυξήσεις στη ζήτηση, όπως για παράδειγμα σε προγραμματισμένες εργασίες και στην ομαδοποιημένη επεξεργασία εικόνων [7]. Επομένως, χρησιμοποιείται κυρίως για σύντομες εργασίες με μικρό όγκο δεδομένων και φόρτους που παρουσιάζουν αυξομειώσεις [11].

2.3 Προκλήσεις για παροχή υπηρεσιών με serverless τρόπο

2.3.1 Ψυχρή εκκίνηση (Cold start)

Μια από τις βασικές προκλήσεις που εντοπίζεται σε μερικές υλοποιήσεις του serverless μοντέλου είναι το πρόβλημα της ψυχρής εκκίνησης (cold start). Πιο συγκεκριμένα σε κάποια framework επιτρέπεται η αποδέσμευση όλων των πόρων

(scale to zero) μετά από ένα χρονικό διάστημα αδράνειας για την αποφυγή κόστους όταν δεν χρησιμοποιείται η υποδομή. Την επόμενη φορά που λαμβάνεται αίτημα πρέπει να δεσμευτούν πόροι εκ νέου και να φορτωθεί το image της εφαρμογής. Ο χρόνος που παρέρχεται από τη δημιουργία του αιτήματος μέχρι την εξυπηρέτηση του ονομάζεται ψυχρή εκκίνηση. Αντιθέτως, αν δεν έχουν αποδεσμευτεί όλοι οι πόροι τότε πρόκειται για θερμή εκκίνηση. Η ψυχρή εκκίνηση αποτελεί ιδιαίτερο πρόβλημα σε εφαρμογές που εκτελούνται σπάνια και χαρακτηρίζονται από μεγάλα διαστήματα αδράνειας [12].

Για την αντιμετώπιση του cold start έχουν προταθεί διάφορες τεχνικές. Κάποιες από αυτές είναι η βελτιστοποίηση των περιβαλλόντων εκτέλεσης μέσω της μείωσης του χρόνου προετοιμασίας και φόρτωσης των container images και η ελάττωση της συχνότητας των ψυχρών εκκινήσεων με περιοδικά αιτήματα (ping) ώστε να μην αποδεσμεύονται οι πόροι [12].

Πιο πολύπλοκες αρχιτεκτονικές περιλαμβάνουν πολλαπλές ουρές που χωρίζονται σε ψυχρή, θερμή και ουρά με πρότυπα όπου τα containers ανατίθενται σε μία από τις ουρές ανάλογα με το πόσο συχνά λαμβάνουν αιτήματα και υπάρχουν διαφορετικά τμήματα που αναλαμβάνουν τη δρομολόγηση των αιτημάτων στην κατάλληλη ουρά [13].

Μια περαιτέρω υλοποίηση προτείνει τη χρήση μικρών containers τα οποία χρησιμοποιούν συστήματα αρχείων χωρισμένα σε επίπεδα και δέσμευση φακέλων για την αποφυγή της αντιγραφής αρχείων. Επίσης, χρησιμοποιούνται pools από cgroups τα οποία χρησιμοποιούνται για τη δημιουργία των containers και Unix domain sockets για την επικοινωνία του συστήματος με τις διεργασίες εντός των containers. Νέες διεργασίες γίνονται fork από ήδη υπάρχουσες στις οποίες έχουν προφορτωθεί βιβλιοθήκες και το σύστημα εκμεταλλεύεται μηχανισμούς caching για την επιπλέον επιτάχυνση της δημιουργίας νέων αντιγράφων [14].

Τέλος, αναφέρουμε τη δυνατότητα ομαδοποίησης όλων των υπηρεσιών ή συναρτήσεων ανάλογα με τη γλώσσα προγραμματισμού ώστε να τρέχουν στην ίδια διεργασία. Αυτό το σύστημα απαιτεί την ύπαρξη ενός προγράμματος που θα ευθύνεται για το διαμοιρασμό του χρόνο του επεξεργαστή ανάμεσα στις διαφορετικές συναρτήσεις. Με αυτόν τον τρόπο δεν αποδεσμεύονται οι πόροι και αποφεύγεται το πρόβλημα της ψυχρής εκκίνησης [15].

2.3.2 Αυτόματη κλιμάκωση (Elasticity management)

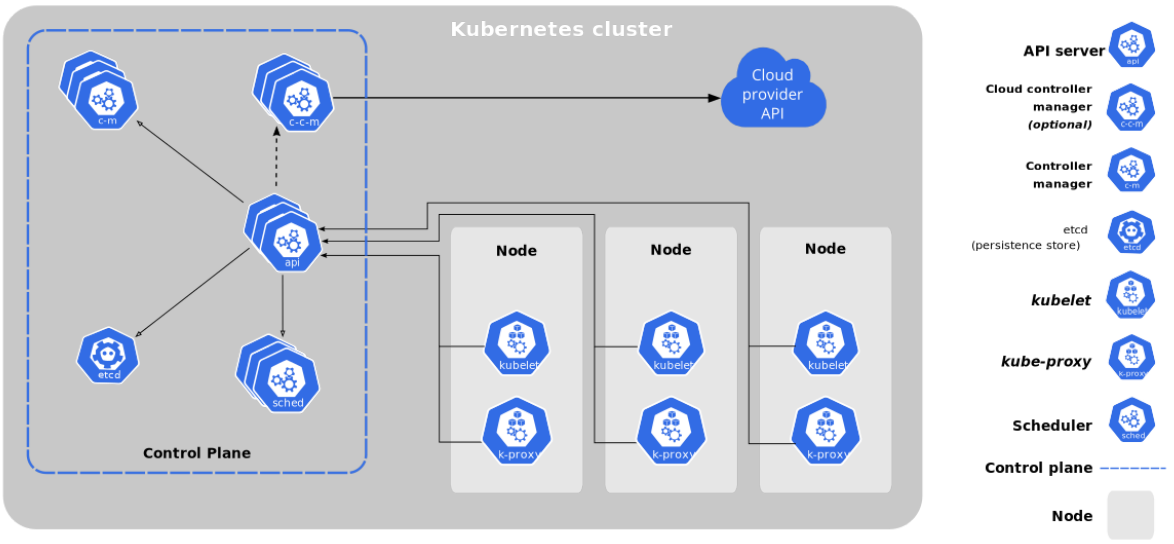
Η βασική πρόκληση στην οποία επικεντρώνεται η παρούσα εργασία είναι η αυτόματη κλιμάκωση. Μια χαρακτηριστική ιδιότητα του serverless μοντέλου είναι η κλιμάκωση των πόρων, ώστε να ανταποκρίνεται στη ζήτηση και στον εισερχόμενο φόρτο. Οι διακυμάνσεις όμως στο φόρτο έχουν ως αποτέλεσμα οι δεσμευμένοι πόροι να διαφέρουν από την πραγματική ζήτηση. Συνεπώς ενδέχεται να έχουμε δέσμευση πόρων που είναι είτε πάνω είτε κάτω από τις απαιτήσεις. Όταν δεσμεύονται παραπάνω πόροι, δημιουργούνται επιπλέον χρεώσεις. Αντιθέτως, όταν οι πόροι δεν επαρκούν, η

απόδοση της εφαρμογής κινδυνεύει να πέσει. Σε περιβάλλοντα υπολογιστικών νεφών συχνά υπάρχουν συμφωνίες μεταξύ πελατών και παρόχων που ορίζουν τις απαιτήσεις για την απόδοση της εφαρμογής και ονομάζονται Service Level Agreements (SLA). Επομένως, η πτώση στην απόδοση μπορεί να οδηγήσει σε παραβίαση αυτής της συμφωνίας.

2.4 Βασικά εργαλεία ενορχήστρωσης για serverless computing

2.4.1 Kubernetes

Ένα από τα πιο γνωστά εργαλεία ενορχήστρωσης εφαρμογών υπολογιστικού νέφους παγκοσμίως είναι το Kubernetes [16]. Το Kubernetes είναι ένα σύστημα που επιτρέπει την ανάπτυξη και τη διαχείριση εφαρμογών και λογισμικού με τη χρήση containers. Χρησιμοποιεί την υπάρχουσα υποδομή ομαδοποιώντας την σε κόμβους (Nodes) που μπορεί να είναι είτε φυσικά είτε εικονικά μηχανήματα. Αυτή η ομάδα κόμβων ονομάζεται cluster. Στη συνέχεια το επίπεδο ελέγχου οργανώνει την εκτέλεση των απαραίτητων δομικών στοιχείων για την εκτέλεση μιας εφαρμογής.



Εικόνα 2.1: Δομικά στοιχεία του Kubernetes [17]

Η εκτέλεση μιας εφαρμογής στο Kubernetes προϋποθέτει την ύπαρξη ενός container image το οποίο χρησιμοποιείται κατά τη δημιουργία και εκτέλεση του αντίστοιχου container. Το Kubernetes στη συνέχεια περικλείει τα containers μέσα σε pods τα οποία με τη σειρά τους δρομολογούνται στους διαθέσιμους κόμβους.

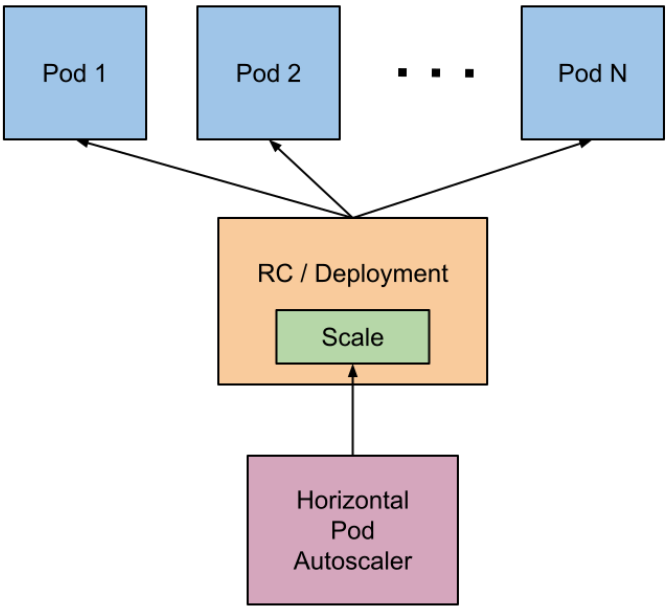
Ένα βασικό χαρακτηριστικό του Kubernetes είναι η δυνατότητα αυτόματης οριζόντιας κλιμάκωσης μέσω ενός συστατικού που ονομάζεται Horizontal Pod Autoscaler (HPA). Ο χρήστης ορίζει τα επιθυμητά επίπεδα χρησιμοποίησης των

διαθέσιμων πόρων και συγκεκριμένα της υπολογιστικής ισχύος (CPU) και της μνήμης (Memory), ενώ ο HPA συλλεγεί περιοδικά μετρικές χρησιμοποίησης και φροντίζει οι μετρικές να παραμείνουν εντός των προκαθορισμένων ορίων. Για παράδειγμα, μπορούμε να θέσουμε κάποια όρια για την υπολογιστική ισχύ και την μνήμη. Αφότου δώσουμε κάποιους αρχικούς πόρους στην εφαρμογή, μπορούμε να ορίσουμε αυτά τα όρια ως ποσοστά επί αυτών των πόρων (π.χ., δίνουμε σε μια εφαρμογή 1 CPU και 128Mi μνήμης και επιθυμούμε η χρησιμοποίηση των δύο πόρων να παραμείνει 50% και 60% αντίστοιχα ως μέσος όρος όλων των pods).

Η βασική συνάρτηση του HPA που χρησιμοποιείται για τον υπολογισμό των επιθυμητών αντιγράφων είναι:

```
desiredReplicas = ceil[currentReplicas * ( currentMetricValue / desiredMetricValue )].
```

Όταν οι εφαρμογές υπερβούν τα όρια χρησιμοποίησης, ο HPA κλιμακώνει την εφαρμογή δημιουργώντας περισσότερα αντίγραφα (pods). Αντίθετα, όταν παρέλθει ένα χρονικό διάστημα στο οποίο η εφαρμογή παρουσιάζει χαμηλή χρησιμοποίηση, ο HPA τερματίζει τα αντίγραφα (pods) που δεν χρειάζονται πλέον. Πέρα από τη μνήμη και την υπολογιστική ισχύ υποστηρίζονται και προσαρμοσμένες μετρικές που ορίζονται από το χρήστη.



Εικόνα 2.2: Τρόπος λειτουργίας του HPA [18]

Ένα επιπλέον χαρακτηριστικό του Kubernetes είναι η επεκτασιμότητα η οποία οδήγησε και στη δημιουργία διαφόρων πλατφορμών που επεκτείνουν τις λειτουργίες του Kubernetes. Συνήθως τρέχουν πάνω από το Kubernetes ως ξεχωριστά δομικά στοιχεία και προσθέτουν τη δική τους λειτουργικότητα. Σε αυτές συγκαταλέγονται και οι πλατφόρμες που σχετίζονται με το serverless computing.

2.4.2 Πλατφόρμες serverless computing

Knative

Το Knative είναι μια πλατφόρμα για serverless computing που προσθέτει δικά του στοιχεία πάνω στα ήδη υπάρχοντα του Kubernetes για να εκτελέσει τη δρομολόγηση των αιτημάτων και τις υπηρεσίες κλιμάκωσης. Αποτελείται από δύο βασικά τμήματα: το Knative Serving και το Knative Eventing [19].

- **Serving**: Αποτελεί το κομμάτι του Knative που είναι υπεύθυνο για τη διαχείριση και την κλιμάκωση των διαθέσιμων πόρων. Χρησιμοποιεί container images για την εκτέλεση του κώδικα και προσφέρει εναλλακτική υλοποίηση για την κλιμάκωση που ονομάζεται Knative Pod Autoscaler (KPA). Ο KPA επιτρέπει την πλήρη αποδέσμευση των πόρων όταν παρέλθει ένα χρονικό διάστημα αδράνειας (scale to zero). Οι βασικές μετρικές που υποστηρίζει ειναι ο ταυτοχρονισμός (concurrency) που αναφέρεται στον αριθμό των ταυτόχρονων αιτημάτων που μπορεί να δεχθεί κάθε αντίγραφο της εφαρμογής και ο αριθμός των αιτημάτων ανά δευτερόλεπτο συνολικά. Εναλλακτικά, υποστηρίζει και τον HPA του Kubernetes με τις μετρικές για ταυτοχρονισμό, αριθμό αιτημάτων ανά δευτερόλεπτο και την υπολογιστική ισχύ (CPU) χωρίς τη δυνατότητα να αποδεσμεύσει όλους τους πόρους (scale to zero). Ένα επιπλέον χαρακτηριστικό του Knative είναι ότι λειτουργεί με αναθεωρήσεις (revisions) οπότε κάθε αλλαγή στον κώδικα θεωρείται καινούρια αναθεώρηση. Με αυτόν τον τρόπο επιτρέπει δοκιμές με προσωρινές αναθεωρήσεις πριν οι αλλαγές καταστούν μόνιμες (blue-green deployment).
- **Eventing**: Αποτελεί το κομμάτι του Knative που δημιουργεί ή αντιδρά σε γεγονότα. Τα γεγονότα αυτά μπορούν να δημιουργηθούν και να διοχετευτούν σε άλλα στοιχεία της υποδομής που ενδέχεται να βρίσκονται εκτός cluster. Ανάλογα εξωτερικά γεγονότα μπορούν να πυροδοτήσουν αλλαγές εντός του cluster. Συνεπώς, το στοιχείο του Eventing χρησιμεύει στη δημιουργία αυτοματοποιημένης ροής εργασίας.

OpenWhisk

Το Apache OpenWhisk είναι μια πλατφόρμα για serverless computing που επιτρέπει την εκτέλεση κώδικα σε μορφή συναρτήσεων. Τα βασικά στοιχεία του είναι η δημιουργία ενεργειών (Actions) και πυροδοτήσεων (Triggers), ώστε να δίνεται η δυνατότητα στον χρήστη να ορίζει ενέργειες οι οποίες μπορούν να χρονοδρομολογηθούν ή να προγραμματιστούν ως αντίδραση σε κάποιο γεγονός. Ακολουθεί επομένως συναρτησιακή λογική και επιτρέπει τη δημιουργία αλυσίδων ενεργειών που λειτουργούν σύγχρονα ή ασύγχρονα. Διαχειρίζεται μόνο του την κλιμάκωση των πόρων προσθέτοντας ή αφαιρώντας τους χωρίς την παρέμβαση ή την ενημέρωση του χρήστη [20].

OpenFaaS

Το OpenFaaS είναι μια πλατφόρμα για serverless computing που επιτρέπει την εκτέλεση κώδικα σε μορφή συναρτήσεων. Υποστηρίζει την αποδέσμευση όλων των πόρων (scale to zero) και διαχειρίζεται την κλιμάκωση με ένα δικό του στοιχείο που ονομάζεται Alert Manager και στο οποίο είναι δυνατό να οριστεί ο ελάχιστος και μέγιστος αριθμός αντιγράφων καθώς και ο παράγοντας κλιμάκωσης δηλαδή πόσο γρήγορα θα κλιμακώνεται η εφαρμογή. Εναλλακτικά γίνεται να χρησιμοποιηθεί ο HPA του Kubernetes [21].

Fission

Το Fission είναι μια πλατφόρμα για serverless computing που επιτρέπει την εκτέλεση κώδικα σε μορφή συναρτήσεων. Διαχειρίζεται την κλιμάκωση μέσω του HPA, χρησιμοποιεί τη μετρική για την υπολογιστική ισχύ και επιτρέπει την πλήρη αποδέσμευση πόρων (scale to zero). Για την εκτέλεση συναρτήσεων φορτώνονται τα περιβάλλοντα εκτέλεσης και φορτώνονται δυναμικά οι συναρτήσεις. Για την αποφυγή της ψυχρής εκκίνησης κρατά μια ομάδα “θερμών” containers για να επιταχύνεται η εκκίνηση τους όταν έρθει αίτημα [22].

Kubeless

Το Kubeless είναι μια πλατφόρμα για serverless computing που επιτρέπει την εκτέλεση κώδικα σε μορφή συναρτήσεων. Για την κλιμάκωση χρησιμοποιεί τον HPA και τις μετρικές για την υπολογιστική ισχύ (CPU) και τον αριθμό των αιτημάτων ανά δευτερόλεπτο. Οι συναρτήσεις του χρήστη πακετάρονται μαζί με το περιβάλλον εκτέλεσης σε ενα container και ύστερα εκτελούνται [23].

2.4.3 Πίνακας συγκρίσεων serverless πλατφορμών

Πλατφόρμα	Μηχανισμός	Scale to zero	Μετρικές κλιμάκωσης
Knative	KPA	Ναι	Concurrency, Requests per second
	HPA	Όχι	Concurrency, Requests per second, CPU
OpenWhisk	Αυτόνομος	Ναι	N/A
OpenFaaS	AlertManager	Ναι	Requests per second
	HPA	Όχι	CPU, Memory
Fission	HPA	Ναι	CPU
Kubeless	HPA	Όχι	CPU, Requests per second

Πίνακας 2.1: Συγκριτικός πίνακας serverless πλατφορμών

2.5 Μηχανισμοί διαχείρισης κλιμακωσιμότητας serverless εφαρμογών

Η αυτοματοποίηση της διαχείρισης κλιμακωσιμότητας είναι ένα από τα βασικά χαρακτηριστικά του serverless μοντέλου. Μια στρατηγική κλιμάκωσης ορίζει το πως θα προσαρμοστεί ο αριθμός και ο τύπος των πόρων του νέφους που έχουν δεσμευτεί για τις απαιτήσεις της εφαρμογής. Τρεις βασικοί τύποι κλιμάκωσης εφαρμόζονται: κλιμάκωση ανά αίτημα, κλιμάκωση με βάση τον ταυτοχρονισμό και κλιμάκωση με βάση τις μετρικές [24, 25].

Στην κλιμάκωση ανά αίτημα η κλιμάκωση γίνεται με βάση τον αριθμό των εισερχόμενων αιτημάτων για μια συνάρτηση. Σε αυτήν την περίπτωση ο μηχανισμός κλιμάκωσης πρέπει είναι έτοιμος να αντιδράσει οποιαδήποτε στιγμή και να είναι σε θέση να αντιμετωπίσει προβλήματα που σχετίζονται με την ψυχρή εκκίνηση. Δεν χρησιμοποιούνται ουρές για τα αιτήματα και ορίζεται ένα ανώτατο όριο για το latency κάθε αιτήματος. Στην κλιμάκωση με βάση τον ταυτοχρονισμό κάθε αντίγραφο μιας συνάρτησης μπορεί να λάβει πολλά αιτήματα ταυτόχρονα. Ορίζεται όμως ένας αριθμός μέγιστων ταυτόχρονων αιτημάτων. Όταν ο αριθμός των εισερχόμενων αιτημάτων γίνει ίσος με αυτήν την τιμή εφαρμόζεται μία κίνηση κλιμάκωσης για να εξυπηρετηθούν τα νέα αιτήματα. Η κλιμάκωση με βάση τις μετρικές προσπαθεί να κρατήσει μετρικές όπως τη χρήση της CPU, τη χρήση της μνήμης, το throughput ή το latency εντός προκαθορισμένων ορίων [24]. Αυτή η προσέγγιση υιοθετείται από αρκετές serverless πλατφόρμες ανοιχτού λογισμικού, συμπεριλαμβανομένου και του Kubeless, όπου ο Kubernetes Horizontal Pod Autoscaler (HPA) είναι υπεύθυνος για την εφαρμογή των αποφάσεων κλιμάκωσης. Η κλιμάκωση με βάση τις μετρικές δεν είναι η καλύτερη επιλογή όσον αφορά την απόδοση στα πλαίσια φόρτων εργασίας που παρουσιάζουν μεγάλες αυξομειώσεις. Ωστόσο, λαμβάνει υπόψη τόσο την απόδοση και τη χρήση των πόρων όσο και δείκτες κόστους. Συνεπώς, θεωρείται ελκυστικό προς τους παρόχους υπολογιστικών νεφών για την διαχείριση της κλιμάκωσης ώστε να εξυπηρετούν τις ανάγκες των πελατών, ενώ παράλληλα δεν οδηγεί σε δέσμευση πλεοναζόντων πόρων και σε αυξημένο κόστος.

Λαμβάνοντας υπόψη τις τρεις διαφορετικές μεθόδους κλιμάκωσης, έχουν προταθεί διάφοροι μηχανισμοί για την υποστήριξη της αυτόματης κλιμάκωσης. Τέτοιοι μηχανισμοί συμπεριλαμβάνουν στατικές μεθόδους βασισμένες σε λύσεις που χρησιμοποιούν thresholds και ανάλυση χρονοσειρών [26, 27], καθώς και περισσότερο δυναμικές προσεγγίσεις βασισμένες σε τεχνικές μηχανικής μάθησης. Στις στατικές προσεγγίσεις, το βασικό μειονέκτημα είναι η ανάγκη ορισμού των κανόνων κλιμάκωσης που μπορούν να ενημερωθούν μόνο με την παρέμβαση του διαχειριστή του συστήματος. Αυτό επιβάλλει ένα σημαντικό επιπλέον κόστος, αφού κάθε σετ κανόνων αντιστοιχεί μόνο με μια συγκεκριμένη συνάρτηση. Μια εφαρμογή που είναι βασισμένη στο μοντέλο των microservices με πολλές συναρτήσεις θα χρειαστεί τον ορισμό ενός μεγάλου σετ κανόνων, κάνοντας δύσκολο τον ορισμό και την ενημέρωση

τους σε περίπτωση αποτυχίας. Η αξιοποίηση των τεχνικών μηχανικής μάθησης είναι πολλά υποσχόμενη για την αύξηση της αυτοματοποίησης αυτής της διαδικασίας [27, 28].

Στην παρούσα εργασία για την αντιμετώπιση των προκλήσεων που αναλύθηκαν στην ενότητα 2.3.2, ακολουθούμε μια προσέγγιση βασισμένη στην κλιμάκωση με βάση τις μετρικές και επικεντρωνόμαστε στην υποστήριξη της αυτοματοποίησης μέσα από την αξιοποίηση τεχνικών Ενισχυτικής Μάθησης. Ο στόχος είναι ο συνδυασμός της υψηλής απόδοσης των εφαρμογών και της συνετής χρήσης των διαθέσιμων πόρων στο υπολογιστικό νέφος, αντισταθμίζοντας τους περιορισμένους πόρους με την επιθυμητή ποιότητα υπηρεσίας των serverless εφαρμογών [29].

3. Μηχανισμοί ενορχήστρωσης με αξιοποίηση τεχνικών ενισχυτικής μάθησης

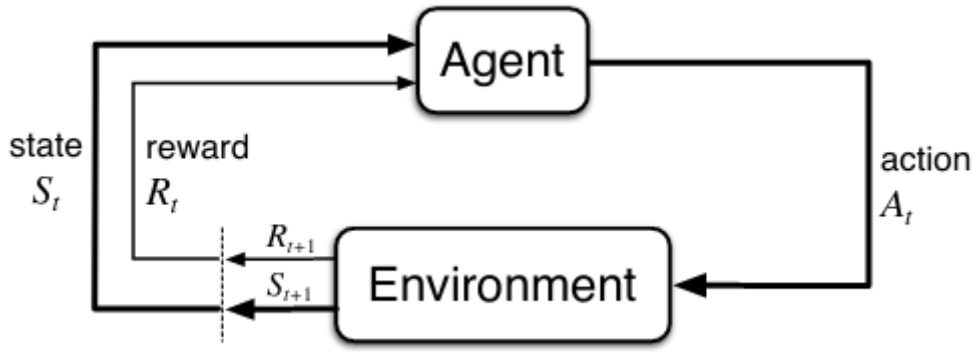
Σε αυτό το κεφάλαιο γίνεται μια εισαγωγή στην ενισχυτική μάθηση και σε βασικούς αλγορίθμους της. Συνοψίζεται η υπάρχουσα έρευνα για τη διαχείριση της κλιμακωσιμότητας σε serverless περιβάλλοντα και αναφέρονται τα εργαλεία που χρησιμοποιήθηκαν για τη μοντελοποίηση των περιβαλλόντων ενισχυτικής μάθησης.

3.1 Εισαγωγή στην ενισχυτική μάθηση

3.1.1 Ορισμός

Με τον όρο μηχανική μάθηση ορίζεται ένας κλάδος αλγορίθμων οι οποίοι μαθαίνουν συλλέγοντας εμπειρία και αξιοποιώντας δεδομένα από το περιβάλλον τους. Ο στόχος είναι η αυτοματοποίηση μιας εργασίας ώστε να εκτελείται με βέλτιστο τρόπο. Στα προβλήματα ενισχυτικής μάθησης εκπαιδεύεται ένας πράκτορας να παίρνει βέλτιστες αποφάσεις αλληλεπιδρώντας με ένα περιβάλλον.

Η βασική διαφορά με τις άλλες κατηγορίες αλγορίθμων μηχανικής μάθησης είναι ότι δεν πρόκειται για αλγόριθμους κατηγοριοποίησης και οι κινήσεις που παίρνει ο πράκτορας παράγουν μια ανταμοιβή. Ο πράκτορας στη συνέχεια πρέπει να ανακαλύψει ποιες κινήσεις θα οδηγήσουν στη μεγαλύτερη ανταμοιβή αθροιστικά.



Εικόνα 3.1: Αλληλεπίδραση πράκτορα-περιβάλλοντος στην ενισχυτική μάθηση [30]

Στην ενισχυτική μάθηση ο πράκτορας αποτελεί το στοιχείο που ευθύνεται για την λήψη των αποφάσεων. Τα υπόλοιπα στοιχεία του προβλήματος μοντελοποιούνται στο περιβάλλον. Σε κάθε χρονική στιγμή t η κατάσταση του περιβάλλοντος κωδικοποιείται σε μια αναπαράσταση S_t και με βάση αυτή ο πράκτορας επιλέγει μια ενέργεια A_t . Ένα χρονικό βήμα αργότερα το περιβάλλον αλλάζει κατάσταση η οποία πλέον αντικατοπτρίζεται στη νέα κατάσταση S_{t+1} και ο πράκτορας λαμβάνει μια ανταμοιβή R_{t+1} η οποία συμβολίζει το πόσο καλή ήταν η κίνηση που πήρε [30].

Τα προβλήματα ενισχυτικής μάθησης μοντελοποιούνται ως Μαρκοβιανές Διαδικασίες Επιλογής (Markov Decision Process/MDP). Αυτό σημαίνει ότι ικανοποιούν τη μαρκοβιανή ιδιότητα η οποία ορίζει ότι η επόμενη κατάσταση εξαρτάται μόνο από την τρέχουσα κατάσταση και την εκάστοτε κίνηση και όχι από τις παρελθοντικές κινήσεις και καταστάσεις. Πρακτικά δηλαδή δεν έχει σημασία με ποιον τρόπο ή με ποιες μεταβάσεις έφτασε το σύστημα στην τρέχουσα κατάσταση. Πέρα από τις παραπάνω πληροφορίες ο πράκτορας κατασκευάζει μια αντιστοίχιση ανάμεσα στις καταστάσεις και στις πιθανές κινήσεις. Αυτή η αντιστοίχιση ονομάζεται πολιτική και συμβολίζεται με π_t.

Στα προβλήματα ενισχυτικής μάθησης μια βασική πρόκληση που συναντάται αρκετά συχνά είναι η ισορροπία μεταξύ εξερεύνησης (exploration) και εκμετάλλευσης (exploitation). Η εξερεύνηση αναφέρεται στη δοκιμή καινούριων κινήσεων ώστε ο αλγόριθμος να εξερευνήσει περισσότερες πιθανές καταστάσεις και κινήσεις με την ελπίδα μεγαλύτερης ανταμοιβής. Αντίθετα η εκμετάλλευση στοχεύει στην αξιοποίηση της μέχρι τώρα συσσωρευμένης γνώσης ώστε η επιλογή για την επόμενη κίνηση να γίνεται με βέλτιστο τρόπο. Συνηθώς στα αρχικά στάδια απαιτείται επαρκής εξερεύνηση ώστε ο πράκτορας να επισκεφτεί αρκετές καταστάσεις και να συλλέξει αρκετά δεδομένα προτού αρχίσει να εκμεταλλεύεται τη γνώση που απέκτησε. Μια πιθανή υλοποίηση αυτής της επιλογής είναι η πολιτική ε-greedy η οποία ορίζει ότι με μια μικρή πιθανότητα ε επιλέγουμε την επόμενη κίνηση τυχαία ενώ με πιθανότητα 1-ε επιλέγουμε τη βέλτιστη κίνηση.

3.2.2 Συναρτήσεις αξίας

Οι συναρτήσεις αξίας (value functions) ορίζονται ως η αναμενόμενη ανταμοιβή που θα πάρει ο πράκτορας και χωρίζονται σε δύο κατηγορίες: τις συναρτήσεις καταστάσεων και τις συναρτήσεις καταστάσεων-κινήσεων. Αυτές οι δύο κατηγορίες ποσοτικοποιούν το πόσο επιθυμητό είναι να βρίσκεται το περιβάλλον σε μια κατάσταση ή ποσο επιθυμητή είναι μια ενέργεια δεδομένου ότι βρισκόμαστε σε μια συγκεκριμένη κατάσταση.

Η συνάρτηση καταστάσεων ορίζεται ως

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s]$$

όπου:

- E_π: αναμενόμενη τιμή
- G_t: ελαττωμένη ανταμοιβή
- S_t: τρέχουσα κατάσταση
- γ: συντελεστής ελάττωσης
- R: ανταμοιβή

Η συνάρτηση καταστάσεων-κινήσεων αντίστοιχα ορίζεται ως

$$q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$$

όπου:

- A_t : τρέχουσα κίνηση

Στόχος του πράκτορα είναι η μεγιστοποίηση των συναρτήσεων. Ο συντελεστής ελάττωσης καθορίζει κατά πόσο συνεισφέρουν οι μελλοντικές ανταμοιβές στην τρέχουσα συνάρτηση αξίας. Όταν ισούται με 1 τότε όλες οι ανταμοιβές συνεισφέρουν ισάξια στη συνολική ανταμοιβή ενώ όσο προσεγγίζει το 0 οι μελλοντικές ανταμοιβές συνεισφέρουν λιγότερο.

Σε πεπερασμένες μαρκοβιανές διαδικασίες επιλογής (Finite MDPs) μπορούμε να ορίσουμε μια βέλτιστη πολιτική η οποία επιστρέφει τη μέγιστη ανταμοιβή. Η αντίστοιχη συνάρτηση καταστάσεων ορίζεται ως

$$u_*(s) = \max_\alpha E[R_{t+1} + \gamma u_*(S_{t+1})|S_t = s, A_t = a]$$

και η αντίστοιχη συνάρτηση καταστάσεων-κινήσεων ορίζεται ως

$$q_*(s,a) = E[R_{t+1} + \gamma \max_a q_*(S_{t+1}, a')|S_t = s, A_t = a]$$

Αυτές οι εξισώσεις ονομάζονται εξισώσεις βελτιστότητας Bellman και χρησιμοποιούνται για την εύρεση της βέλτιστης πολιτικής.

3.2 Αλγόριθμοι ενισχυτικής μάθησης

3.2.1 Βασικά χαρακτηριστικά αλγορίθμων

Οι αλγόριθμοι ενισχυτικής μάθησης μπορούν να κατηγοριοποιηθούν με διάφορους τρόπους. Μερικά από τα χαρακτηριστικά τους αποτελούν τα εξής:

- **Χώρος καταστάσεων**: Ανάλογα με την κωδικοποίηση των καταστάσεων ο χώρος μπορεί να είναι είτε συνεχής είτε διακριτός.
- **Χώρος κινήσεων**: Ανάλογα με την κωδικοποίηση των κινήσεων ο χώρος μπορεί να είναι είτε συνεχής είτε διακριτός.
- **Πολιτική**: Αν ο αλγόριθμος ενημερώνει τις τιμές του με βάση την ίδια πολιτική που εφαρμόζει τότε ονομάζεται On-policy ενώ αν ενημερώνει τις τιμές του με βάση τη βέλτιστη πολιτική τότε ονομάζεται Off-policy.

- **Μοντέλο**: Αν ο αλγόριθμος κατασκευάζει ένα μοντέλο ώστε να μπορεί να προβλέψει τις μεταβάσεις του περιβάλλοντος τότε ονομάζεται model-based. Αν βασίζεται σε δοκιμές (trial and error) τότε ονομάζεται model-free.
- **Online vs offline**: Ανάλογα με τον τρόπο συλλογής των δεδομένων η μάθηση του αλγορίθμου χαρακτηρίζεται είτε ως online είτε ως offline. Η online μάθηση αναφέρεται στη συλλογή δεδομένων, αλλαγή της πολιτικής και στη συνέχεια εκ νέου συλλογή δεδομένων αλλά με την καινούργια πολιτική. Αντιθέτως στην offline μάθηση τα δεδομένα συλλέγονται μία φορά με κάποια αυθαίρετη πολιτική και οι αλλαγές στην πολιτική γίνεται με αυτά τα αρχικά δεδομένα.

3.2.2 Δυναμικός προγραμματισμός

Ο δυναμικός προγραμματισμός είναι μια μέθοδος επίλυσης μαθηματικών και υπολογιστικών προβλημάτων. Χρησιμοποιείται για την επίλυση πεπερασμένων μαρκοβιανών διαδικασιών επιλογής (finite MDPs) παρόλο που δεν προϋποθέτει ένα τέλειο μοντέλο του περιβάλλοντος. Δύο βασικές προσεγγίσεις του δυναμικού προγραμματισμού αποτελούν η επανάληψη πολιτικής (policy iteration) και η επανάληψη αξίας (value iteration). Και οι δύο βασίζονται στις συναρτήσεις βελτιστότητας Bellman και αποσκοπούν στην προσέγγιση της βέλτιστης πολιτικής ή συνάρτησης αξίας μέσω διαδοχικών επαναλήψεων και ενημερώσεων. Όταν επιτευχθεί σύγκλιση, οι αλγόριθμοι τερματίζουν. Μια ενδιαφέρουσα ιδιότητα αυτών των μεθόδων είναι η χρήση μελλοντικών εκτιμήσεων για την ενημέρωση τωρινών τιμών. Αυτή η μέθοδος ονομάζεται bootstrapping.

3.2.3 Μέθοδοι Monte Carlo

Οι μέθοδοι Monte Carlo είναι μια προσέγγιση η οποία βασίζεται στη συλλογή δεδομένων με τη μορφή δειγμάτων και πιο συγκεκριμένα επεισοδίων δειγμάτων. Αυτό σημαίνει ότι η συλλεγόμενη εμπειρία χωρίζεται σε πολλαπλά βήματα τα οποία ονομάζονται επεισόδιο. Συνεπώς οι αλλαγές στην πολιτική και στις συναρτήσεις αξίας ομαδοποιούνται ανά επεισόδιο αντί για κάθε βήμα. Οι βασικές διαφορές με τον δυναμικό προγραμματισμό είναι η χρήση εμπειρίας με τη μορφή δειγμάτων για την οποία δεν χρειάζεται μοντέλο του περιβάλλοντος και η έλλειψη της μεθόδου bootstrapping δηλαδή οι εκτιμήσεις δεν βασίζονται πάνω σε άλλες εκτιμήσεις.

3.2.4 Μέθοδοι Temporal difference

Οι μέθοδοι Temporal difference (TD) συνδυάζουν στοιχεία από το δυναμικό προγραμματισμό και τις μεθόδους Monte Carlo. Συγκεκριμένα χρησιμοποιούν δείγματα εμπειρίας όπως οι μέθοδοι Monte Carlo αλλά ενημερώνουν τις εκτιμήσεις τους μετά από κάθε χρονικό βήμα χωρίς να περιμένουν το τέλος του επεισοδίου. Αντίστοιχα χρησιμοποιούν τη μέθοδο bootstrapping όπως ο δυναμικός

προγραμματισμός. Στη συνέχεια παρουσιάζονται δύο τέτοιοι αλγόριθμοι: ο Q-learning και ο DynaQ+.

3.2.4.1 Q-learning

Q-learning ονομάζεται ένας αλγόριθμος Temporal difference ο οποίος δε χρησιμοποιεί μοντέλο του περιβάλλοντος και επομένως ονομάζεται model-free. Στόχος του είναι η εύρεση της βέλτιστης πολιτικής η οποία μεγιστοποιεί την ανταμοιβή σε διαδοχικά βήματα δεδομένου ότι ξεκινά από μια οποιαδήποτε κατάσταση. Το γράμμα Q αναφέρεται στη συνάρτηση καταστάσεων-κινήσεων. Επομένως ο αλγόριθμος στηρίζεται στη συνάρτηση βελτιστότητας Bellman για να υπολογίσει την αναμενόμενη ανταμοιβή για την εφαρμογή μιας κίνησης όντας σε κάποια κατάσταση.

Η βασική συνάρτηση ενημέρωσης ορίζεται ως

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{\alpha} Q(S_{t+1}, \alpha) - Q(S_t, A_t)]$$

όπου:

- S_t : τρέχουσα κατάσταση
- A_t : τρέχουσα κίνηση
- $Q(S_t, A_t)$: αναμενόμενη ανταμοιβή αν επιλεγεί η κίνηση A_t στην κατάσταση S_t
- α : ρυθμός μάθησης. Όσο πλησιάζει το 1, οι αλλαγές γίνονται πιο γρήγορα.
- R_{t+1} : ανταμοιβή για την τρέχουσα κίνηση
- γ : συντελεστής ελάττωσης

Ακολούθως, παρουσιάζεται η βασική λειτουργία του αλγορίθμου. Μετά την αρχικοποίηση των τιμών Q, ξεκινά ο αλγόριθμος να βελτιώνει επαναληπτικά αυτές τις τιμές. Συγκεκριμένα για κάθε χρονικό βήμα επιλέγει μια κίνηση με βάση κάποια πολιτική (π.χ., ε-greedy) και ενημερώνει την τιμή με βάση την επόμενη κατάσταση και την ανταμοιβή που πήρε.

Αλγόριθμος 3.1: Ψευδοκώδικας αλγορίθμου Q-learning [30]

```
Algorithm parameters: step size α∈(0,1], small ε >0
Initialize Q(s,a), ∀s∈S+, α∈A(s), arbitrarily, and Q(terminal-state,·) = 0
Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R,S′
        Q(S,A) ← Q(S,A) + α [R + γ max_α Q(S′,α) - Q(S,A)]
        S←S′;
    until S is terminal
```

3.2.4.2 DynaQ και DynaQ+

Ο αλγόριθμος DynaQ αποτελεί επέκταση του απλού Q-learning. Αποτελείται από δύο βασικά στοιχεία. Την άμεση ενισχυτική μάθηση και τη μάθηση μέσω μοντέλου ή αλλιώς έμμεση ενισχυτική μάθηση. Το κομμάτι της άμεσης ενισχυτικής μάθησης είναι αντίστοιχο με την ενημέρωση στον αλγόριθμο Q-learning δηλαδή προϋποθέτει την αλληλεπίδραση με το περιβάλλον και έπειτα την ενημέρωση των τιμών. Κατά τη διάρκεια εκτέλεσης αυτού του βήματος κρατάμε την μετάβαση από την κατάσταση S στην S′ μέσω της κίνησης A δεδομένου ότι αυτή η κίνηση ανταμείβεται με αμοιβή R. Στη συνέχεια με βάση τις αποθηκευμένες μεταβάσεις του μοντέλου εκτελείται ένα στάδιο σχεδιασμού (planning) με το οποίο εννοείται η προσομοίωση εμπειρίας χρησιμοποιώντας τις μεταβάσεις του μοντέλου για την ενημέρωση των τιμών. Οπότε σε ένα ντετερμινιστικό περιβάλλον ο αλγόριθμος DynaQ βρίσκει πιο γρήγορα τη βέλτιστη πολιτική.

Ακολούθως, παρουσιάζεται η βασική λειτουργία του αλγορίθμου. Σε κάθε βήμα εκτελείται η βασική ενημέρωση του Q-learning αλγορίθμου. Στη συνέχεια προστίθεται η μετάβαση που παρατηρήθηκε στο μοντέλο. Τέλο εκτελούνται n βήματα σχεδιασμού στα οποία προσομοιώνεται εμπειρία χρησιμοποιώντας το μοντέλο χωρίς να αλληλεπιδρά ο αλγόριθμος με το περιβάλλον.

Αλγόριθμος 3.2: Ψευδοκώδικας αλγορίθμου DynaQ [30]

```
Initialize Q(s,a) and Model(s,a) for all s∈S and a∈A(s)
Loop forever:
    (a) S ← current (nonterminal) state
    (b) A ← ε-greedy(S,Q)
    (c) Take action A; observe resultant reward, R, and state,S′
    (d) Q(S,A) ← Q(S,A) + α [R + γ max_α Q(S′,α) - Q(S,A)]
    (e) Model(S,A) ← R,S′ (assuming deterministic environment)
    (f) Loop repeat n times:
        S ← random previously observed state
        A ← random action previously taken in S
        R,S′ ← Model(S,A)
        Q(S,A) ← Q(S,A) + α [R + γ max_α Q(S′,α) - Q(S,A)]
```

Ο αλγόριθμος DynaQ+ αποτελεί επέκταση του αλγορίθμου DynaQ. Πέρα από τις τιμές Q και το μοντέλο κρατάμε και τις τιμές τ για κάθε κίνηση σε κάθε κατάσταση. Συγκεκριμένα η τιμή τ για μια κίνηση σε μία κατάσταση αντιστοιχεί στον αριθμό των χρονικών βημάτων που έχουν περιέλθει από τότε που δοκιμάστηκε η κίνηση. Στη συνέχεια κατά τη διάρκεια του σχεδιασμού προστίθεται στην ανταμοιβή αυτής της κίνησης ένα μπόνους που είναι μεγαλύτερο όσο περισσότερα χρονικά βήματα έχουν περάσει από τότε που δοκιμάστηκε η κίνηση. Με αυτόν τον τρόπο επιβραβεύεται η δοκιμή κινήσεων που δεν έχουν δοκιμαστεί για αρκετά χρονικά βήματα.

Ακολούθως, παρουσιάζεται η βασική λειτουργία του αλγορίθμου. Μετά από μια μετάβαση μηδενίζουμε την τιμή τ του ζευγαριού τρέχουσας κατάστασης-κίνησης και αυξάνουμε όλες τις υπόλοιπες τιμές όπως φαίνεται στα βήματα (d) και (e). Στη συνέχεια στα βήματα σχεδιασμού αυξάνουμε την ανταμοιβή κατά ένα μπόνους κ * √τ. Το κ είναι μια υπερπαράμετρος η οποία παίρνει μια αυθαίρετη μικρή τιμή.

Αλγόριθμος 3.3: Ψευδοκώδικας αλγορίθμου DynaQ+ [30]

```
Initialize Q(s,a), τ(s,a) and Model(s,a) for all s∈S and a∈A(s)
Loop forever:
    (a) S ← current (nonterminal) state
    (b) A ← ε-greedy(S,Q)
    (c) Take action A; observe resultant reward, R, and state,S′
    (d) τ(s,a) ← τ(s,a) + 1 for all s∈S and a∈A(s)
    (e) τ(S,A) ← 0
    (f) Q(S,A) ← Q(S,A) + α [R + γ max_α Q(S′,α) - Q(S,A)]
    (g) Model(S,A) ← R,S′ (assuming deterministic environment)
    (h) Loop repeat n times:
        S ← random previously observed state
        A ← random action previously taken in S
        R,S′ ← Model(S,A)
        R ← R + κ √τ(S, A)
        Q(S,A) ← Q(S,A) + α [R + γ max_α Q(S′,α) - Q(S,A)]
```

3.2.4.3 Deep Q-learning

Ο αλγόριθμος Deep Q-learning αποτελεί παραλλαγή του Q-learning. Ο βασικός αλγόριθμος Q-learning συνήθως υλοποιείται με τη χρήση ενός πίνακα και επομένως για την υλοποίηση του απαιτεί την ύπαρξη διακριτού χώρου καταστάσεων και διακριτού χώρου κινήσεων. Όταν ο χώρος καταστάσεων του προβλήματος είναι συνεχής απαιτείται η μετατροπή του σε διακριτό με αποτέλεσμα όμως την απώλεια πληροφορίας. Μια αντιμετώπιση αυτού του προβλήματος είναι η χρήση νευρωνικών δικτύων ώστε να είναι εφικτή η χρήση συνεχούς χώρου καταστάσεων. Ο Deep Q-learning είναι λοιπόν η υλοποίηση του Q-learning όπου αντικαθιστάται ο πίνακας με τις Q τιμές από ένα βαθύ νευρωνικό δίκτυο [31].

Η χρήση ενός νευρωνικού δικτύου ενέχει κάποια προβλήματα που προκαλούν αστάθεια στη συμπεριφορά του πράκτορα. Αυτή η αστάθεια προκαλείται κυρίως από τη συσχέτιση των δεδομένων εισόδου όταν αυτά εισάγονται σειριακά και από τις μεγάλες αλλαγές στην πολιτική που ενδέχεται να εμφανιστούν όταν ενημερώνουμε τις Q τιμές. Ο Deep Q-learning αντιμετωπίζει αυτά τα δύο προβλήματα με τη χρήση δύο τεχνικών: την επανάληψη εμπειρίας (experience replay) και τη χρήση ενός δικτύου στόχου (target network). Η επανάληψη εμπειρίας πραγματοποιείται με τη συλλογή τυχαίων παρελθοντικών δειγμάτων αντί των πιο πρόσφατων για να αποφευχθεί η συσχέτιση των δειγμάτων εισόδου. Το δίκτυο στόχος είναι ένα νευρωνικό δίκτυο το οποίο

ενημερώνεται περιοδικά και καθυστερεί σε σχέση με το βασικό δίκτυο με τις Q τιμές. Στόχος αυτής της μεθόδου είναι η αύξηση της σταθερότητας και η αποφυγή μεγάλων διακυμάνσεων κατά τη διάρκεια της εκπαίδευσης [31].

3.3 Αξιοποίηση τεχνικών ενισχυτικής μάθησης για διαχείριση της κλιμακωσιμότητας serverless εφαρμογών

Η ενισχυτική μάθηση έχει δείξει μεγάλες προοπτικές για την αυτόματη επίλυση προβλημάτων λήψης αποφάσεων σε περίπλοκα, αβέβαια περιβάλλοντα. Θεωρείται μια πολλά υποσχόμενη προσέγγιση για την διαχείριση της αυτόματης κλιμάκωσης στο υπολογιστικό νέφος [27] σε σύγκρισης με τις υπάρχουσες πιο στατικες προσεγγίσεις που βασίζονται σε λύσεις που υλοποιούνται με thresholds και ανάλυση χρονοσειρών. Η ενισχυτική μάθησης προσφέρει τη δυνατότητα προσαρμογής των πολιτικών κλιμάκωσης για να εγγυηθεί την ικανοποίηση της ποιότητας υπηρεσίας (Quality of Service/QoS) στα πλαίσια διαφόρων προβλημάτων που σχετίζονται με την απόδοση [32]. Εφόσον η μάθηση των πολιτικών επιτυγχάνεται μέσω της αλληλεπίδρασης με το περιβάλλον, δεν απαιτείται ανθρώπινη παρέμβαση. Η δυναμικότητα και η προσαρμοστικότητα υποστηρίζονται, αφού η διαδικασία μάθησης είναι συνεχής και οι παραχθείσες πολιτικές μπορούν να μεταβληθούν σύμφωνα με τις αλλαγές που συμβαίνουν στο περιβάλλον του υπολογιστικού νέφους [27, 32].

Διάφορες προσεγγίσεις αυτόματης κλιμάκωσης υποστηριζόμενες από ενισχυτική μάθηση έχουν προταθεί στη βιβλιογραφία [27, 32]. Στις περισσότερες τέτοιες λύσεις για περιβάλλοντα υπολογιστικού νέφους, παράγονται model-free πράκτορες, με τον Q-learning, τον SARSA και τον Deep Q-learning να είναι οι πιο ευρέως χρησιμοποιούμενοι αλγόριθμοι ενισχυτικής μάθησης [27]. Ανάμεσα στις υπάρχουσες προσεγγίσεις, ελάχιστες από αυτές θεωρούνται πειραματισμοί πάνω σε πραγματική υποδομή, ενώ πολύ λίγες από αυτές αναφέρονται σε πραγματικά περιβάλλοντα. Μια τέτοια προσέγγιση ερευνάται στο [25], όπου ένα μοντέλο ενισχυτικής μάθησης βασισμένο στο Q-learning - για την μάθηση αποδοτικών πολιτικών αυτόματης κλιμάκωσης με βάση τον αριθμό των αιτημάτων - έχει σχεδιαστεί και αξιολογηθεί με βάση τη χρήση από την serverless πλατφόρμα του Knative. Καταδεικνύεται ότι το προτεινόμενο μοντέλο μπορεί να προσαρμόσει τον ταυτοχρονισμό κατάλληλα χωρίς πρότερη γνώση εντός περιορισμένου χρόνου και υπερνικά την προεπιλεγμένη ρύθμιση του Knative σε ότι αφορά το throughput [25]. Στο [32] προτείνεται μια πλατφόρμα κλιμάκωσης πόρων που χρησιμοποιεί βαθιά ενισχυτική μάθηση και εφαρμόζεται σε serverless εφαρμογές. Η προτεινόμενη λύση αξιοποιεί ένα κομμάτι ανίχνευσης ανωμαλιών για να ανιχνεύει τα επίμονα προβλήματα απόδοσης στο σύστημα ως έναυσμα για το κομμάτι λήψης αποφάσεων της ενισχυτικής μάθησης ώστε να κλιμακώσει την εφαρμογή και να επιλυθεί το πρόβλημα.

Στην παρούσα εργασία προσπαθούμε να γεφυρώσουμε το χάσμα στον ορισμό και την ανάπτυξη περιβαλλόντων για serverless εφαρμογές. Λίγες ερευνητικές

προσπάθειες είναι διαθέσιμες στη βιβλιογραφία για αυτό το σκοπό. Επιπλέον, το αντικείμενο της εργασίας, δηλαδή η δυναμική κλιμάκωση με βάση τις μετρικές (CPU, μνήμη) -σε σχέση με την κλιμάκωση ανά αίτημα και την κλιμάκωση με βάση τον ταυτοχρονισμό- δεν έχει εξερευνηθεί στο πλαίσιο serverless εφαρμογών. Παρουσιάζουμε λοιπόν τον τρόπο με τον οποίο προσεγγίζουμε τη μοντελοποίηση των περιβαλλόντων και αναδεικνύουμε την πιθανότητα για μελλοντικές μετατροπές και επεκτάσεις. Επικεντρωνόμαστε στην ανάπτυξη πρακτόρων ενισχυτικής μάθησης βασισμένους στους αλγόριθμους Q-learning, DynaQ+ και Deep Q-Network(DQN)/Deep Q-learning και αναλύουμε τα αποτελέσματα των δοκιμών μας.

3.4 Βασικά εργαλεία μοντελοποίησης περιβαλλόντων και πρακτόρων ενισχυτικής μάθησης

3.4.1 OpenAI Gym

Το OpenAI Gym είναι ένα εργαλείο για τη μοντελοποίηση περιβαλλόντων και την ανάπτυξη αλγορίθμων ενισχυτικής μάθησης. Πέρα από αρκετά ήδη μοντελοποιημένα προβλήματα τα οποία είναι διαθέσιμα με τη βιβλιοθήκη, το Gym επιτρέπει τη δημιουργία προσαρμοσμένων περιβαλλόντων αρκεί να ακολουθούν μια πρότυπη δομή [33, 34]. Συγκεκριμένα μερικές συναρτήσεις που πρέπει ή μπορεί προαιρετικά να υλοποιήσει ένα περιβάλλον είναι οι εξής:

- **reset**: Η συνάρτηση reset επαναφέρει το περιβάλλον στην αρχική κατάσταση του. Η αρχικές συνθήκες ορίζονται αυθαίρετα από τον χρήστη.
- **step**: Η συνάρτηση step δέχεται μια κίνηση και είναι υπεύθυνη για την εφαρμογή της στο περιβάλλον. Αφότου εφαρμοστούν οι αλλαγές, η συνάρτηση επιστρέφει τέσσερα στοιχεία: την παρατήρηση (observation), την ανταμοιβή (reward), μια μεταβλητή που δείχνει αν η εκτέλεση ολοκληρώθηκε (done) και μια μεταβλητή με αυθαίρετες πληροφορίες από το χρήστη (info). Η παρατήρηση περιέχει μετρικές που συλλέχθηκαν από το περιβάλλον και αντικατοπτρίζουν την κατάσταση του μετά τις αλλαγές. Αντίστοιχα η ανταμοιβή υπολογίζεται ως συνάρτηση αυτών των αλλαγών.
- **render**: Η συνάρτηση render υλοποιείται προαιρετικά και καθιστά εφικτή την αναπαράσταση του περιβάλλοντος για παράδειγμα με γραφικό τρόπο.
- **close**: Η συνάρτηση close φροντίζει για την αποδέσμευση πόρων όταν τελειώσει η εκτέλεση του περιβάλλοντος.

3.4.2 Tensorflow και Tensorflow Agents

Το Tensorflow είναι μια βιβλιοθήκη λογισμικού που χρησιμοποιείται στη μηχανική μάθηση. Ειδικεύεται στη δημιουργία και εκπαίδευση βαθιών νευρωνικών δικτύων [35].

Το Tensorflow Agents είναι μια βιβλιοθήκη για τη δημιουργία πρακτόρων ενισχυτικής μάθησης. Περιέχει βασικές υλοποιήσεις αλγορίθμων βαθιάς ενισχυτικής μάθησης (Deep Reinforcement Learning). Ο πράκτορας έχει δύο βασικές υποχρεώσεις: να ορίσει μια πολιτική με την οποία θα αλληλεπιδρά με το περιβάλλον και να βελτιώσει/εκπαιδεύσει αυτήν την πολιτική με βάση την εμπειρία που συνέλεξε [36].

3.5 Βασικά εργαλεία ανάπτυξης υποδομής

3.5.1 Kubeless

Το Kubeless όπως αναφέρεται και στην ενότητα 2.4.2.5 είναι μια serverless πλατφόρμα. Εντός του cluster αποτελεί το στοιχείο που ευθύνεται για τη δημιουργία συναρτήσεων και τη διάθεση τους ως υπηρεσίες μέσα από τη δημιουργία των κατάλληλων pods και Kubernetes Services [23].

3.5.2 Metrics server

Ο Kubernetes metrics server είναι ένα εργαλείο που επιτρέπει τη συλλογή μετρικών για τους χρησιμοποιούμενους πόρους. Συγκεκριμένα επιστρέφει την τρέχουσα χρήση CPU και μνήμης και χρησιμοποιείται από τον HPA για τη συλλογή μετρικών [37].

3.5.3 NGINX Ingress Controller

Στην πλατφόρμα του Kubernetes η προώθηση των αιτημάτων γίνεται με τη χρήση ενός ingress controller. Ένας ingress controller είναι υπεύθυνος για την προώθηση των εξωτερικών αιτημάτων στην κατάλληλη υπηρεσία εντός του cluster. Στην παρούσα εργασία χρησιμοποιήθηκε για την δημιουργία HTTP triggers για τις συναρτήσεις του Kubeless ώστε να είναι προσβάσιμες και εκτός του cluster [38].

3.5.4 Prometheus

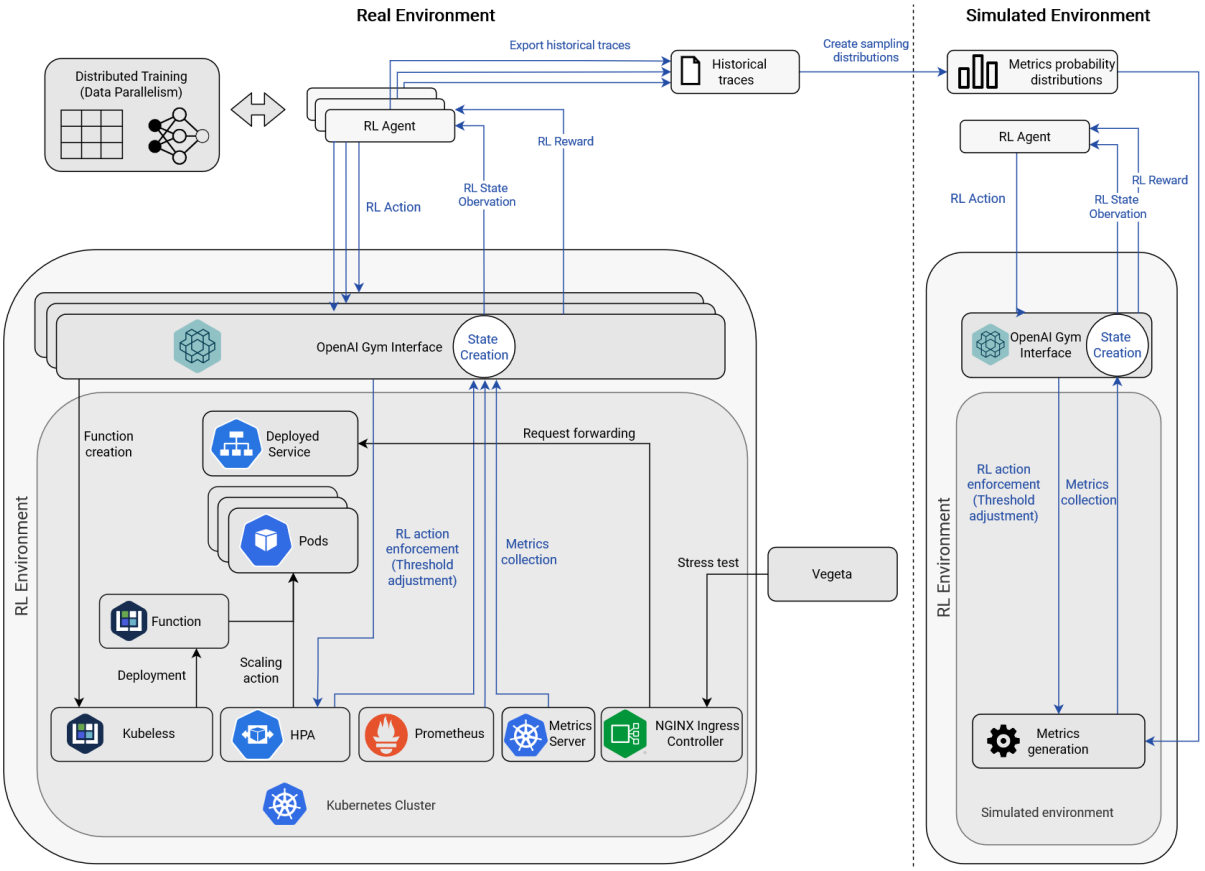
Το Prometheus είναι ένα εργαλείο για τη συλλογή διαφόρων μετρικών με την μορφή χρονοσειρών. Στην παρούσα εργασία χρησιμοποιήθηκε σε συνδυασμό με τις μετρικές που παρέχει ο NGINX Ingress Controller για τον υπολογισμό του latency και του throughput [39].

3.5.5 Vegeta

Το Vegeta είναι ένα εργαλείο παραγωγής φόρτου HTTP. Στην παρούσα εργασία χρησιμοποιήθηκε για τη δημιουργία φόρτου στις εφαρμογές μας [40].

3.6 Υποδομή δοκιμών και αρχιτεκτονική

Στην εικόνα 3.2 παρουσιάζεται εποπτικά η υποδομή που χρησιμοποιήθηκε για την εργασία. Εντός του Kubernetes Cluster υπάρχουν τα εργαλεία Kubeless, Metrics Server, Prometheus και NGINX Ingress Controller. Το Gym παρέχει ένα interface για το περιβάλλον μας και είναι υπεύθυνο για τη δημιουργία Kubeless συναρτήσεων, την εφαρμογή των κινήσεων του πράκτορα και την επιστροφή της τρέχουσα κατάστασης και ανταμοιβής. Στο εσωτερικό του cluster ο HPA είναι υπεύθυνος για τον αριθμό των αντιγράφων (pods) της εφαρμογής και πάνω του εφαρμόζονται οι κινήσεις του πράκτορα. Ο HPA μαζί με το Prometheus και τον Metrics server παρέχουν τις απαραίτητες μετρικές για την περιγραφή της κατάστασης. Τέλος, με το Vegeta δημιουργούμε διαφορετικό φόρτο στην εφαρμογή μέσω HTTP αιτημάτων τα οποία προωθούνται από τον NGINX Ingress Controller. Στόχος της υποδομής είναι η εκπαίδευση του πράκτορα ώστε να παίρνει τις αποφάσεις που οδηγούν στη μέγιστη ανταμοιβή ελαχιστοποιώντας τους πόρους (pods) χωρίς πτώση στην απόδοση (latency, throughput). Παράλληλα με το πραγματικό περιβάλλον δοκιμάστηκε και μια προσομοίωσή του στο οποίο οι μετρικές παράγονται από κατανομές πιθανοτήτων που κατασκευάστηκαν με βάση το πραγματικό περιβάλλον. Η λειτουργία του αναλύεται εκτενέστερα σε επόμενο κεφάλαιο.



Εικόνα 3.2: Αρχιτεκτονική υποδομής της εργασίας

4. Ανάπτυξη περιβαλλόντων ενισχυτικής μάθησης

4.1 Σύντομη επισκόπηση των περιβαλλόντων ενισχυτικής μάθησης

Η παρούσα εργασία επικεντρώνεται στη δημιουργία ενός περιβάλλοντος ενισχυτικής μάθησης σε serverless αρχιτεκτονική με στόχο την αυτοματοποίηση της κλιμακωσιμότητας. Αρχικά, ορίζουμε κάποια όρια χρησιμοποίησης πόρων για τις εφαρμογές στον HPA και δημιουργούμε έναν πράκτορα ενισχυτικής μάθησης ο οποίος αυξάνει ή μειώνει αυτά τα όρια ώστε να κάνει βέλτιστη διαχείριση των πόρων χωρίς να επηρεάζεται αρνητικά η απόδοση.

Τα περιβάλλοντα δοκιμάστηκαν με μια εφαρμογή που υπολογίζει τον v-οστό αριθμό Fibonacci και στα πλαίσια της εκπαίδευσης του πράκτορα εφαρμόστηκαν διαφορετικοί φόρτοι αιτημάτων.

Για τη μοντελοποίηση του περιβάλλοντος χρησιμοποιήθηκε το OpenAI Gym. Τα περιβάλλοντα που δημιουργήθηκαν στα πλαίσια αυτής της εργασίας χωρίζονται σε δύο βασικές κατηγορίες: τα συνεχή χώρου και τα διακριτού χώρου καταστάσεων. Τα περιβάλλοντα με συνεχή χώρο καταστάσεων επιστρέφουν την κατάσταση του περιβάλλοντος ως συνεχείς τιμές ενώ στα περιβάλλοντα με διακριτό χώρο καταστάσεων η κατάσταση παίρνει διακριτές τιμές, κωδικοποιείται σε μία μεταβλητή και επομένως οι πιθανές καταστάσεις είναι πεπερασμένες.

Οι αλγόριθμοι που εφαρμόστηκαν για τη δημιουργία πρακτόρων ενισχυτικής μάθησης είναι ο απλός Q-learning με τη χρήση πίνακα για τις τιμές Q κάθε κατάστασης (tabular Q-learning), ο αλγόριθμος DynaQ+ ο οποίος αποτελεί επέκταση του προηγούμενου αλγορίθμου με τη χρήση μοντέλου και την προσθήκη βημάτων σχεδιασμού και προσομοιωμένης εμπειρίας και τέλος ο αλγόριθμος βαθιάς ενισχυτικής μάθησης (DQN/Deep Q-learning) που αποτελεί προσαρμογή του αλγορίθμου Q-learning με χρήση ενός νευρωνικού δικτύου για τις Q τιμές.

Ένα βασικό πρόβλημα που προέκυψε κατά τη διάρκεια της εργασίας ήταν ο μεγάλος απαιτούμενος χρόνος εκπαίδευσης που οφείλεται στο γεγονός ότι πρέπει να περιμένουμε να σταθεροποιηθεί το περιβάλλον. Αυτό είναι αναγκαίο για να είναι σίγουρο ότι οι αλλαγές εφαρμόστηκαν και η κατάσταση του περιβάλλοντος έχει αλλάξει ώστε να αντικατοπτρίζει τις αλλαγές που επιφέρουν οι κινήσεις του πράκτορα. Συνεπώς, πέρα από τα περιβάλλοντα που αλληλεπιδρούν με την υποδομή δημιουργήθηκαν και παραλλαγές που χρησιμοποιούν τεχνικές προσομοίωσης για την αποφυγή του μεγάλου χρόνου αναμονής.

Οι βασικές μετρικές που χρησιμοποιήθηκαν για τον έλεγχο της απόδοσης της εφαρμογής είναι το latency δηλαδή ο χρόνος που χρειάζεται από τη δημιουργία ενός

αιτήματος στην εφαρμογή μέχρι την εξυπηρέτηση του και το throughput. Συγκεκριμένα για το throughput θεωρούμε δύο ξεχωριστές μετρικές από τις οποίες η μία δείχνει το ποσοστό των επιτυχημένων αιτημάτων στην εφαρμογή (success ratio) και η άλλη το ρυθμό εξυπηρέτησης των αιτημάτων δηλαδή επιτυχημένα αιτήματα ανά δευτερόλεπτο (throughput). Αυτές οι μετρικές χρησιμοποιήθηκαν για τον υπολογισμό της ανταμοιβής και συγκεκριμένα η ανταμοιβή είναι ψηλότερη όταν οι συγκεκριμένες μετρικές είναι εντός των προκαθορισμένων ορίων (όπως ορίζονται από το SLA) ενώ παίρνουν πολύ χαμηλές τιμές όταν έχουμε παράβαση τους. Τέλος, στην ανταμοιβή συνεισφέρει και ο αριθμός των αντιγράφων της εφαρμογής. Συγκεκριμένα η εφαρμογή ανταμείβεται περισσότερο όταν χρησιμοποιεί τους λιγότερους πόρους και λιγότερο όταν χρησιμοποιεί περισσότερους. Συνδυάζοντας λοιπόν αυτές τις τρεις παραμέτρους του latency, throughput και αριθμού αντιγράφων, κατασκευάζουμε μια ανταμοιβή η οποία διδάσκει στον πράκτορα να μεγιστοποιεί την απόδοση ελαχιστοποιώντας την χρήση των πόρων.

Ακολουθεί ένας εποπτικός πίνακας των περιβαλλόντων που αναπτύχθηκαν με το OpenAI Gym.

Όνομα	Μετρικές κατάστασης	Μετρικές απόδοσης (SLA)	Αριθμός καταστάσεων	Αριθμός κινήσεων	Αλγόριθμοι πρακτόρων
k8s-env-discrete-state-discrete-action-V0	CPU, Μνήμη, Pods, Latency	Latency	12.000	9	Τυχαίος
k8s-env-discrete-state-discrete-action-V2	Μνήμη, Pods, Latency	Latency	175	3	Τυχαίος, Q-learning
k8s-env-discrete-state-discrete-action-V3	CPU, Pods, Latency	Latency	175	3	Q-learning
k8s-env-discrete-state-discrete-action-V4	CPU, Pods, Latency	Latency	175	3	Q-learning
k8s-env-discrete-state-discrete-action-V5	CPU÷CPU Threshold, Pods, Latency	Latency	245	3	Q-learning, DynaQ+
k8s-env-discrete-state-discrete-action-V6	CPU, CPU Threshold, Pods, Latency	Latency	1225	3	Q-learning με προσομοίωση, DynaQ+ με προσομοίωση
k8s-env-continuous-state-discrete-action-V0	CPU, CPU threshold, Μνήμη, Threshold μνήμης, Pods, Τερματισμένα pods, HPA error,	Latency, Throughput	N/A	9	Deep Q-learning

	Throughput, Success ratio, Latency				
k8s-env-continuous-state-discrete-action-V1	CPU, CPU threshold, Pods, Τερματισμένα pods, HPA error, Latency	Latency	N/A	3	Deep Q-learning με προσομοίωση

Πίνακας 4.1: Εποπτικός πίνακας περιβαλλόντων ενισχυτικής μάθησης

4.2 Περιβάλλοντα διακριτού χώρου καταστάσεων και διακριτού χώρου κινήσεων

4.2.1 Πραγματικό περιβάλλον

Σε αυτήν την ενότητα δίνεται η περιγραφή του περιβάλλοντος k8s-env-discrete-state-discrete-action-V5 και του περιβάλλοντος k8s-env-discrete-state-discrete-action-V6 ως αντιπροσωπευτικών της κατηγορίας περιβαλλόντων διακριτού χώρου καταστάσεων και κινήσεων. Στο τέλος του κεφαλαίου γίνεται μια συνοπτική αναφορά στα υπόλοιπα περιβάλλοντα της ίδιας κατηγορίας και στις διαφορές μεταξύ τους.

Περιγραφή του περιβάλλοντος

Το περιβάλλον k8s-env-discrete-state-discrete-action-V5 είναι ένα περιβάλλον διακριτού χώρου καταστάσεων και διακριτού χώρου κινήσεων. Αυτό σημαίνει ότι η κατάσταση και οι κινήσεις είναι πεπερασμένες. Το περιβάλλον της εφαρμογής αφορά τη δημιουργία ενός HPA που ορίζει ένα όριο χρησιμοποίησης για την υπολογιστική ισχύ (CPU) ώστε να δημιουργούνται περισσότερα αντίγραφα όταν η χρήση του CPU ξεπερνά τα όρια και να αποδεσμεύονται πόροι όταν η χρήση παραμένει χαμηλή. Το περιβάλλον δίνει τη δυνατότητα αλλαγής αυτού του ορίου (threshold) και σκοπός του πράκτορα είναι η προσαρμογή του threshold ώστε να επιτυγχάνει την καλύτερη ισορροπία μεταξύ υψηλής απόδοσης και χαμηλής χρήσης πόρων.

Συγκεκριμένα οι πληροφορίες που χρησιμοποιήθηκαν για την περιγραφή της τρέχουσας κατάστασης του περιβάλλοντος είναι το πηλίκο της τρέχουσας χρησιμοποίησης του CPU προς το τρέχον threshold για το CPU στον HPA, το ποσοστό των χρησιμοποιούμενων αντιγράφων της εφαρμογής (pods) και το πηλίκο του latency προς το SLA latency που αποτελεί την προδιαγραφή για την απόδοση της εφαρμογής. Όλες οι μεταβλητές αναπαριστούν ένα ποσοστό. Για να παραμείνει μικρός ο χώρος καταστάσεων τα παραπάνω ποσοστά χωρίστηκαν σε διακριτά επίπεδα και κβαντίστηκαν. Τα πιθανά επιπέδια συμπίπτουν με τα εξής διαστήματα αριθμών: [0, 20),

[20, 40), [40, 60), [60,80), [80, 100] συν δύο επιπλέον διαστήματα (100, 150) και [150, +∞) για τις τιμές που μπορούν να ξεπεράσουν το ποσοστό του 100% δηλαδή το πηλίκο της χρησιμοποίησης του CPU προς το threshold και το πηλίκο του latency προς το SLA latency. Συνεπώς το πηλίκο του CPU προς το threshold και το πηλίκο του latency προς το SLA latency μπορούν να πάρουν 7 πιθανές τιμές ενώ το ποσοστό των χρησιμοποιούμενων pods μπορεί να πάρει 5 και άρα έχουμε 7 * 7 * 5 = 245 πιθανές καταστάσεις.

Ο χώρος των πιθανών κινήσεων έχει μέγεθος 3 και οι πιθανές κινήσεις αφορούν τη μεταβολή του threshold. Συγκεκριμένα ορίζουμε τις κινήσεις {0, 1, 2} όπου η κίνηση 0 σημαίνει μείωση του threshold κατά 20%, η κίνηση 1 σημαίνει ότι δε γίνεται καμία αλλαγή και η κίνηση 2 σημαίνει αύξηση του threshold κατά 20%.

Η ανταμοιβή στο συγκεκριμένο περιβάλλον υπολογίζεται με τη συνεισφορά δύο παραμέτρων: του τρέχοντα αριθμού των pods και του τρέχοντος latency.

- Στο κομμάτι των pods η ανταμοιβή είναι μια γραμμική συνάρτηση που δίνει μέγιστη τιμή (100 στην περίπτωση μας) για τον ελάχιστο αριθμό pods (ίσο με 1 pod) και ελάχιστη τιμή (μηδενική ανταμοιβή) για τον μέγιστο αριθμό των pods (ο μέγιστος αριθμός ορίζεται αυθαίρετα από τον χρήστη).
- Στο κομμάτι του latency η ανταμοιβή είναι μέγιστη γύρω από την προδιαγραφή του SLA latency και πέφτει εκθετικά εκατέρωθεν αυτής της τιμής. Η πτώση είναι πιο ραγδαία όταν βρισκόμαστε πάνω από αυτήν την τιμή και πιο ομαλή όταν βρισκόμαστε κάτω. Επίσης, για να επιταχύνουμε την αντίδραση του πράκτορα μεταθέσαμε το μέγιστο σημείο ανταμοιβής στο 80% του SLA latency ώστε η ανταμοιβή να μικραίνει προληπτικά πριν ακόμα έχουμε παράβαση του.

Οι δύο αυτές ανταμοιβές συνεισφέρουν με διαφορετικά βάρη όπου μεγαλύτερο βάρος δόθηκε στο latency (0.7) και μικρότερο στα pods (0.3). Τέλος, υπάρχουν δύο επιπλέον έλεγχοι και συγκεκριμένα όταν έχουμε τον ελάχιστο αριθμό pods δηλαδή 1 pod και δεν παραβιάζεται το SLA τότε θεωρούμε ότι είναι το καλύτερο δυνατό σενάριο άρα δίνουμε την μέγιστη ανταμοιβή 100 ενώ όταν έχουμε παράβαση του SLA latency τότε η ανταμοιβή μηδενίζεται.

Ο ψευδοκώδικας για τον υπολογισμό της ανταμοιβής είναι ο εξής

Αλγόριθμος 4.1: Ψευδοκώδικας υπολογισμού ανταμοιβής

```
reward = 0
Max Reward = 100
Min Reward = 0
pod weight = 0.3
latency weight = 0.7
```

```
#smooth operation with low latency
if num of pod = 1 and current_latency <= sla_latency:
  Reward = max
  return Reward
else if current_latency > sla_latency:
  Reward = 0
  return Reward

pod reward = -100 / (max_pod - 1) * num_pods + 100 * max_pod / (max_pod - 1)
reward += pod weight * pod reward

# if current_latency < sla_latency (bad for provider)
if current_latecy / sla_latency < 0.8:
  latency reward = 100 * e^(-0.3 * d * (0.8 - current_latency / sla_latency)^2)
# if current_latency > sla (bad for client and provider)
else if current_latecy / sla_latency > 0.8:
  latency reward = 100 * e^(-10 * d * (0.8 - current_latency / sla_latency)^2)
reward += latency weight * latency reward

# where: d defines the sharpness of reward function
```

Το περιβάλλον k8s-env-discrete-state-discrete-action-V6 είναι μεταγενέστερη έκδοση του περιβάλλοντος V5 στο οποίο οι μεταβλητές κατάστασης είναι το CPU, το CPU threshold στον HPA, ο αριθμός των pods και το latency. Ο χώρος καταστάσεων έχει μέγεθος 1225 και ο χώρος κινήσεων μέγεθος 3.

Περιγραφή του πράκτορα

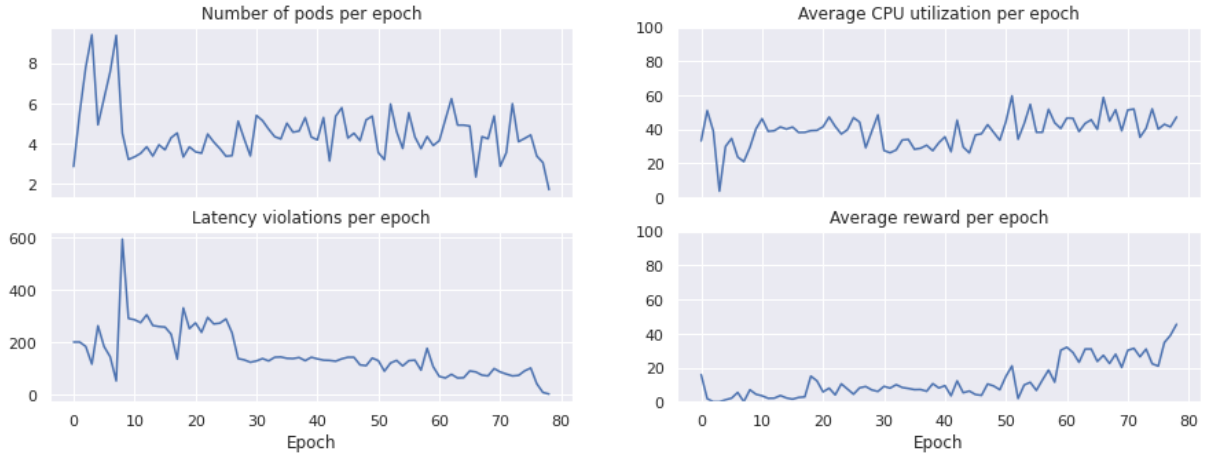
Οι δύο αλγόριθμοι που εφαρμόστηκαν είναι ο Q-learning με πίνακα και ο DynaQ+ επίσης με τη χρήση πίνακα. Η περιγραφή της λειτουργίας των αλγορίθμων βρίσκεται στην ενότητα 3.2.4.2. Ο πράκτορας αλληλεπιδρά με το περιβάλλον μέσω των βασικών συναρτήσεων που ορίζει το Gym και με αυτόν τον τρόπο μπορεί να επαναφέρει το περιβάλλον στην αρχική του κατάσταση και να εφαρμόζει κινήσεις.

Για να εξισορροπηθεί η εξερεύνηση με την εκμετάλλευση εφαρμόστηκε μια πολιτική ε-greedy. Δηλαδή με πιθανότητα ε επιλέγεται μια τυχαία τιμή και με πιθανότητα 1-ε επιλέγεται η μέγιστη τιμή Q. Η τιμή του ε ξεκινά από 1 και μειώνεται σταδιακά μέχρι να πάρει την τιμή 0.2 οπότε και μένει σταθερό. Με αυτόν τον τρόπο ο πράκτορας αρχικά παίρνει κινήσεις που βοηθούν την εξερεύνηση νέων καταστάσεων και κινήσεων και όσο βελτιώνει την πολιτική με τις Q τιμές, η εξερεύνηση μειώνεται και αντικαθίσταται από την εκμετάλλευση της ήδη κατασκευασμένης γνώσης. Επίσης, ο πίνακας Q αρχικοποιείται με την τιμή 50.

Η εκπαίδευση χωρίζεται σε epochs όπου το κάθε epoch χωρίζεται σε 16 βήματα. Ο φόρτος στα χρονικά βήματα ακολουθεί το μοτίβο λ → λ → 3λ → 3λ → 5λ → 5λ → 7λ → 7λ → 9λ →9λ → 7λ → 7λ → 5λ → 5λ → 3λ → 3λ όπου λ ο βασικός ρυθμός αιτημάτων ανά δευτερόλεπτο.

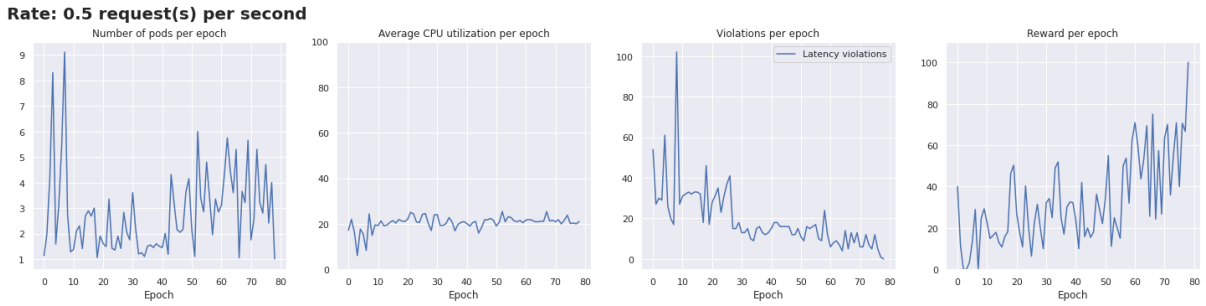
Περιγραφή αποτελεσμάτων

Τα αποτελέσματα για τον Q-learning αλγόριθμο στο περιβάλλον V5 παρουσιάζονται στην Εικόνα 4.1:

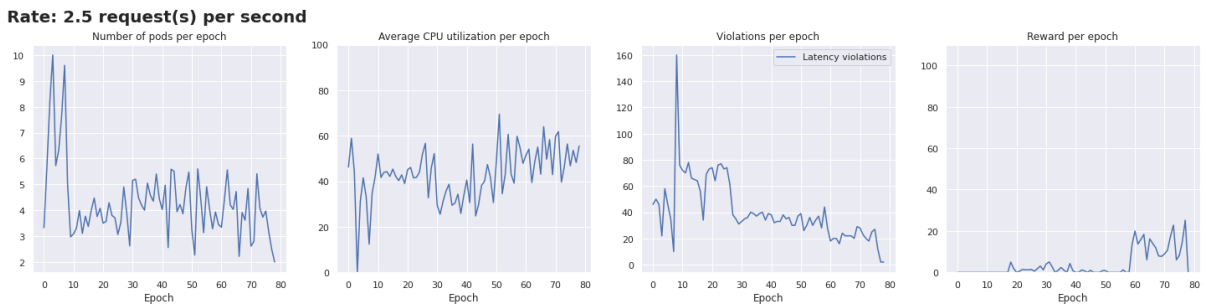


Εικόνα 4.1: Αποτελέσματα Q-learning

Επίσης παρατίθενται ξεχωριστές γραφικές για ένα χαμηλό και ένα υψηλό σημείο του φόρτου για τον Q-learning:



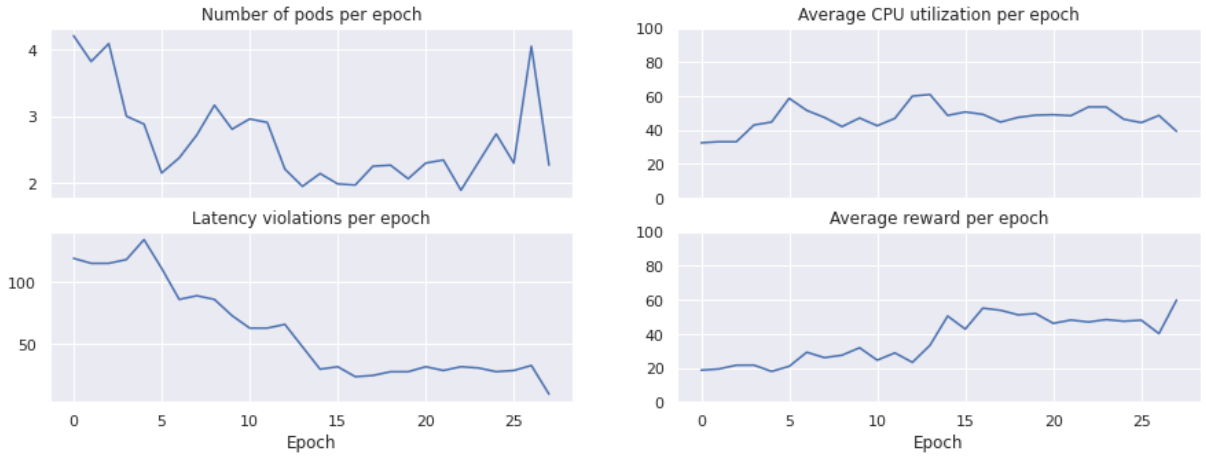
Εικόνα 4.2: Αποτελέσματα χαμηλού φόρτου στον Q-learning



Εικόνα 4.3: Αποτελέσματα υψηλού φόρτου στον Q-learning

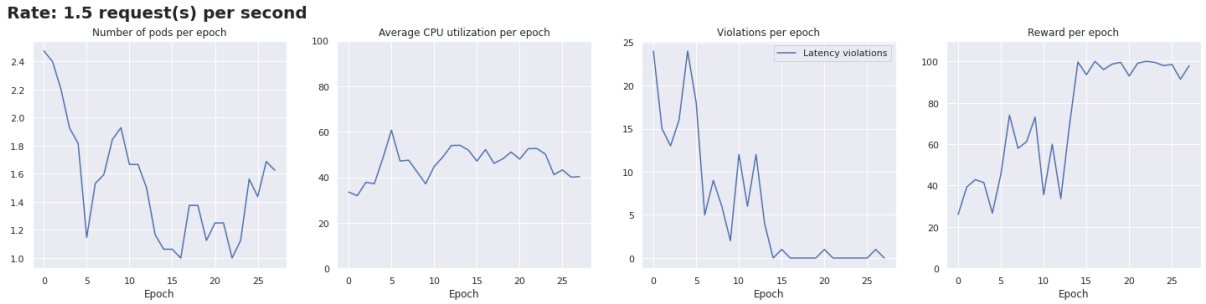
Παρατηρούμε ότι ο αλγόριθμος μαθαίνει να μειώνει τον αριθμό των pods άρα χρησιμοποιεί λιγότερους πόρους. Επίσης, η μέση ανταμοιβή σταδιακά αυξάνεται και οι παραβιάσεις του SLA παρουσιάζουν σταδιακή μείωση.

Τα αποτελέσματα για τον DynaQ+ αλγόριθμο στο περιβάλλον V6 με εκτέλεση για λιγότερα epoch παρουσιάζονται στην Εικόνα 4.4:

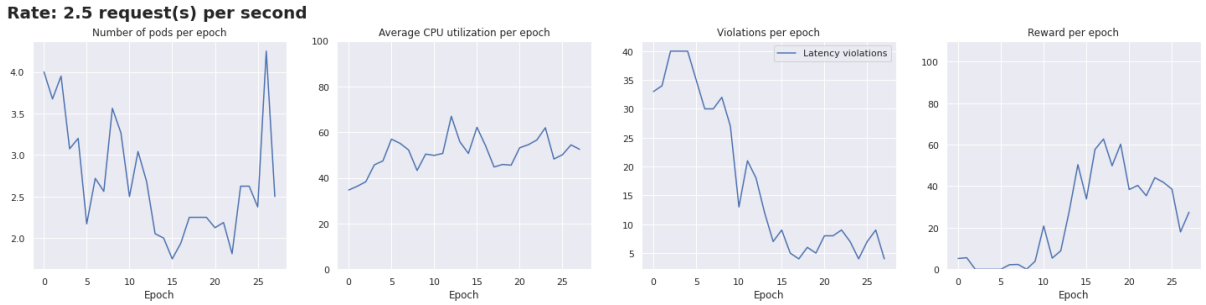


Εικόνα 4.4: Αποτελέσματα DynaQ+

Επίσης παρατίθενται ξεχωριστές γραφικές για το χαμηλό και υψηλό σημείο του φόρτου για τον DynaQ+:



Εικόνα 4.5: Αποτελέσματα χαμηλού φόρτου στον DynaQ+



Εικόνα 4.6: Αποτελέσματα χαμηλού φόρτου στον DynaQ+

Παρουσιάζεται μια τάση μείωσης των πόρων και των παραβιάσεων με σταδιακή αύξηση της ανταμοιβής.

Συνοψίζοντας, η μάθηση στα συγκεκριμένα περιβάλλοντα με τα συγκεκριμένα τεστ και με την εφαρμογή των παραπάνω αλγορίθμων παρουσιάζει μια σχετική μάθηση και ενδιαφέροντα αποτελέσματα. Με την βοήθεια της διαδικασίας προσομοίωσης που περιγράφεται στην επόμενη υποενότητα θα μπορέσουμε να δούμε την εξέλιξη της μάθησης χωρίς να έχουμε τους χρονικούς περιορισμούς που μας επιβάλλει η σταθεροποίηση των μετρικών στην πραγματική υποδομή.

Παραλλαγές του περιβάλλοντος

- Το περιβάλλον k8s-env-discrete-state-discrete-action-V4 αποτελεί παλαιότερη έκδοση του περιβάλλοντος V5 που όπως φαίνεται στον πίνακα 4.1 και στην κατάσταση χρησιμοποιεί τις μεταβλητές CPU, Pods και Latency επομένως έχει λιγότερες πιθανές καταστάσεις αλλά χάνει πληροφορία μέσω αυτής της κωδικοποίησης.
- Το περιβάλλον k8s-env-discrete-state-discrete-action-V3 αποτελεί παλαιότερη έκδοση του περιβάλλοντος V4 με τη διαφορά ότι τα βήματα σε κάθε κίνηση στο V3 είναι ανά 10%.
- Το περιβάλλον k8s-env-discrete-state-discrete-action-V2 αποτελεί παλαιότερη έκδοση του περιβάλλοντος V3 με τη διαφορά ότι η βασική μετρική που χρησιμοποιείται στον HPA είναι η μνήμη και οι αντίστοιχες κινήσεις αλλάζουν τα όρια της.
- Το περιβάλλον k8s-env-discrete-state-discrete-action-V0 αποτελεί την αρχική έκδοση των διακριτών περιβαλλόντων όπου γίνεται χρήση και του CPU και της μνήμης τόσο στην κατάσταση όσο και στις κινήσεις. Οι κινήσεις και τα διακριτά διαστήματα στα οποία κβαντίζονται οι μεταβλητές κατάστασης έχουν βήμα 10% αντί για 20%. Σαν αποτέλεσμα ο χώρος καταστάσεων είναι πολύ μεγάλος (12.000) αλλά επιτρέπει μια πιο λεπτομερή απεικόνιση της κατάστασης.

4.2.2 Προσομοίωση περιβάλλοντος

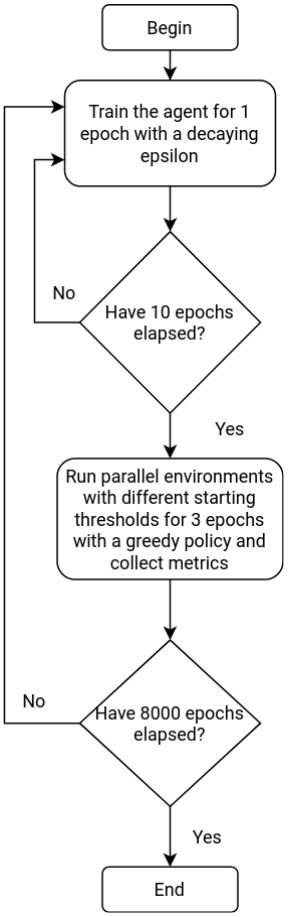
Περιγραφή του περιβάλλοντος

Στο περιβάλλον k8s-env-discrete-state-discrete-action-V6 δοκιμάστηκε η προσομοίωση των επιστρεφόμενων τιμών του περιβάλλοντος ώστε να αποφευχθεί η χρονοβόρα αλληλεπίδραση του πράκτορα με το περιβάλλον και ο χρόνος αναμονής για εφαρμογή της κίνησης. Για την προσομοίωση των τιμών του περιβάλλοντος χρησιμοποιήθηκαν τιμές από προηγούμενα τεστ. Συγκεκριμένα παίρνοντας ιστορικές τιμές που ήταν αποθηκευμένες σε αρχείο υπολογίστηκε η συνάρτηση δειγματικής κατανομής του δειγματικού μέσου. Παρόλο που δεν είναι η καλύτερη μέθοδος για προσομοίωση, παρήγαγε ικανοποιητικά αποτελέσματα. Για την κατασκευή των τιμών λοιπόν συλλέχθηκαν επαναληπτικά δείγματα και υπολογίστηκε ο μέσος όρος τους. Στη

συνέχεια υπολογίστηκε ο μέσος όρος και η τυπική απόκλιση αυτών των τιμών. Το περιβάλλον κατόπιν με αυτές τις τιμές είναι ικανό να παράγει τυχαίες τιμές κανονικής κατανομής με τον υπολογισμένο μέσο όρο και κανονική κατανομή. Οι μετρικές που προσομοιώθηκαν είναι ο αριθμός των αντιγράφων, η χρήση του CPU και το latency. Με αυτόν τον τρόπο ο χρόνος εφαρμογής των κινήσεων καθίσταται αμελητέος και η εκπαίδευση επιταχύνεται αρκετά.

Περιγραφή του πράκτορα

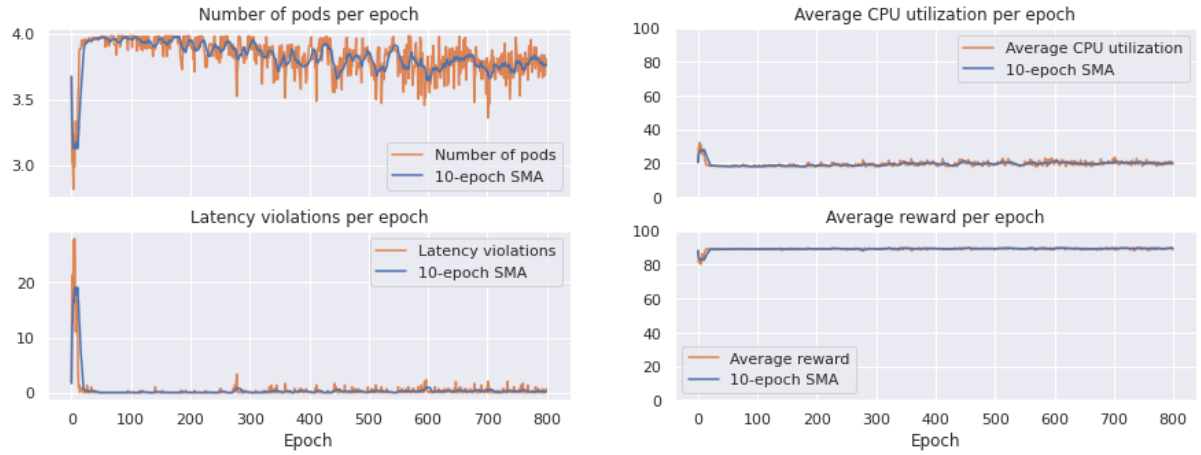
Στο περιβάλλον με την προσομοίωση εφαρμόστηκαν οι αλγόριθμοι Q-learning και DynaQ+. Η πολιτική που εφαρμόζεται είναι ε-greedy με σταδιακή μείωση του ε. Το ε ξεκινά από μια υψηλή τιμή (0.97) που ευνοεί την εξερεύνηση και μετά από 10000 βήματα φτάνει την κατώτατη τιμή (0.2). Και στους δύο αλγόριθμους η εκπαίδευση διαρκεί 8000 epochs όπου κάθε epoch αποτελείται από 16 βήματα με τον φόρτο να ακολουθεί το μοτίβο λ → λ → 3λ → 3λ → 5λ → 5λ → 7λ → 7λ → 9λ → 9λ → 7λ → 7λ → 5λ→ 5λ → 3λ → 3λ όπου λ ο βασικός ρυθμός αιτημάτων. Κάθε 10 epochs εκτελούμε ένα evaluation όπου χρησιμοποιούνται οι καλύτερες τιμές (greedy πολιτική) για να αξιολογήσουμε την απόδοση του αλγόριθμου. Κατά την αξιολόγηση αρχικοποιούμε 5 αντίγραφα της εφαρμογής με καθένα από τα πιθανά CPU thresholds στον HPA αντίστοιχα {20, 40, 60, 80, 100} και η αξιολόγηση διαρκεί 3 epochs.



Εικόνα 4.7: Διάγραμμα ροής εκτέλεσης αλγορίθμου

Περιγραφή αποτελεσμάτων

Τα αποτελέσματα για τον Q-learning αλγόριθμο ως ο μέσος όρος τριών διαφορετικών πειραμάτων παρουσιάζονται στην Εικόνα 4.8:



Εικόνα 4.8: Αποτελέσματα προσομοίωσης με Q-learning

Παρατηρούμε ότι υπάρχει μάθηση και ο αλγόριθμος αυξάνει την ανταμοιβή και μειώνει τις παραβιάσεις του SLA. Οι τιμές σταθεροποιούνται μετά από περίπου 80 evaluation epochs τα οποία αντιστοιχούν σε 800 training epochs εφόσον τρέχουμε μία αξιολόγηση ανά 10 βήματα εκπαίδευσης. Επίσης, ο αριθμός των pods παρουσιάζει πτωτική τάση που σημαίνει χαμηλότερη χρησιμοποίηση πόρων και άρα λιγότερο κόστος.

Για την καλύτερη ανάλυση των αποτελεσμάτων και την ανάδειξη της αρχικής μάθησης παρατίθεται ένα διάγραμμα που αφορά μόνο την εκπαίδευση χωρίς ξεχωριστή φάση αξιολόγησης για τα πρώτα 1000 epochs εκπαίδευσης:

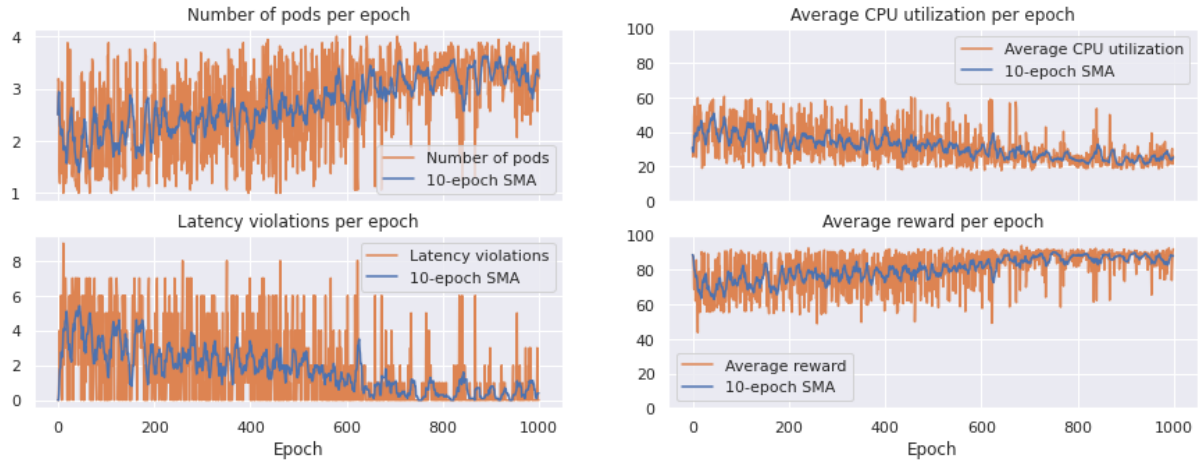
Εικόνα 4.9: Αποτελέσματα προσομοίωσης με Q-learning
μόνο με φάση εκπαίδευσης

Τα αποτελέσματα για τον DynaQ+ αλγόριθμο ως ο μέσος όρος τριών διαφορετικών πειραμάτων παρουσιάζονται στην Εικόνα 4.10:

Εικόνα 4.10: Αποτελέσματα προσομοίωσης με DynaQ+

Εδώ παρατηρείται η ίδια τάση μείωσης των παραβιάσεων του SLA και αύξησης της ανταμοιβής αλλά οι τιμές δεν είναι τόσο σταθερές και παρουσιάζουν μεγάλη διακύμανση ειδικά όσον αφορά την ανταμοιβή.

Παρόμοια παρατίθεται ένα διάγραμμα χωρίς φάση αξιολόγησης για τα πρώτα 1000 epochs εκπαίδευσης:



Εικόνα 4.11: Αποτελέσματα προσομοίωσης με DynaQ+
μόνο με φάση εκπαίδευσης

4.3 Περιβάλλοντα συνεχή χώρου καταστάσεων και διακριτού χώρου κινήσεων

4.3.1 Πραγματικό περιβάλλον

Περιγραφή του περιβάλλοντος

Το περιβάλλον k8s-env-continuous-state-discrete-action-V0 είναι ένα περιβάλλον συνεχή χώρου καταστάσεων και διακριτού χώρου κινήσεων. Επομένως, το βασικό πλεονέκτημα του έναντι των διακριτών περιβαλλόντων είναι το γεγονός ότι η πληροφορία για την κατάσταση του περιβάλλοντος διατηρείται ακέραια χωρίς να υποστεί μετατροπή σε διακριτά επίπεδα. Αυτό θεωρητικά επιτρέπει την καλύτερη προσαρμογή του πράκτορα σε αλλαγές στην κατάσταση. Σε αυτό το περιβάλλον δημιουργούμε έναν HPA ο οποίος κλιμακώνει την εφαρμογή με βάση τις μετρικές του CPU και της μνήμης. Ορίζονται κάποια όρια χρησιμοποίησης για αυτές τις μετρικές η υπέρβαση των οποίων προκαλεί τη δημιουργία αντιγράφων και αντίστοιχα αποδέσμευση τους όταν η χρησιμοποίηση είναι πολύ χαμηλή.

Για την περιγραφή της κατάστασης του περιβάλλοντος χρησιμοποιήθηκαν οι μετρικές του CPU, του CPU threshold, της μνήμης, του threshold μνήμης, του αριθμού των pods, του αριθμού των τερματισμένων pods, του HPA error, του throughput, του ποσοστού επιτυχίας των αιτημάτων και του latency.

Ο χώρος των πιθανών κινήσεων έχει μέγεθος 9 και οι πιθανές κινήσεις αφορούν τη μεταβολή των δύο threshold για το CPU και τη μνήμη. Συγκεκριμένα ορίζουμε τις κινήσεις {0, 1, 2} όπου η κίνηση 0 σημαίνει μείωση του threshold κατά 20%, η κίνηση 1 σημαίνει ότι δε γίνεται καμία αλλαγή και η κίνηση 2 σημαίνει αύξηση του threshold κατά 20%. Η κίνηση λοιπόν αποτελείται από δύο αριθμούς [α, β] όπου α και β είναι αριθμοί στο σύνολο {0, 1, 2} και το α συμβολίζει την κίνηση για το CPU threshold και το β την κίνηση για το threshold μνήμης.

Για τον υπολογισμό της ανταμοιβής συνυπολογίζονται τρεις συνιστώσες: ο αριθμός των pods, το latency και το throughput.

- Στο κομμάτι των pods η ανταμοιβή είναι μια γραμμική συνάρτηση που δίνει μέγιστη τιμή (100 στην περίπτωση μας) για τον ελάχιστο αριθμό pods (ίσο με 1 pod) και ελάχιστη τιμή (μηδενική ανταμοιβή) για τον μέγιστο αριθμό των pods (ο μέγιστος αριθμός ορίζεται αυθαίρετα από τον χρήστη).
- Στο κομμάτι του latency η ανταμοιβή είναι μέγιστη γύρω από την προδιαγραφή του SLA latency και πέφτει εκθετικά εκατέρωθεν αυτής της τιμής. Η πτώση είναι πιο ραγδαία όταν βρισκόμαστε πάνω από αυτήν την τιμή και πιο ομαλή όταν βρισκόμαστε κάτω. Επίσης, για να επιταχύνουμε την αντίδραση του πράκτορα μεταθέσαμε το μέγιστο σημείο ανταμοιβής στο 80% του SLA latency ώστε η ανταμοιβή να μικραίνει προληπτικά πριν ακόμα έχουμε παράβαση του.

- Στο κομμάτι του throughput ελέγχονται δύο παράμετροι: το ποσοστό επιτυχίας των αιτημάτων και ο ρυθμός επιτυχημένων αιτημάτων ανά δευτερόλεπτο. Η ανταμοιβή πέφτει όταν ξεπεραστεί το διπλάσιο του SLA throughput ή όταν το ποσοστό επιτυχίας πέσει κάτω από το 95%.

Οι τρεις αυτές ανταμοιβές συνεισφέρουν με διαφορετικά βάρη όπου μεγαλύτερο βάρος δόθηκε στο throughput (0.4) και στο latency (0.4) και μικρότερο στα pods (0.2). Τέλος, υπάρχουν τρεις επιπλέον έλεγχοι και συγκεκριμένα όταν έχουμε τον ελάχιστο αριθμό pods δηλαδή 1 pod, δεν παραβιάζεται το SLA για το latency και το throughput και το ποσοστό επιτυχίας είναι μεγαλύτερο του 95% τότε θεωρούμε ότι είναι το καλύτερο δυνατό σενάριο άρα δίνουμε την μέγιστη ανταμοιβή 100 ενώ όταν έχουμε παράβαση του SLA latency τότε η ανταμοιβή μηδενίζεται.

Ο ψευδοκώδικας για τον υπολογισμό της ανταμοιβής είναι ο εξής

Αλγόριθμος 4.2: Ψευδοκώδικας υπολογισμού ανταμοιβής

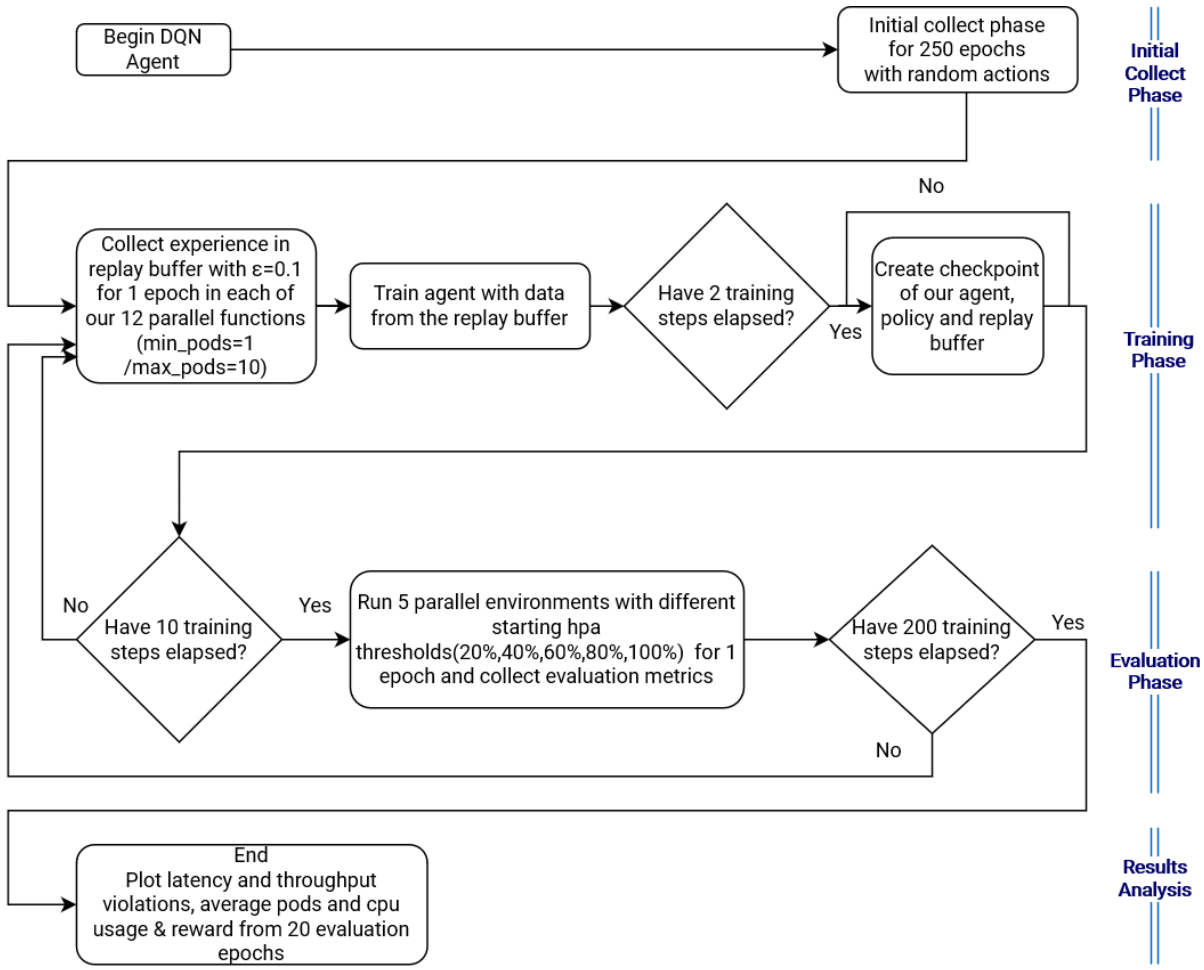
```
reward = 0
Max Reward = 100
Min Reward = 0
pod weight = 0.2
latency weight = 0.4
throughput_weight = 0.4

#smooth operation with low latency and high success ratio
if num of pod = 1 and current_latency <= sla_latency
    and throughput <= 2 * sla_throughput and success_ratio >= 0.95:
  Reward = max
  return Reward
else if current_latency > sla_latency:
  Reward = 0
  return Reward

pod reward = -100 / (max_pod - 1) * num_pods + 100 * max_pod / (max_pod - 1)
reward += pod weight * pod reward

# if current_latency < sla_latency (bad for provider)
if current_latecy / sla_latency < 0.8:
  latency reward = 100 * e^(-0.3 * d * (0.8 - current_latency / sla_latency)^2)
# if current_latency > sla (bad for client and provider)
else if current_latecy / sla_latency > 0.8:
  latency reward = 100 * e^(-10 * d * (0.8 - current_latency / sla_latency)^2)
reward += latency weight * latency reward

if success_ratio >= 0.95 and throughput < 2 * sla_throughput:
  throughput_reward = 100 * e^(-0.05 * d * (2 - throughput / sla_throughput)^2)
else:
```

```
  throughput_reward = 100 * e^(-10 * d * (2 - throughput / sla_throughput)^2)

reward += throughput_weight * throughput_reward

# where: d defines the sharpness of reward function
```

Περιγραφή του πράκτορα

Ο πράκτορας χρησιμοποιεί τον αλγόριθμο του Deep Q-learning που αποτελεί υλοποίηση του Q-learning αλγόριθμου όπου για τις τιμές Q χρησιμοποιείται ένα νευρωνικό δίκτυο. Η λειτουργία του αλγόριθμου περιγράφεται στην ενότητα 3.2.4.3. Όπως εξηγείται σε εκείνη την ενότητα, η εκπαίδευση του νευρωνικού δικτύου γίνεται με δεδομένα που μπαίνουν σε έναν replay buffer.

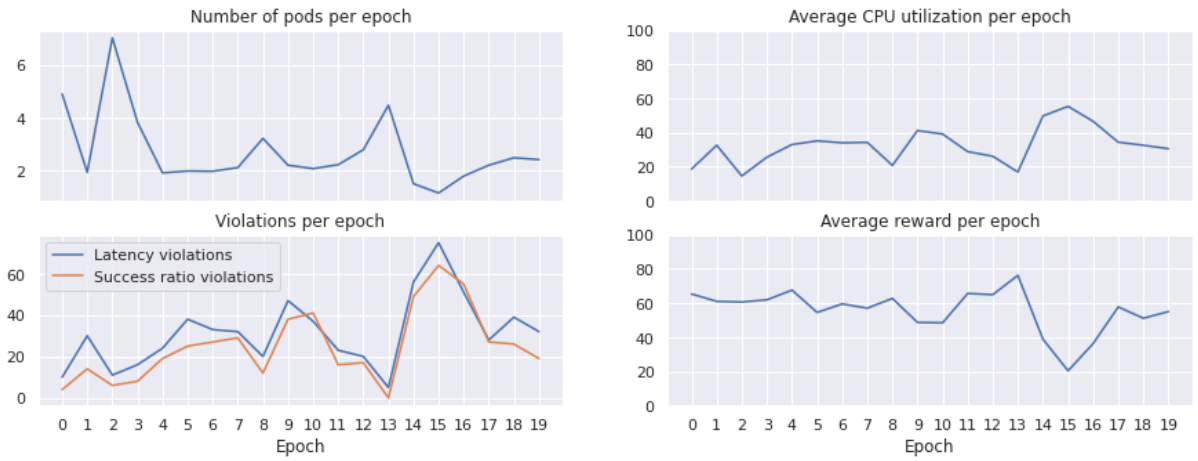
Στην αρχική φάση του αλγόριθμου πριν την εκπαίδευση συλλέγουμε δεδομένα με τυχαίο τρόπο για να αποκτήσει τυχαίες τιμές ο buffer και στη συνέχεια όταν ξεκινήσει η εκπαίδευση συλλέγουμε δεδομένα με πολιτική ε-greedy και ε ίσο με 0.1. Σταδιακά τα τυχαία δεδομένα γίνονται λιγότερα από τα δεδομένα με τις βέλτιστες κινήσεις με αποτέλεσμα να έχουμε εξερεύνηση στην αρχή και εκμετάλλευση αργότερα. Η εκπαίδευση διαρκεί 200 epochs/βήματα εκπαίδευσης. Συγκεκριμένα έχουμε παράλληλες υλοποιήσεις των συναρτήσεων μας και αφότου συλλεχθούν δεδομένα ενός epoch για καθεμία από αυτές, τα δεδομένα προστίθενται στον buffer και στη συνέχεια εκπαιδεύεται το νευρωνικό με μια δέσμη (batch) δεδομένων. Κάθε epoch αποτελείται από 4 βήματα με τον φόρτο να ακολουθεί το μοτίβο $\lambda \rightarrow 7\lambda \rightarrow 7\lambda \rightarrow \lambda$ όπου λ ο βασικός ρυθμός αιτημάτων. Όταν παρέρχονται 10 βήματα εκπαίδευσης, αρχικοποιούμε 5 διαφορετικές συναρτήσεις όπου στην καθεμία δημιουργείται ένας HPA με όλα τα πιθανά CPU thresholds {20, 40, 60, 80, 100}. Για καθεμία από αυτές αξιολογούμε τις αποφάσεις τους για τη διάρκεια ενός epoch αν εφαρμοστεί greedy πολιτική δηλαδή με πλήρη εκμετάλλευση των τιμών που έχει μάθε ο αλγόριθμος μέχρι στιγμής. Η ροή του αλγόριθμου απεικονίζεται στο διάγραμμα ροής της εικόνας 4.12.



Εικόνα 4.12: Διάγραμμα ροής εκτέλεσης αλγορίθμου

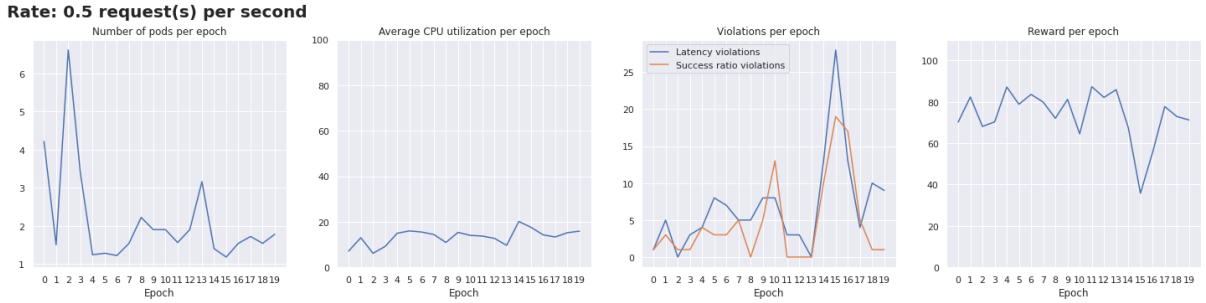
Περιγραφή αποτελεσμάτων

Τα αποτελέσματα για τον Deep Q-learning αλγόριθμο παρουσιάζονται στην Εικόνα 4.13:

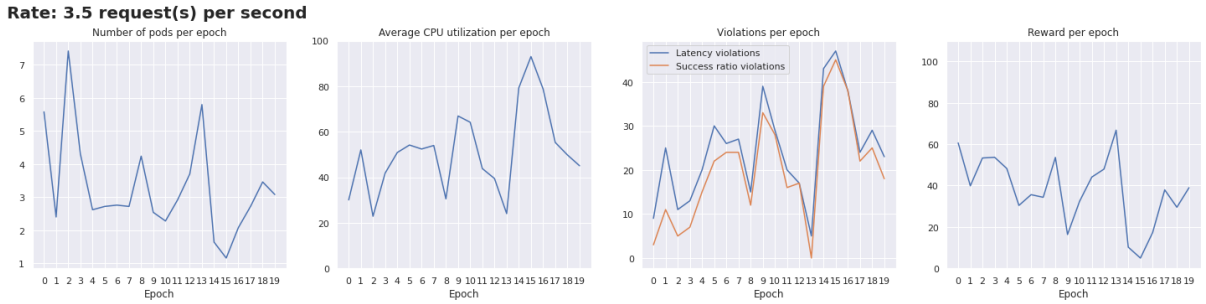


Εικόνα 4.13: Αποτελέσματα Deep Q-learning

Επίσης παρατίθενται δύο διαγράμματα με τη χαμηλή και την υψηλή φάση φόρτου της εκπαίδευσης:



Εικόνα 4.14: Αποτελέσματα χαμηλού φόρτου



Εικόνα 4.15: Αποτελέσματα υψηλού φόρτου

Παρατηρούμε ότι υπάρχει μάθηση και ειδικά μέχρι το 13ο epoch αξιολόγησης υπάρχει βελτίωση στις μετρικές με αύξηση της ανταμοιβής και μείωση των παραβιάσεων των SLA όμως στη συνέχεια οι παραβιάσεις αυξάνονται και η ανταμοιβή πέφτει. Αυτό ίσως οφείλεται στο γεγονός ότι ο χρόνος εκπαίδευσης δεν είναι αρκετός και το νευρωνικό δίκτυο δεν έχει ενημερώσει τα βάρη του κατάλληλα ώστε να παίρνει καλές αποφάσεις.

4.3.2 Προσομοίωση περιβάλλοντος

Περιγραφή του περιβάλλοντος

Το περιβάλλον k8s-env-continuous-state-discrete-action-V1 αποτελεί απλοποιημένη μορφή του περιβάλλοντος k8s-env-continuous-state-discrete-action-V0 όπου ως μεταβλητή κατάστασης χρησιμοποιείται μόνο το CPU και ως μετρική απόδοσης μόνο το latency. Ο υπολογισμός της ανταμοιβής γίνεται όπως και στα περιβάλλοντα διακριτού χώρου καταστάσεων V5 και V6. Στο συγκεκριμένο περιβάλλον εφαρμόστηκε προσομοίωση όπως περιγράφεται στην ενότητα 4.2.2.1.

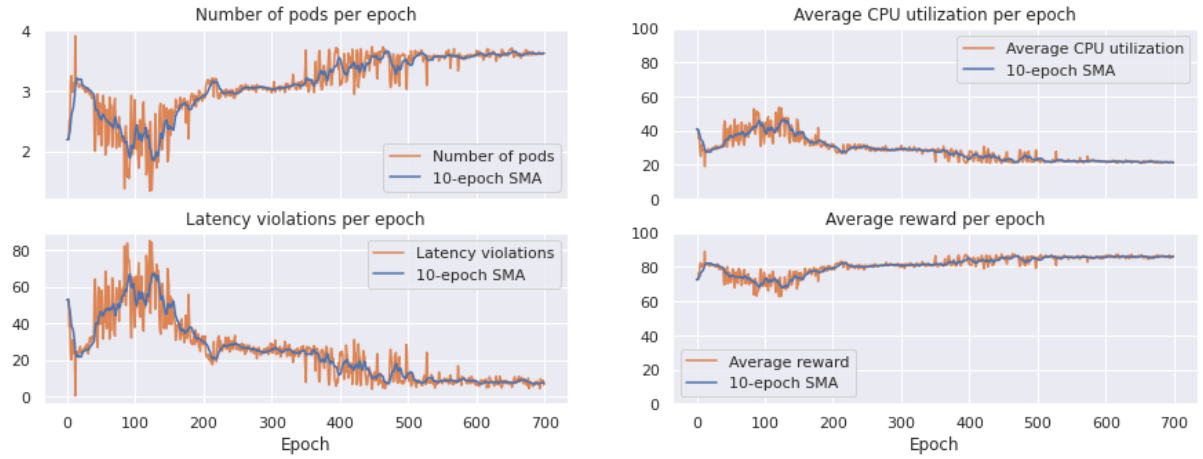
Περιγραφή του πράκτορα

Η εκπαίδευση έγινε με δύο διαφορετικά μοτίβα φόρτου και συγκεκριμένα τα $\lambda \rightarrow 7\lambda \rightarrow 7\lambda \rightarrow \lambda$ και $\lambda \rightarrow \lambda \rightarrow 3\lambda \rightarrow 3\lambda \rightarrow 5\lambda \rightarrow 5\lambda \rightarrow 7\lambda \rightarrow 7\lambda \rightarrow 9\lambda \rightarrow 9\lambda \rightarrow 7\lambda \rightarrow 7\lambda \rightarrow 5\lambda \rightarrow 5\lambda \rightarrow 3\lambda \rightarrow 3\lambda$ όπου λ ο βασικός ρυθμός αιτημάτων. Η ροή είναι παρόμοια με αυτήν που

απεικονίζεται στην εικόνα 4.12 με τη διαφορά ότι έχουμε συλλογή αρχικών δεδομένων για 50 epochs και εκπαίδευση για 7000 epochs στο μοτίβο με τα 4 χρονικά βήματα και 8000 epochs στο μοτίβο με τα 16 χρονικά βήματα. Οι μετρήσεις εκτελέστηκαν τρεις διαφορετικές φορές ώστε να εξομαλυνθούν οι τυχαιότητες κατά την εκπαίδευση.

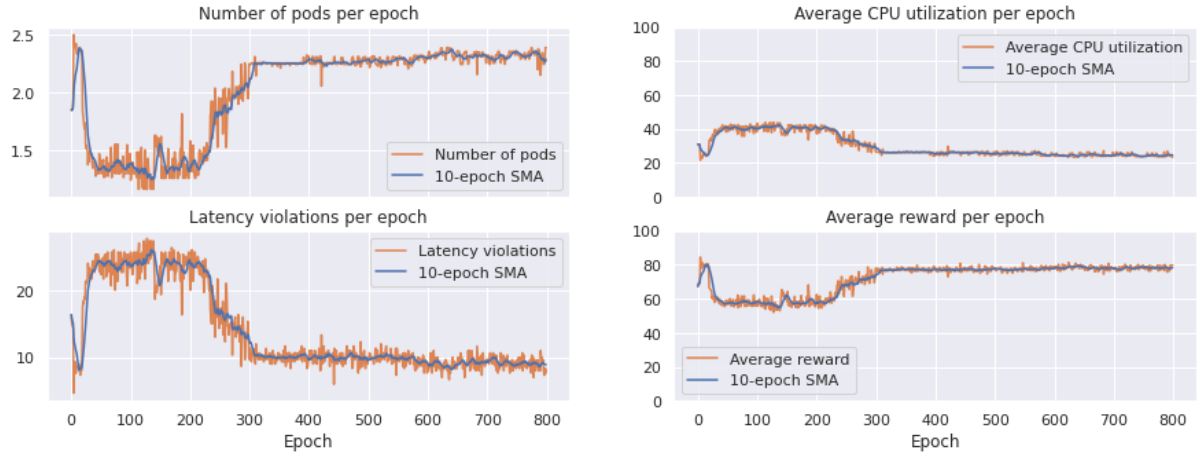
Περιγραφή αποτελεσμάτων

Τα αποτελέσματα για τον Deep Q-learning αλγόριθμο ως ο μέσος όρος τριών διαφορετικών εκτελέσεων για το μοτίβο των 4 χρονικών βημάτων μαζί με τον κινητό μέσο όρο για 10 epochs παρουσιάζονται στην Εικόνα 4.16:



Εικόνα 4.16: Αποτελέσματα Deep Q-learning για το μοτίβο των 4 χρονικών βημάτων

Τα αποτελέσματα για τον Deep Q-learning αλγόριθμο ως ο μέσος όρος τριών διαφορετικών εκτελέσεων για το μοτίβο των 16 χρονικών βημάτων μαζί με τον κινητό μέσο όρο για 10 epochs παρουσιάζονται στην Εικόνα 4.17:



Εικόνα 4.17: Αποτελέσματα Deep Q-learning για το μοτίβο των 16 χρονικών βημάτων

Παρατηρούμε ότι υπάρχει μάθηση με αύξηση της ανταμοιβής και ελαχιστοποίηση των παραβιάσεων. Η εκπαίδευση χρειάζεται περισσότερα epochs για το πιο περίπλοκο μοτίβο φόρτου με τα 16 χρονικά βήματα.

4.4 Σύνοψη αποτελεσμάτων

Σύγκριση αλγορίθμων ενισχυτικής μάθησης

Συνολικά την καλύτερη επίδοση έδειξε ο αλγόριθμος Q-learning στο διακριτό περιβάλλον και συγκεκριμένα στην προσομοίωση. Οι τιμές σταθεροποιούνται πιο γρήγορα σε σχέση με τους υπόλοιπους αλγόριθμους και οι παραβιάσεις του SLA ελαχιστοποιούνται ενώ η ανταμοιβή σταθεροποιείται σε μια υψηλή τιμή. Ο αλγόριθμος DynaQ+ παρουσιάζει επίσης ικανοποιητική μάθηση με μεγαλύτερη διακύμανση όμως στις τιμές. Στα συνεχή περιβάλλοντα ο αλγόριθμος Deep Q-learning παρουσίασε επίσης θετικά αποτελέσματα με σημαντικά μεγαλύτερο χρόνο εκπαίδευσης που δικαιολογείται από το γεγονός ότι εκπαιδεύεται νευρωνικό δίκτυο. Συγκεκριμένα ο πράκτορας όπως ήταν αναμενόμενο έμαθε πιο γρήγορα στο μοτίβο των 4 χρονικών βημάτων σε σχέση με το μοτίβο των 16 χρονικών βημάτων αφού στη δεύτερη περίπτωση υπάρχει μεγαλύτερη ανάγκη για διαφοροποιημένη εκπαίδευση σε περισσότερους συνδυασμούς φόρτου.

Σύγκριση περιβαλλόντων διακριτού και συνεχούς χώρου καταστάσεων

Το περιβάλλον διακριτού χώρου καταστάσεων V6 και το περιβάλλον συνεχούς χώρου καταστάσεων V1 έχουν αρκετά παρόμοια περιγραφή κατάστασης του περιβάλλοντος με τις βασικές μετρικές να είναι το CPU και το latency. Η βασική διαφορά φυσικά είναι ότι το διακριτό περιβάλλον περιγράφει την κατάσταση μετατρέποντας τις μετρικές που συλλέγονται σε διακριτές τιμές ενώ το συνεχές περιβάλλον κρατά την αρχική πληροφορία ακέραια. Επίσης, στο διακριτό περιβάλλον οι αλγόριθμοι που εφαρμόστηκαν βασίστηκαν στη χρήση πίνακα για την υποστήριξη των αλγορίθμων ενώ στο συνεχές στη χρήση νευρωνικού δικτύου.

Συγκρίνοντας τα αποτελέσματα των προσομοιώσεων παρατηρείται ότι στο διακριτό περιβάλλον η μάθηση συγκλίνει σε λιγότερα από 100 epoch αξιολόγησης στις τελικές τιμές της και η ανταμοιβή παίρνει υψηλή τιμή ενώ οι παραβιάσεις του latency τείνουν στο μηδέν. Αντίθετα στο συνεχές περιβάλλον η μάθηση διαρκεί για περισσότερα από 300 epoch αξιολόγησης στο πιο περίπλοκο μοτίβο φόρτου. Στο συνεχές περιβάλλον παρατηρείται επίσης χαμηλότερη χρήση πόρων εξαρχής ενώ ο πράκτορας στο διακριτό περιβάλλον χρειάζεται αρκετά epoch για να δείξει παρόμοια τάση μείωσης. Παρότι η αναπαράσταση του περιβάλλοντος κλιμακωσιμοτητας με συνεχή τρόπο είναι πιο κοντά στην πραγματικότητα απαιτεί πολύ περισσότερο χρονο

εκπαίδευσης. Γι αυτό τον λόγο η χρήση διακριτών περιβάλλον έκανε εφικτή την παρατήρηση συγκλισης μάθησης σε πραγματικό χρόνο ενώ η ίδια σύγκλιση στο συνεχές περιβάλλον φάνηκε μόνο μέσω της προσομοίωσης.

5. Συμπεράσματα και μελλοντικές επεκτάσεις

Βασικός στόχος της εργασίας υπήρξε η μοντελοποίηση προβλημάτων αυτόματης κλιμάκωσης σε serverless περιβάλλοντα και η δοκιμή της ενισχυτικής μάθησης σε αυτό το πλαίσιο για τη βελτιστοποίηση της απόδοσης και την ελαχιστοποίηση του κόστους. Με τη χρήση του OpenAI Gym δημιουργήθηκαν περιβάλλοντα τα οποία περιγράφουν την κατάσταση και επιστρέφουν την κατάλληλη ανταμοιβή ώστε να είναι εφικτή η ανάπτυξη πρακτόρων ενισχυτικής μάθησης. Περισσότερες πληροφορίες για τον κώδικα αναφέρονται στην ενότητα Ανοικτό Αποθετήριο Κώδικα.

Δοκιμάστηκαν τόσο διακριτές όσο και συνεχείς αναπαραστάσεις του περιβάλλοντος και της κατάστασής του. Παρά την απώλεια πληροφορίας κατά την προσαρμογή της κατάστασης σε διακριτά επίπεδα τα διακριτά περιβάλλοντα έδειξαν επαρκή μάθηση με τη χρήση του Q-learning αλγόριθμου. Αντίστοιχα, στα συνεχή περιβάλλοντα ο αλγόριθμος Deep Q-learning παρουσίασε βελτίωση της απόδοσης αν και ο χρόνος εκπαίδευσης ήταν συγκριτικά μεγαλύτερος λόγω της χρήσης νευρωνικού δικτύου. Επειδή η υποδομή απαιτεί αρκετό χρόνο για την εφαρμογή των κινήσεων, εφαρμόστηκαν προσομοιώσεις που κατέστησαν εφικτή την εκπαίδευση για μεγάλο αριθμό χρονικών βημάτων.

Μελλοντικά θα μπορούσε να γίνει πιο συστηματική η συλλογή δεδομένων από πραγματικά περιβάλλοντα με τη χρήση καλύτερων μεθόδων για τη δημιουργία προσομοιώσεων και την αποφυγή των μεγάλων χρόνων εκπαίδευσης. Επίσης, θα μπορούσαν να εφαρμοστούν και άλλοι αλγόριθμοι με τα περιβάλλοντα συνεχή χώρου καταστάσεων όπως ο Double Q-learning [41] και ο Soft Actor Critic [42]. Τέλος, ο κώδικας που αναπτύχθηκε μπορεί να τροποποιηθεί και να επεκταθεί για τη δημιουργία προσαρμοσμένων περιβαλλόντων ώστε να δουλεύουν και με άλλες serverless πλατφόρμες και να έχουν διαφορετικό ορισμό ανταμοιβής ή περιγραφή χώρου καταστάσεων.

Βιβλιογραφία

[1] Sriram, Ilango & Khajeh-Hosseini, Ali. (2010). *Research Agenda in Cloud Technologies*
URL: http://arxiv.org/abs/1001.3259.

[2] Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra and B. Hu, "Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends," 2015 IEEE 8th International Conference on Cloud Computing, 2015, pp. 621-628, doi:https://doi.org/10.1109/CLOUD.2015.88

[3] Badger, M. , Grance, T. , Patt-Corner, R. and Voas, J. (2012), Cloud Computing Synopsis and Recommendations, Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, [online], doi:https://doi.org/10.6028/NIST.SP.800-146

[4] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2019. *The rise of serverless computing*. Commun. ACM 62, 12 (December 2019), 44–54. DOI:https://doi.org/10.1145/3368454

[5] Serverless Architecture in Short
https://specify.io/concepts/serverless-baas-faas
Ημερομηνία πρόσβασης: 14/6/2021

[6] Sarah Allen and et al. 2018. CNCF Serverless Whitepaper.
https://github.com/cncf/wg-serverless/blob/master/whitepapers/serverless-overview/cncf_serverless_whitepaper_v1.0.pdf.
Ημερομηνία πρόσβασης: 14/6/2021

[7] What is serverless?
https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless
Ημερομηνία πρόσβασης: 1/6/2021

[8] Serverless Computing
https://mjbright.github.io/Talks/2018-Oct-07_Pyconfr_Serverless/#slide=7
Ημερομηνία πρόσβασης: 14/6/2021

[9] Serverless Computing Applications and Architectures
https://www.xenonstack.com/insights/serverless-computing/
Ημερομηνία πρόσβασης: 14/6/2021

[10] What is serverless computing? | Serverless definition
https://www.cloudflare.com/learning/serverless/what-is-serverless/
Ημερομηνία πρόσβασης: 1/6/2021

[11] S. Eismann et al., "Serverless Applications: Why, When, and How?," in IEEE Software, vol. 38, no. 1, pp. 32-39, Jan.-Feb. 2021, doi:https://doi.org/10.1109/MS.2020.3023302

[12] P. Vahidinia, B. Farahani and F. S. Aliee, "Cold Start in Serverless Computing: Current Trends and Mitigation Strategies," 2020 International Conference on Omni-layer Intelligent Systems (COINS), 2020, pp. 1-7, doi:https://doi.org/10.1109/COINS49042.2020.9191377

[13] K. Solaiman and M. A. Adnan, "WLEC: A Not So Cold Architecture to Mitigate Cold Start Problem in Serverless Computing," 2020 IEEE International Conference on Cloud Engineering (IC2E), 2020, pp. 144-153, doi:https://doi.org/10.1109/IC2E48712.2020.00022

[14] Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2018. SOCK: rapid task provisioning with serverless-optimized containers. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference* (*USENIX ATC '18*). USENIX Association, USA, 57–69 URL:https://www.usenix.org/conference/atc18/presentation/oakes.

[15] Sol Boucher, Anuj Kalia, David G. Andersen, & Michael Kaminsky (2018). Putting the "Micro" Back in Microservice. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)* (pp. 645–650). USENIX Association URL:https://www.usenix.org/conference/atc18/presentation/boucher.

[16] Kubernetes (K8s)
https://github.com/kubernetes/kubernetes/
Ημερομηνία πρόσβασης: 4/6/2021

[17] Kubernetes Components
https://kubernetes.io/docs/concepts/overview/components/
Ημερομηνία πρόσβασης: 4/6/2021

[18] Horizontal Pod Autoscaler
https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
Ημερομηνία πρόσβασης: 4/6/2021

[19] Knative
https://knative.dev
Ημερομηνία πρόσβασης: 4/6/2021

[20] Apache OpenWhisk
https://openwhisk.apache.org/
Ημερομηνία πρόσβασης: 4/6/2021

[21] OpenFaaS
https://www.openfaas.com/
Ημερομηνία πρόσβασης: 6/6/2021

[22] Fission
https://fission.io/
Ημερομηνία πρόσβασης: 6/6/2021

[23] Kubeless
https://kubeless.io/
Ημερομηνία πρόσβασης: 6/6/2021

[24] N. Mahmoudi and H. Khazaei, "Performance Modeling of Serverless Computing Platforms," in *IEEE Transactions on Cloud Computing*, doi:https://doi.org/10.1109/TCC.2020.3033373.

[25] L. Schuler, S. Jamil, N. Kühl, Ai-based resource allocation: Reinforcement learning for adaptive auto-scaling in serverless environments, CoRR abs/2005.14410 (2020). arXiv:2005.14410.
URL:https://arxiv.org/abs/2005.14410

[26] P. Gouvas, E. Fotopoulou, A. Zafeiropoulos, C. Vassilakis, A Context Model and Policies Management Frame-work for Reconfigurable-by-design Distributed Applications, Procedia Computer Science 97 (2016) 122–125.
doi:https://doi.org/10.1016/j.procs.2016.08.288.
URL:https://www.sciencedirect.com/science/article/pii/S1877050916321044

[27] Y. Garí, D. A. Monge, E. Pacini, C. Mateos, C. G. Garino, Reinforcement learning-based application Autoscaling in the Cloud:A survey, Engineering Applications of Artificial Intelligence 102 (2021) 104288.
doi:https://doi.org/10.1016/j.engappai.2021.104288.

[28] M. Imdoukh, I. Ahmad, M. G. Alfailakawi, Machine learning-based auto-scaling for containerized applications, Neural Computing and Applications 32 (13) (2020) 9745–9760. doi:https://doi.org/10.1007/s00521-019-04507-z.

[29] C. Lin, H. Khazaei, Modeling and optimization of performance and cost of serverless applications, IEEE Transactions on Parallel and Distributed Systems 32 (3) (2021) 615–632. doi:https://10.1109/TPDS.2020.3028841.

[30] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

[31] Mnih, V., Kavukcuoglu, K., Silver, D. *et al.* Human-level control through deep reinforcement learning. *Nature* **518,** 529–533 (2015). doi:https://doi.org/10.1038/nature14236

[32] S. Kardani-Moghaddam, R. Buyya, K. Ramamohanarao, 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. doi:https://doi.org/10.1109/TPDS.2020.3025914.

[33] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, & Wojciech Zaremba. (2016). OpenAI Gym, URL:http://arxiv.org/abs/1606.01540

[34] OpenAI Gym
https://gym.openai.com/
Ημερομηνία πρόσβασης: 6/6/2021

[35] Tensorflow
https://www.tensorflow.org/
Ημερομηνία πρόσβασης: 6/6/2021

[36] Tensorflow Agents
https://www.tensorflow.org/agents
Ημερομηνία πρόσβασης: 7/6/2021

[37] Metrics server
https://github.com/kubernetes-sigs/metrics-server
Ημερομηνία πρόσβασης: 14/6/2021

[38] NGINX Ingress Controller
https://kubernetes.github.io/ingress-nginx/
Ημερομηνία πρόσβασης: 14/6/2021

[39] Prometheus
https://prometheus.io/
Ημερομηνία πρόσβασης: 14/6/2021

[40] Vegeta
https://github.com/tsenart/vegeta
Ημερομηνία πρόσβασης: 10/6/2021

[41] Hado van Hasselt, Arthur Guez, & David Silver. (2015). Deep Reinforcement Learning with Double Q-learning.
URL:https://arxiv.org/abs/1509.06461

[42] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, & Sergey Levine. (2019). Soft Actor-Critic Algorithms and Applications.
URL:https://arxiv.org/abs/1812.05905

Παραρτήματα

Γλωσσάριο

Υπολογιστικό νέφος (Cloud computing): Η προσφορά υπολογιστικών πόρων σε πελάτες μέσω του διαδικτύου.
Ενορχήστρωση (Orchestration): Η αυτόματη ρύθμιση, οργάνωση και διαχείριση υπολογιστικών συστημάτων και λογισμικού.
Ελαστικότητα (Elasticity): Δυνατότητα κλιμάκωσης εφαρμογών προς τα πάνω (αύξηση πόρων) και προς τα κάτω (αποδέσμευση πόρων).
Εικονική Μηχανή (Virtual Machine/VM): Μηχανισμός εικονικοποίησης που αποσκοπεί στην προσομοίωση ολόκληρων υπολογιστικών αρχιτεκτονικών.
Container: Μηχανισμός εικονικοποίησης σε επίπεδο λειτουργικού συστήματος. Με τη χρήση τους καθίσταται δυνατή η εκτέλεση πολλαπλών αντιγράφων στο ίδιο λειτουργικό σύστημα.
Container image: Ελαφρύ, αυτοδύναμο και εκτελέσιμο πακέτο λογισμικού που περιέχει όλα τα προαπαιτούμενα για την εκτέλεση μιας εφαρμογής. Αποτελούν τη βάση για την εκτέλεση των container.
Docker: Λογισμικό που επιτρέπει τη δημιουργία και την εκτέλεση container.
Kubernetes: Σύστημα ενορχήστρωσης container που παρέχει βασικούς μηχανισμούς για την ανάπτυξη, συντήρηση και κλιμάκωση εφαρμογών.
Kubernetes node: Φυσικά ή εικονικά μηχανήματα τα οποία χρησιμοποιεί το Kubernetes για την εκτέλεση των containers.
Kubernetes pod: Το μικρότερο δομικό στοιχείο του Kubernetes. Αποτελείται από ένα ή περισσότερα containers τα οποία ανατίθενται στον ίδιο κόμβο.
Kubernetes service: Διεπαφή που καθιστά δυνατή την πρόσβαση σε μια εφαρμογή αναθέτοντας της μια διεύθυνση IP και προωθεί αιτήματα στα κατάλληλα pods.
Οριζόντια κλιμάκωση (Horizontal scaling): Κλιμάκωση της υποδομής προσθέτοντας περισσότερους κόμβους ή αντίγραφα.
Κατακόρυφη κλιμάκωση (Vertical scaling): Κλιμάκωση της υποδομής προσθέτοντας πόρους (υπολογιστική ισχύ, μνήμη, χώρο αποθήκευσης) σε έναν κόμβο.
Service Level Agreement (SLA): Συμφωνία μεταξύ παρόχου και πελάτη που καθορίζει την ποιότητα και διαθεσιμότητα της εφαρμογής και τις υποχρεώσεις του παρόχου.
Νευρωνικό δίκτυο: Κύκλωμα συνδεδεμένων τεχνητών νευρώνων που μπορεί να εκπαιδευτεί για την επίλυση αλγοριθμικών προβλημάτων.

Ανοικτό Αποθετήριο Κώδικα

Η ανάπτυξη του κώδικα για τη μοντελοποίηση των περιβαλλόντων και τη δημιουργία πρακτόρων ενισχυτικής μάθησης έγινε με τη χρήση ενός αποθετηρίου κώδικα στο Gitlab.

Ο κώδικας είναι γραμμένος σε Python και τα περιβάλλοντα που αναπτύχθηκαν με το OpenAI Gym είναι διαθέσιμα ως ένα πακέτο Python. Οι πράκτορες ενισχυτικής μάθησης εκτελέστηκαν με τη βοήθεια Jupyter Notebooks και η δημιουργία και εκτέλεση συναρτήσεων έγινε πάνω στο Kubernetes και το Kubeless. Η πλήρης υποδομή περιγράφεται στην ενότητα 3.6. Η περιγραφή της λειτουργίας των περιβαλλόντων μαζί με οδηγούς για το στήσιμο και την προετοιμασία τους βρίσκεται στο Wiki του αποθετηρίου.

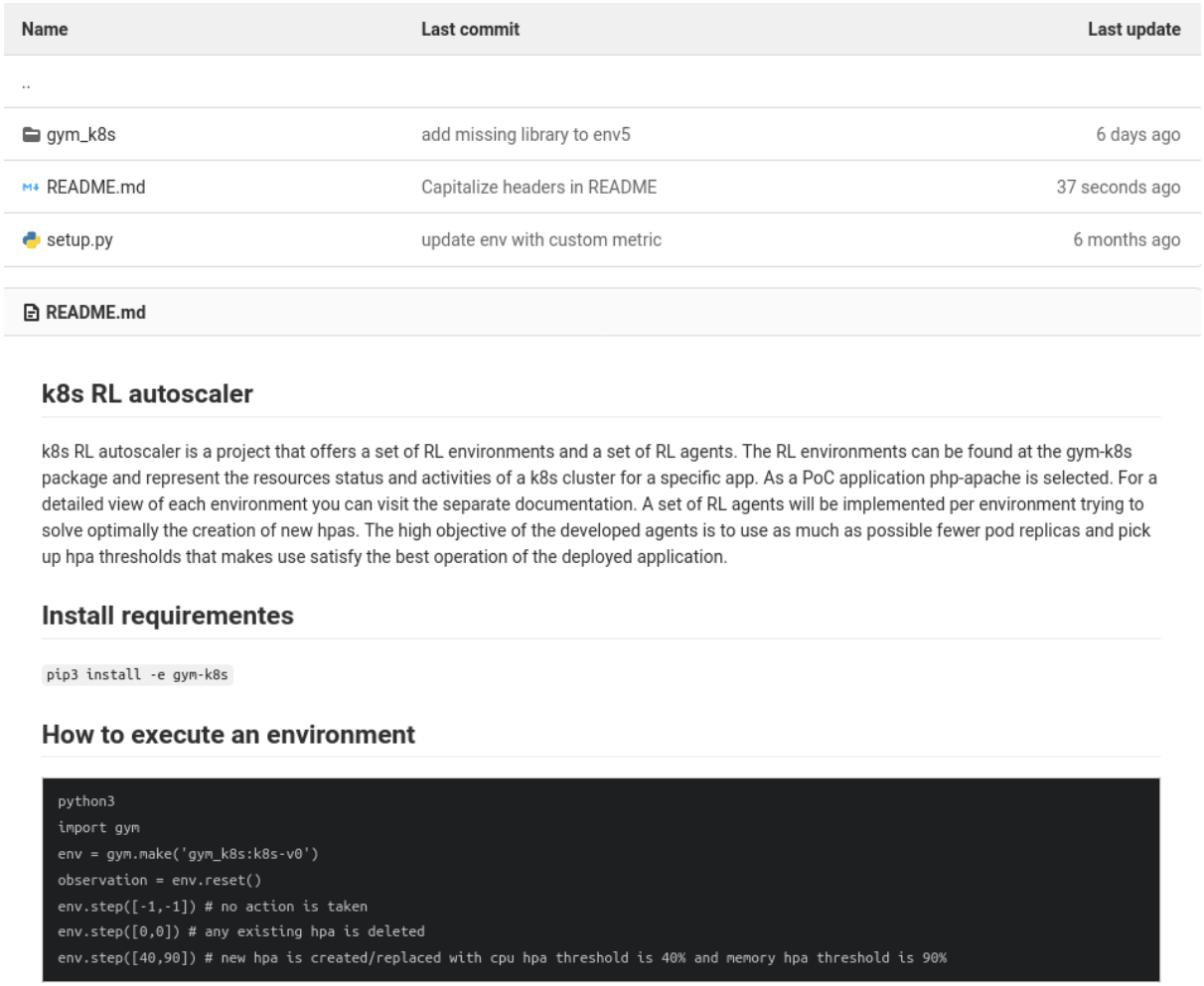
Οι εκδόσεις του λογισμικού που χρησιμοποιήθηκαν είναι οι εξής:

- Kubernetes: 1.19.4
- Kubeless: 1.0.8
- python: 3.7.10
- tensorflow: 2.4.1
- tf-agents: 0.7.1

Για το στήσιμο του serverless περιβάλλοντος δημιουργήθηκε επίσης ένα αποθετήριο κώδικα που περιέχει παραδείγματα serverless συναρτήσεων καθώς και scripts που διευκολύνουν την εγκατάσταση των απαραίτητων εργαλείων.

Το αποθετήριο κώδικα για τα περιβάλλοντα βρίσκεται στην εξής τοποθεσία:
https://gitlab.com/netmode/k8s-rl-autoscaler

Το αποθετήριο κώδικα για το στήσιμο των serverless περιβαλλόντων βρίσκεται στην εξής τοποθεσία:
https://gitlab.com/Nickgraviton/serverless-elasticity-management



Name	Last commit	Last update
..		
gym_k8s	add missing library to env5	6 days ago
README.md	Capitalize headers in README	37 seconds ago
setup.py	update env with custom metric	6 months ago

README.md

k8s RL autoscaler

k8s RL autoscaler is a project that offers a set of RL environments and a set of RL agents. The RL environments can be found at the gym-k8s package and represent the resources status and activities of a k8s cluster for a specific app. As a PoC application php-apache is selected. For a detailed view of each environment you can visit the separate documentation. A set of RL agents will be implemented per environment trying to solve optimally the creation of new hpas. The high objective of the developed agents is to use as much as possible fewer pod replicas and pick up hpa thresholds that makes use satisfy the best operation of the deployed application.

Install requirementes

```
pip3 install -e gym-k8s
```

How to execute an environment

```
python3
import gym
env = gym.make('gym_k8s:k8s-v0')
observation = env.reset()
env.step([-1,-1]) # no action is taken
env.step([0,0]) # any existing hpa is deleted
env.step([40,90]) # new hpa is created/replaced with cpu hpa threshold is 40% and memory hpa threshold is 90%
```

Εικόνα 1: Πακέτο python για τη χρήση περιβαλλόντων με το OpenAI Gym στο αποθετήριο

Name	Description	State metrics	SLA metrics	States	Actions	Implemented	Agents
k8s_env_with_performance_metric	Continuous environment with continuous state and discrete actions. Detailed definition	CPU, Memory, Pods, Terminated pods, HPA error, Throughput, Latency	Latency, Throughput	N/A	124	Yes	Random
k8s-env-discrete-state-discrete-action-V0	Discretized environment. Detailed definition	CPU, Memory, Pods, Latency	Latency	12.000	9	Yes	Random
k8s-env-discrete-state-discrete-action-V2	Discretized environment. Detailed definition	Memory, Pods, Latency	Latency	175	3	Yes	Random, Qlearning
k8s-env-discrete-state-discrete-action-V3	Discretized environment. Detailed definition	CPU, Pods, Latency	Latency	175	3	Yes	Qlearning
k8s-env-discrete-state-discrete-action-V4	Discretized environment (same as V3 but with 20% step changes in the HPA's threshold). Detailed definition	CPU, Pods, Latency	Latency	175	3	Yes	Qlearning
k8s-env-discrete-state-discrete-action-V5	Discretized environment. Detailed definition	CPU÷CPU Threshold, Pods, Latency	Latency	245	3	Yes	Qlearning, DynaQ+
k8s-env-discrete-state-discrete-action-V6	Discretized environment. Detailed definition	CPU, CPU Threshold, Pods, Latency	Latency	1225	3	Yes	Simulated Environment Qlearning, Simulated Environment DynaQ+
k8s-env-continuous-state-discrete-action-V0	Continuous environment with continuous state and discrete actions. Detailed definition	CPU, CPU threshold, Memory, Memory threshold, Pods, Terminated pods, HPA error, Throughput level, Throughput Rate, Latency	Latency, Throughput	N/A	9	Yes	DQN
k8s-env-continuous-state-discrete-action-V1	Continuous environment with continuous state and discrete actions. Detailed definition	CPU, CPU threshold, Pods, Terminated pods, HPA error, Latency	Latency	N/A	3	Yes	DQN, Simulated Environment DQN

Εικόνα 2: Εποπτικός πίνακας αναπτυχθέντων περιβαλλόντων από το wiki του αποθετηρίου