Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
ΔΠΜΣ Επιστήμη Δεδομένων και
Μηχανική Μάθηση

Σχεδιασμός και υλοποίηση ευφυούς μηχανισμού διαμοιρασμού κοινόχρηστων πόρων, σε πολυπύρηνα συστήματα, με χρήση βαθιάς ενισχυτικής μάθησης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΝΙΚΗΦΟΡΟΣ Ι. ΜΑΝΔΗΛΑΡΑΣ

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Αύγουστος 2020

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
ΔΠΜΣ Επιστήμη Δεδομένων και
Μηχανική Μάθηση

Σχεδιασμός και υλοποίηση ευφυούς μηχανισμού διαμοιρασμού κοινόχρηστων πόρων, σε πολυπύρηνα συστήματα, με χρήση βαθιάς ενισχυτικής μάθησης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΝΙΚΗΦΟΡΟΣ Ι. ΜΑΝΔΗΛΑΡΑΣ

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 31η Αυγούστου 2020.

Νεκτάριος Κοζύρης	Διονύσιος Πνευματικάτος	Γεώργιος Γκούμας
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.

Αθήνα, Αύγουστος 2020

Νικηφόρος Ι. Μανδηλαράς

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

Περίληψη

Η μέση χρήση των εξυπηρετητών στα σύγχρονα κέντρα δεδομένων είναι εξαιρετικά χαμηλή, μη ξεπερνώντας το 50%. Αιτία για αυτό αποτελούν τα όρια επίδοσης που συνάπτουν οι πάροχοι με τους πελάτες, καθώς προκειμένου αυτά να διασφαλίζονται, προτιμάται η απομονωμένη εκτέλεση των υπηρεσιών. Η ανάγκη για απομόνωση προκύπτει λόγω του ανταγωνισμού για κοινόχρηστους πόρους, όπως είναι η κρυφή μνήμη τελευταίου επιπέδου. Ο ανταγωνισμός που εμφανίζεται μεταξύ των συνεκτελούμενων εφαρμογών, επηρεάζει αρνητικά την απόδοση των υπηρεσιών και θέτει εν αμφιβόλω την διατήρηση του επιπέδου επίδοσής τους.

Για την αντιμετώπιση τέτοιων καταστάσεων, έχουν πλέον ενσωματωθεί στους σύγχρονους επεξεργαστές, τεχνολογίες που παρέχουν υποστήριξη για την επίβλεψη της χρήσης αλλά και για το διαμοιρασμό κοινόχρηστων πόρων.

Στην παρούσα διπλωματική, αξιοποιώντας τις εν λόγω τεχνολογίες, όπως και μεθόδους βαθιάς ενισχυτικής μάθησης, υλοποιούμε έναν ευφυή μηχανισμό διαμοιρασμού της κρυφής μνήμης τελευταίου επιπέδου, ενός πολυπύρηνου συστήματος. Στόχος είναι η διατήρηση της επίδοσης μιας κρίσιμης υπηρεσίας, όταν αυτή συνεκτελείται με άλλες εφαρμογές, αλλά ταυτόχρονα και η αύξηση της χρησιμοποίησης των πόρων του συστήματος. Η ενισχυτική μάθηση επιτρέπει την αυτοματοποιημένη υλοποίηση τέτοιων στόχων, με τη χρήση πρακτόρων που διερευνούν έναν χώρο καταστάσεων και αξιοποιούν τη γνώση που συλλέγουν για το περιβάλλον, ώστε να λάβουν τις κατάλληλες αποφάσεις για την επίτευξη του τελικού στόχου.

Αξιολογούμε τον μηχανισμό μας σε συνεκτελέσεις της υπηρεσίας Memcached με εφαρμογές μηχανικής μάθησης. Αποδεικνύουμε πως ο μηχανισμός μπορεί να προφυλάξει με συνέπεια την επίδοση της κρίσιμης υπηρεσίας και ταυτόχρονα να αυξήσει τη διεκπεραιωτικότητα των εφαρμογών χαμηλής προτεραιότητας. Τέλος δείχνουμε πως η μάθηση νευρωνικών δικτύων προσφέρει δυνατότητες γενίκευσης της αποκτώμενης γνώσης και χρήσης αυτής σε νέες εφαρμογές.

Λέξεις κλειδιά

Πολυεπεξεργαστικά συστήματα, Κοινόχρηστη κρυφή μνήμη, LLC, Διαμοιρασμός κοινόχρηστης μνήμης, Συνεκτέλεση εφαρμογών, Intel RDT, Ενισχυτική Μάθηση, Νευρωνικά Δίκτυα, Βαθιά Ενισχυτική Μάθηση, DQN

Abstract

The average usage of servers in modern data centers is extremely low, not exceeding 50 %. The reason for this, is the Service-Level Agreements (SLAs) that the providers sign with their customers. In order to ensure those agrements, the isolated execution of the services is preferred. The need for isolation arises due to the competition for shared resources, such as the last level cache memory. The competition that occurs between the coexecuted applications, negatively affects the performance of the services and calls into question the maintenance of their level of performance.

To deal with such situations, technologies have now been integrated into modern processors, that provide support for usage monitoring as well as for partitioning of shared resources.

In the present thesis, we utilize these technologies along with deep reinforcement learning methods, in order to implement an intelligent mechanism for partitioning the last level cache of a multicore system. The goal is to maintain the performance of a latency critical service when it is coexecuted with other applications, but also to increase the utilization of system resources. Reinforcement learning enables the automated implementation of such goals, using agents who explore a state space and utilize the knowledge they gather from the environment, in order to make the appropriate decisions and achieve their ultimate goal.

We evaluate our mechanism in coexecutions of Memcached service with machine learning workloads. We prove that the mechanism can consistently protect the performance of the critical service and at the same time increase the throughput of low priority applications. Finally, we show that the training of neural networks offers opportunities to generalize the acquired knowledge and use it in new applications.

Key words

Multiprocessors, Shared cache, LLC, Cache partitioning, coexecution, Intel RDT, Reinforcement Learning, Neural Nets, Deep Reinforcement Learning, DQN

Ευχαριστίες

Πρώτα απ΄ όλα, θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή αυτής της διπλωματικής κ. Νεκτάριο Κοζύρη, για την ευκαιρία που μου έδωσε, να ασχοληθώ με το συγκεκριμένο θέμα αλλά και για την έμπνευση και το ενδιαφέρον που μου καλλιέργησε, κατά τη διάρκεια των σπουδών μου.

ιδιαίτερες ευχαριστίες θα ήθελα να αποδώσω στον Ερευνητή κ. Κωνσταντίνο Νίκα για την καθοδήγησή του και τη διαρκή και άμεση στήριξη του. Πέρα των άλλων το ενδιαφέρον του για το θέμα της διπλωματικής εργασίας, μου έδωσε σημαντικό κίνητρο για την ολοκλήρωση αυτής.

Ακόμα, θα ήθελα να ευχαριστήσω όλη την ομάδα του Εργαστηρίου Υπολογιστικών Συστημάτων της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, για τη συνεχή βοήθεια που μου παρείχαν, η οποία συνέβαλε ουσιαστικά στην ομαλή διεξαγωγή αυτής της διπλωματικής.

Τέλος, θα ήθελα να ευχαριστήσω όλους τους ανθρώπους που στάθηκαν δίπλα μου και συνέβαλαν κατά τρόπο ξεχωριστό, στο απαιτητικό αυτό διάστημα των σπουδών μου.

Νικηφόρος Ι. Μανδηλαράς,

Αθήνα, 31η Αυγούστου 2020

Περιεχόμενα

Κατάλογος πινάκων

Κατάλογος σχημάτων

Κατάλογος αλγορίθμων

Κεφάλαιο 1

Εισαγωγή

1.1 Προκλήσεις στα σύγχρονα κέντρα δεδομένων

Τα σύγχρονα κέντρα δεδομένων καλούνται να εξυπηρετήσουν ετερόκλιτες ανάγκες. Από τη μία οι μεγάλοι οργανισμοί φιλοξενούν στα κέντρα δεδομένων τους ένα πλήθος υπηρεσιών όπως μηχανές αναζήτησης, βάσεις δεδομένων, εικονοποιημένες δικτυακές λειτουργίες (Network Virtual Functions, NFV) αλλά και κάθε είδους υπηρεσίες ιστού. Από την άλλη, με την ανάπτυξη των υπηρεσιών Υπολογιστικού Νέφους (Cloud Services), ολοένα και περισσότερες εταιρίες επιλέγουν τα κέντρα δεδομένων των παρόχων για την κάλυψη των υπολογιστικών τους αναγκών έναντι ιδιόκτητων υποδομών, με στόχο τη μείωση των λειτουργικών εξόδων. Ιδιαίτερα τα τελευταία χρόνια οι σημαντικές υπολογιστικές απαιτήσεις των αλγορίθμων μηχανικής μάθησης και επεξεργασίας δεδομένων, που ήρθαν δυναμικά στο προσκήνιο, οδήγησαν τις περισσότερες εταιρίες στη χρήση cloud πόρων για την εξυπηρέτηση αυτών.

Παραδοσιακά στα κέντρα δεδομένων προτιμάται η απομόνωση των κρίσιμων υπηρεσιών με δέσμευση ολόκληρου του μηχανήματος, αποκλειστικά για τις ανάγκες αυτών. Διασφαλίζεται έτσι ότι αυτές πληρούν τα όρια που τίθενται, ακόμα και στο χειρότερο σενάριο. Ωστόσο η χρήση των μηχανημάτων είναι εξαιρετικά χαμηλή, καθώς οι υπηρεσίες αυτές τα απασχολούν πλήρως για ένα μικρό μόνο ποσοστό του χρόνου.

Ιδανικά για τους παρόχους υπηρεσιών νέφους, υπηρεσίες και μη άμεσες εργασίες (offline-batch jobs), όπως αυτές της μηχανικής μάθησης, θα μπορούσαν να συνεκτελούνται στο ίδιο μηχάνημα αν διασφαλιζόταν, ότι τα όρια των υπηρεσιών δε θα παραβιάζονταν. Ζητούμενο λοιπόν είναι ένας μηχανισμός διαμοιρασμού των πόρων των εξυπηρετητών, που να μπορεί να προστατεύει την επίδοση της κρίσιμης υπηρεσίας κατά τη συνεκτέλεση, αλλά ταυτόχρονα να διαθέτει πόρους στις εφαρμογές χαμηλής προτεραιότητας όποτε το επιτρέπουν οι συνθήκες.

Για την αντιμετώπιση αυτής της ανάγκης πολλές θεωρήσεις έχουν λάβει χώρα ([1], [2], [3], [4]). Σκοπός των εν λόγω εργασιών είναι κυριότερα ο έλεγχος του ανταγωνισμού που εμφανίζεται στην κρυφή μνήμη τελευταίου επιπέδου (Last Level Cache, LLC), καθώς η προστασία της πρόσβασης στην LLC είναι κομβική για την ποιότητα

των υπηρεσιών. Πιο πρόσφατες μελέτες επιστρατεύουν στην προσπάθεια αυτή και μεθόδους μηχανικής μάθησης, προκειμένου με αυτοματοποιημένο τρόπο να αναγνωρίσουν τις πτυχές του προβλήματος.

1.2 Συνεισφορά

Στα πλαίσια της παρούσας διπλωματικής εργασίας, μελετάται ένα σενάριο συνεκτέλεσης εφαρμογών, μιας υπηρεσίας με συγκεκριμένα όρια ως προς το χρόνο απόκρισης της (latency) σε εισερχόμενη κίνηση και διαφόρων διεργασιών χαμηλής προτεραιότητας. Στόχος μας είναι η διασφάλιση της ποιότητας της υπηρεσίας και ταυτόχρονα η όσο το δυνατό μεγαλύτερη αξιοποίηση του εξυπηρετητή, μέσω της διάθεσης πόρων στις λοιπές εφαρμογές.

Ο μηχανισμός που υλοποιείται για την αντιμετώπιση του συγκεκριμένου προβλήματος, βασίζεται σε μετρικές υλικού και λογισμικού, προκειμένου κάθε στιγμή να αποφαίνεται για τον τρόπο με τον οποίο θα διαμοιράσει την κρυφή μνήμη τελευταίου επιπέδου. Για το λόγο αυτό κάνουμε χρήση των τεχνολογιών που παρέχονται από την Intel και καθιστούν εφικτή την επίβλεψη της χρήσης της LLC (Cache Monitoring Technology, CMT) και το διαμοιρασμό αυτής, μεταξύ υπηρεσίας και εφαρμογών (Cache Allocation Technology, CAT).

Στόχος μας είναι να καταφέρουμε να εξάγουμε από τις μετρικές που συλλέγουμε τα πρότυπα που διέπουν το πρόβλημα και πάνω σε αυτή τη γνώση να βασίσουμε αυτοματοποιημένα τις όποιες αποφάσεις μας. Έτσι αντί ευριστικών μεθόδων που έχουν χρησιμοποιηθεί κατά κόρον, αξιοποιούμε μεθόδους βαθιάς ενισχυτικής μάθησης, μιας τεχνικής που συνδυάζει το πεδίο που καλείται ενισχυτική μάθηση, με την δυναμική των νευρωνικών δικτύων.

Η ενισχυτική μάθηση υποθέτει έναν ευφυή πράκτορα (agent), ο οποίος με γνώση των μετρικών του περιβάλλοντος της συνεκτέλεσης, λαμβάνει μία απόφαση για το διαμοιρασμό των πόρων. Την απόφασή του, την εφαρμόζει άμεσα στο σύστημα και ανάλογα με τα αποτελέσματα αυτής λαμβάνει μια ανταμοιβή και τη νέα κατάσταση του συστήματος. Στόχος του πράκτορα είναι να μεγιστοποιήσει την ανταμοιβή του, μαθαίνοντας να παίρνει τις κατάλληλες αποφάσεις.

1.3 Οργάνωση του τόμου

Η διπλωματική εργασία είναι οργανωμένη σε έξι κεφάλαια. Στο κεφάλαιο 2 παρουσιάζουμε το πρόβλημα της συνεκτέλεσης εφαρμογών στα σύγχρονα κέντρα δεδομένων και αναφέρουμε συγκεκριμένες προσεγγίσεις που έχουν γίνει πάνω σε αυτό. Στο κεφάλαιο 3 παρουσιάζεται αναλυτικά η θεωρία της ενισχυτικής μάθησης, καθώς και η χρήση νευρωνικών δικτύων στη βαθιά ενισχυτική μάθηση. Στο κεφαλαίο 4 έχουμε την περιγραφή του συστήματος, στο οποίο διεξήχθησαν τα πειράματά μας. Παρουσιάζονται αναλυτικά

τα διάφορα τμήματα που χρησιμοποιήθηκαν και ο τρόπος διασύνδεσης αυτών. Έπειτα στο 5ο κεφάλαιο έχουμε την αναλυτική παρουσίαση των πειραμάτων που διεξήχθησαν. Τέλος στο κεφάλαιο 6 καταθέτουμε τα συμπεράσματά μας, συνοψίζουμε τη μελέτη που έχει γίνει και προτείνουμε πιθανές επεκτάσεις.

Κεφάλαιο 2

Υπόβαθρο

2.1 Εισαγωγή

Οι υπηρεσίες κρίσιμης απόκρισης που φιλοξενούνται σε κέντρα δεδομένων, όπως είναι μηχανές αναζήτησης, βάσεις δεδομένων, υπηρεσίες βίντεο, μουσικής, ηλεκτρονικού ταχυδρομείου και άλλες, αξιοποιούν πλήρως τους πόρους που τους ανατίθενται για ένα μικρό μόνο διάστημα του χρόνου. Συνηθίζεται σε τέτοιες παραγωγικές υπηρεσίες να προδιαγράφονται οι πόροι για την περίπτωση του χειρότερου σεναρίου και έτσι σε διαστήματα χαμηλής κίνησης, μεγάλο μέρος αυτών, μένει αναξιοποίητο. Το γεγονός ότι αυτό το πλεόνασμα πόρων αποφεύγεται να επαναχρησιμοποιείται οφείλεται στο ότι ο ανταγωνισμός που εμφανίζεται για μοιραζόμενους πόρους, μπορεί να βλάψει την επίδοση των υπηρεσιών κρίσιμης απόκρισης, οδηγώντας σε παραβιάσεις των συμφωνηθέντων ορίων λειτουργίας. Η συνεπαγόμενη υποχρησιμοποίηση των πόρων έχει σαν συνέπεια τη μείωση της αποδοτικότητας των κέντρων δεδομένων και καθώς οι ανάγκες για υπολογιστική δύναμη αυξάνονται ραγδαία, κρίνεται επιτακτική η διαχείριση του ανταγωνισμού για κοινόχρηστους πόρους.

Είναι γεγονός πως υπάρχουν μηχανισμοί λογισμικού, που μπορούν να απομονώσουν την εκτέλεση των διεργασιών, όσο αφορά συγκεκριμένους πόρους, δηλαδή πυρήνες εκτέλεσης, μέγεθος μνήμης και χρήση του δικτύου. Παρόλα αυτά υπάρχουν και πόροι που δεν μπορούν να διαμοιραστούν με τέτοιους μηχανισμούς, όπως είναι η κρυφή μνήμη τελευταίου επιπέδου και το bandwidth της κύριας μνήμης.

Οι σύγχρονοι επεξεργαστές διαθέτουν πολλούς πυρήνες καθώς και ιεραρχία κρυφών μνημών (on-chip memory), προκειμένου να μειώνονται οι προσβάσεις του επεξεργαστή στη κύρια μνήμη. Τμήμα αυτής της ιεραρχίας μνημών είναι ιδιωτικό για τον καθένα από τους πυρήνες του επεξεργαστή, ενώ συνηθέστερα το τελευταίο τμήμα της (Last-Level Cache, LLC) είναι μοιραζόμενο μεταξύ όλων των επεξεργαστών της μονάδας. Λόγο αυτού, δημιουργείται ανταγωνισμός στη LLC μεταξύ εφαρμογών, που εκτελούνται στην ίδια επεξεργαστική μονάδα, ακόμα και αν αυτές διαθέτουν αποκλειστική πρόσβαση σε πυρήνες, κύρια μνήμη, δίκτυο. Ο ανταγωνισμός για τη LLC μπορεί να οδηγήσει ακόμα και σε κορεσμό του bandwidth της κύριας μνήμης, πόρος που επίσης δε μπορεί να διαμοιραστεί με κάποιον μηχανισμό λογισμικού, καθώς οι εφαρμογές 'διώχνουν' συνεχώς blocks η μία της άλλης. Συνέπεια του αυξημένου miss rate είναι η ενδεχόμενη κακή

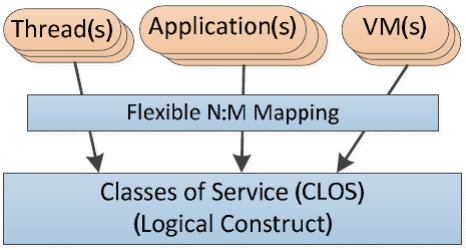
επίδοση των συνεκτελούμενων εφαρμογών όπως και η αυξημένη κατανάλωση ενέργειας.

Για την αντιμετώπιση αυτού του προβλήματος διάφοροι μηχανισμοί έχουν αναπτυχθεί. Σε αυτή την κατεύθυνση έχει συμβάλλει και η ανάπτυξη τεχνολογιών διαμοιρασμού με υποστήριξη από το υλικό, όπως είναι η τεχνολογία RDT [5], που ενσωματώνεται στους νέους επεξεργαστές της Intel.

2.2 Intel RDT

Οι νέοι επεξεργαστές της οικογένειας Intel Xeon E5 v3 και έπειτα υποστηρίζουν την Resource Director Technology (RDT). Ουσιαστικά πρόκειται για ένα σύνολο τεσσάρων διαφορετικών τεχνολογιών που αφορούν την παρακολούθηση και τον διαμοιρασμό της LLC αλλά και του Bandwidth κύριας μνήμης. Αυτές είναι, η Cache Monitoring Technology (CMT), η Cache Allocation Technology (CAT), η Memory Bandwidth Monitoring (MBM) και η Memory Bandwidth Allocation (MBA).Η τεχνολογία CMT επιβλέπει τη χρήση της LLC από ομάδα είτε πυρήνων είτε εκτελούμενων εφαρμογών. Αντίστοιχα η τεχνολογία MBM επιβλέπει τη χρήση του bandwidth προς την κύρια μνήμη. Από την άλλη οι τεχνολογίες CAT και MBA δίνουν τη δυνατότητα καθορισμού του τμήματος της LLC ή του bandwidth αντίστοιχα, που θα διατεθεί σε κάθε ομάδα είτε πυρήνων είτε εφαρμογών. Όλες αυτές οι τεχνολογίες υποστηρίζονται από το υλικό με προσθήκη καταχωρητών.

Όσο αφορά την τεχνολογία CAT, η αντιστοίχιση μεταξύ πυρήνων ή διεργασιών και μεριδίου στη LLC, γίνεται μέσω ενός επιπέδου αφαίρεσης, το Class of Service (COS). Ο επεξεργαστής διαθέτει ένα πλήθος διαφορετικών COS, όπου το καθένα αποτελεί ένα είδος bitmask σε επίπεδο ways, που προσδιορίζει το τμήμα της cache, στο οποίο δίνει πρόσβαση. Άρα για την ανάθεση ενός τμήματος LLC απαιτούνται δύο ενέργειες. Ο προσδιορισμός του επιθυμητού bitmask για κάποιο COS και η αντιστοίχιση αυτού με την ομάδα πυρήνων ή διεργασιών που επιθυμούμε. Να αναφέρουμε ότι το COS προσδιορίζει μόνο τα ways στα οποία οι πυρήνες ή οι διεργασίες έχουν δυνατότητα εγγραφής νέων blocks, ενώ τα reads αυτών μπορεί να επιτύχουν στο σύνολο της cache.



Σχήμα 2.1: Συσχέτιση με COS

2.3 Διαχείριση ανταγωνισμού για την Κοινόχρηστη Μνήμη

Στη συνέχεια περιγράφουμε κάποιους από αυτούς τους μηχανισμούς που συναντήσαμε στη βιβλιογραφία κατά τη διάρκεια ανάπτυξης αυτής της διπλωματικής εργασίας. Μπορεί οι στόχοι αυτών ή οι μέθοδοι που εφαρμόζουν να διαφέρουν, πάντως έχουν σαν κοινό παρανομαστή τη διαχείριση του ανταγωνισμού σε μοιραζόμενους πόρους.

Στη μελέτη των K. Nikas κ.α. ([1]) που υλοποιούν τον Dicer, έναν μηχανισμό δυναμικού διαχωρισμού κρυφής μνήμης, επιχειρείται η ικανοποίηση των αναγκών μιας εφαρμογής υψηλής προτεραιότητας και η ταυτόχρονη υψηλή χρησιμοποίηση του server καθώς παράλληλα συνεκτελούνται, σε διαφορετικούς πυρήνες, άλλες εφαρμογές χαμηλής προτεραιότητας. Ο μηχανισμός αυτός παρακολουθεί hardware μετρικές των εφαρμογών και με ένα σύνολο ευριστικών κανόνων λαμβάνει περιοδικά απόφαση, για τα ways που θα αναθέσει στις δύο ομάδες εφαρμογών. Η συλλογή των hardware μετρικών και η εφαρμογή των αποφάσεων γίνεται με χρήση της τεχνολογίας RDT.

Στην εργασία των Lo κ.α. ([2]) επιχειρείται μια πιο ολιστική προσέγγιση του προβλήματος της συνεκτέλεσης εφαρμογών. Στόχος είναι η προφύλαξη όλων των μοιραζόμενων πόρων από το να φτάσουν σε κορεσμό, καθώς διαπιστώνεται, ότι αυτή είναι η αιτία που οδηγεί σε κακή ποιότητας υπηρεσία. Ως εκ τούτου, ο μηχανισμός Heracles που υλοποιούν, έχει διαφορετικούς ελεγκτές για ομάδες μοιραζόμενων πόρων, οι οποίοι λαμβάνουν αποφάσεις για το διαμοιρασμό, με βάση και πάλι κάποιους ευριστικούς κανόνες.

Ο Heracles δίνει επιπλέον έμφαση στην προστασία της υπηρεσίας υψηλής προτεραιότητας και διακόπτει τη συνεκτέλεση, όταν τα όρια που έχουν τεθεί πιέζονται. Επίσης, σε αντίθεση με την προηγούμενη μελέτη, λαμβάνονται υπόψιν και software μετρικές της υπηρεσίας. Μια τέτοια επιλογή έχει το πλεονέκτημα, ότι αναφερόμαστε στα μεγέθη που όντως χαρακτηρίζουν την ποιότητα της εφαρμογής, καθώς η χρήση του IPC ή και περισσότερων hardware metrics μπορεί να μην αποτυπώνει την εμπειρία του τελικού χρήστη. Από την άλλη όμως, υπάρχει απώλεια σε γενικότητα, αφού πιθανόν να απαιτείται ειδική μέριμνα για την εξαγωγή τέτοιων μετρικών, ανάλογα την υπηρεσία κρίσιμης απόκρισης.

Οι προσεγγίσεις που έχουμε αναφέρει έως τώρα χρησιμοποιούσαν κλασσικές ευριστικές μεθόδους. Σε πιο πρόσφατες δουλειές όμως, επιστρατεύονται και τεχνικές μηχανικής μάθησης, που γνώρισαν δυναμική ανάπτυξη τα τελευταία χρόνια. Συγκεκριμένα, χρησιμοποιείται συχνά η ενισχυτική μάθηση, καθώς ο online τρόπος λειτουργίας της ταιριάζει με τη φύση αυτής της κατηγορίας προβλημάτων.

Οι Y. Li κ.α. ([3]) ακολουθούν την προσέγγιση της ενισχυτικής μάθησης αντικαθιστώντας τους ευριστικούς κανόνες με έναν έξυπνο πράκτορα που λαμβάνει την ίδια είσοδο, δηλαδή hardware και software μετρικές και λαμβάνει μια απόφαση. Επιπλέον, όπως και άλλες δουλειές ([6, 7, 4]) αντιμετωπίζουν το πρόβλημα του ανταγωνισμού στην μοιραζόμενη κρυφή μνήμη τελευταίου επιπέδου με έμμεσο τρόπο, μειώνοντας δηλαδή τη

συχνότητα των πυρήνων που έχουν ανατεθεί στις εφαρμογές βέλτιστης προσπάθειας, όταν εντοπίζουν ότι αυτές, δημιουργούν αυξημένη πίεση στη λειτουργία της κρίσιμης υπηρεσίας.

Οι R. Nishtala κ. α. ([4]) προχωρούν ένα βήμα παραπέρα και χρησιμοποιούν έναν πράκτορα που βασίζεται σε βαθιά ενισχυτική μάθηση, έναν συνδυασμό δηλαδή ενισχυτικής μάθησης και τεχνητών νευρωνικών δικτύων. Μελετούν επίσης, το πως μπορούν να προσεγγίσουν το tail latency, που είναι συνήθως η μετρική που χαρακτηρίζει τη λειτουργία μιας υπηρεσίας, από hardware μετρικές. Το σενάριό τους διαφοροποιείται κάπως από τα προηγούμενα, με την έννοια ότι προσπαθούν να διαμοιράσουν τους διαθέσιμους πόρους, μεταξύ πολλών κρίσιμων υπηρεσιών.

2.4 Συνεκτελέσεις εφαρμογών

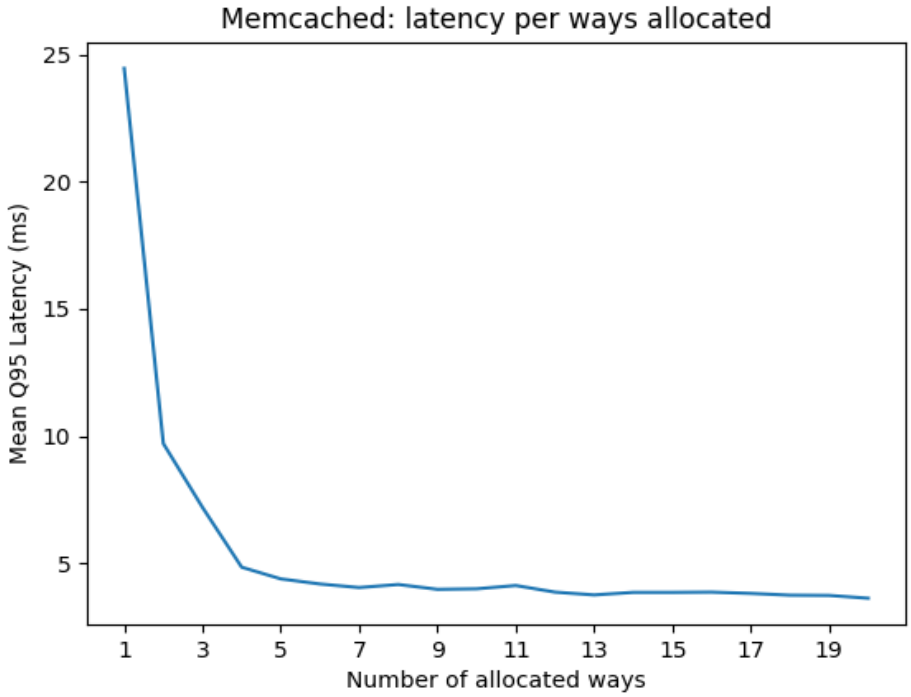
Στο παρόν τμήμα της διπλωματικής εργασίας θα μελετήσουμε τη συμπεριφορά της κρίσιμης υπηρεσίας Memcached [8], την οποία χρησιμοποιούμε στα πειράματά μας και αναλύουμε εκτενέστερα στην ενότητα 4.1, όταν αυτή συνεκτελείται με άλλες, χαμηλής προτεραιότητας εφαρμογές. Πρώτα από όλα, παραθέτουμε τα χαρακτηριστικά του επεξεργαστή Intel® Xeon® Processor E5-2630 v4, όπου διεξήχθησαν τα πειράματά μας.

Χαρακτηριστικά	Τιμές
Αρχιτεκτονική	Broadwell
Αριθμός πυρήνων	10
Αριθμός νημάτων	20
Βασική συχνότητα	2.20 GHz
LLC	25 MB
Assosiativity	20
DRAM Bandwidth	68.3 GB/sec

Πίνακας 2.1: Χαρακτηριστικά του επεξεργαστή Intel Xeon Processor E5-2630 v4

Ακόμα παρουσιάζουμε την επίδοση της υπηρεσίας Memcached, ως συνάρτηση του αριθμού ways που της αποδίδονται στην LLC. Όπως έχουμε αναφέρει το κύριο μέγεθος που μας απασχολεί για να χαρακτηρίσουμε την επίδοση μιας υπηρεσίας είναι το tail latency, δηλαδή ο χρόνος στον οποίο εξυπηρετείται σχεδόν το σύνολο των αιτήσεων που δέχεται.

Παρατηρούμε στην εικόνα 2.2, πως όταν εκτελείται μόνη της η υπηρεσία Memcached μπορεί να ικανοποιηθεί με ένα μικρό πλήθος ways. Από εκεί και έπειτα η χρήση επιπλέον ways δε φαίνεται να μειώνει αισθητά το latency. Συνεπώς υπάρχει περιθώριο να διατεθούν τα υπόλοιπα ways σε εφαρμογές χαμηλής προτεραιότητας.



Σχήμα 2.2: Επίδραση του αποδιδόμενου τμήματος μνήμης LLC στην επίδοση της υπηρεσίας Memcached

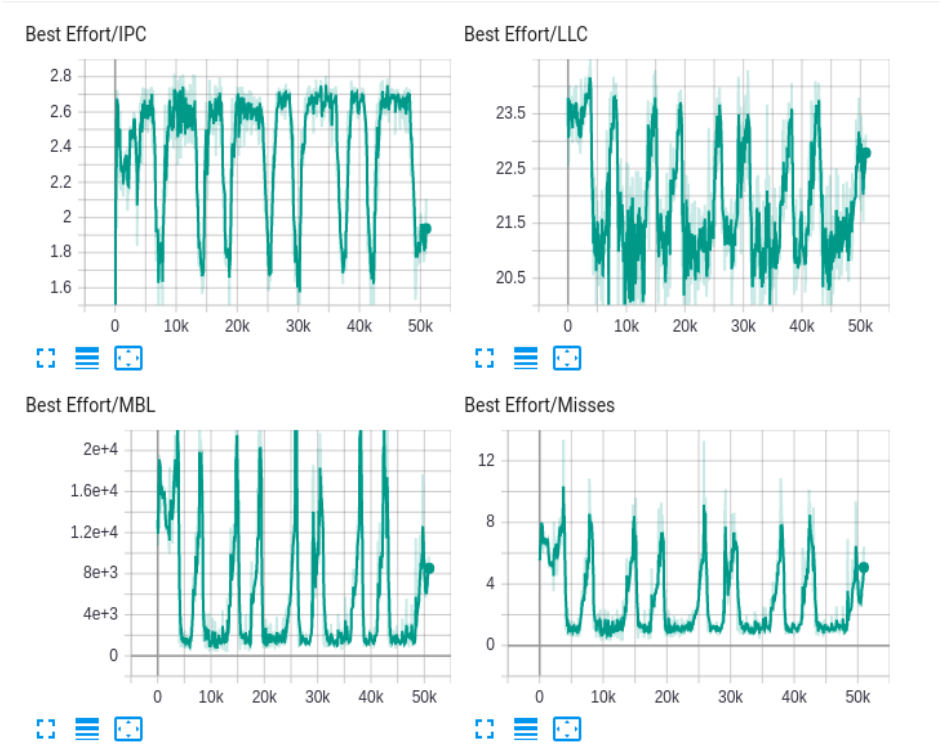
2.4.1 Επίδραση της συνεκτέλεσης στην επίδοση της υπηρεσίας

Στη συνέχεια εκτελούμε την υπηρεσία μαζί με εννέα αντίγραφα κάποιας εφαρμογής χαμηλής προτεραιότητας, προκειμένου να εντοπίσουμε αν επηρεάζουν και με ποιο τρόπο την επίδοση της εφαρμογής υψηλής προτεραιότητας. Περισσότερα για τις εφαρμογές καλύτερης προσπάθειας που χρησιμοποιούμε στα πειράματά μας, αναφέρουμε στην ενότητα 4.3.

Στο σχήμα 2.3 βλέπουμε τις διάφορες φάσεις του Spark Job in-memory analytics. Παρατηρούμε ότι περιοδικά εμφανίζονται spikes στα misses κάτι που οδηγεί, όπως είναι λογικό, σε πτώση του IPC και αύξηση της χρήσης του bandwidth, καθώς νέα δεδομένα πρέπει να έρθουν στην cache από την κύρια μνήμη.

Στον αντίποδα τώρα στο σχήμα 2.4 παρατηρούμε τις μετρικές για την υπηρεσία Memcached. Με μία πρώτη ματιά μπορούμε να πούμε πως η επίδραση της συνεκτέλεσης είναι εμφανέστατη, αφού βλέπουμε στις μετρικές της κρίσιμης υπηρεσίας να αποτυπώνονται οι φάσεις της εφαρμογής καλύτερης προσπάθειας. Έτσι λοιπόν η κατάληψη μεγαλύτερου τμήματος της LLC από την best effort εφαρμογή, επέφερε μείωση του τμήματος της κρίσιμης υπηρεσίας, κάτι που με τη σειρά του οδήγησε σε αύξηση των misses, πτώση του IPC και εν τέλει εκτόξευση του latency.

Παραθέτουμε στη συνέχεια στα σχήματα 2.5, 2.6 τις αντίστοιχες γραφικές και για

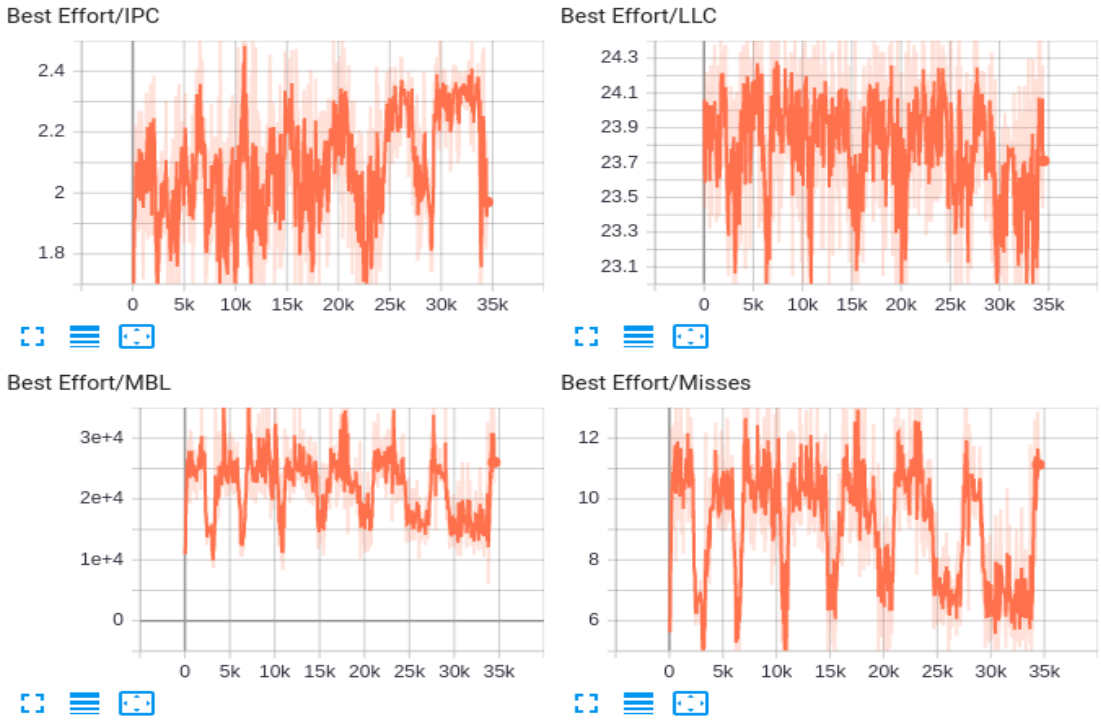


Σχήμα 2.3: Συμπεριφορά του Spark job In-Memory Analytics όπως αυτή αποτυπώνεται από τις μετρικές IPC, LLC χωρητικότητα, Memory Bandwidth και Misses ανά χίλιους κύκλους ρολογιού

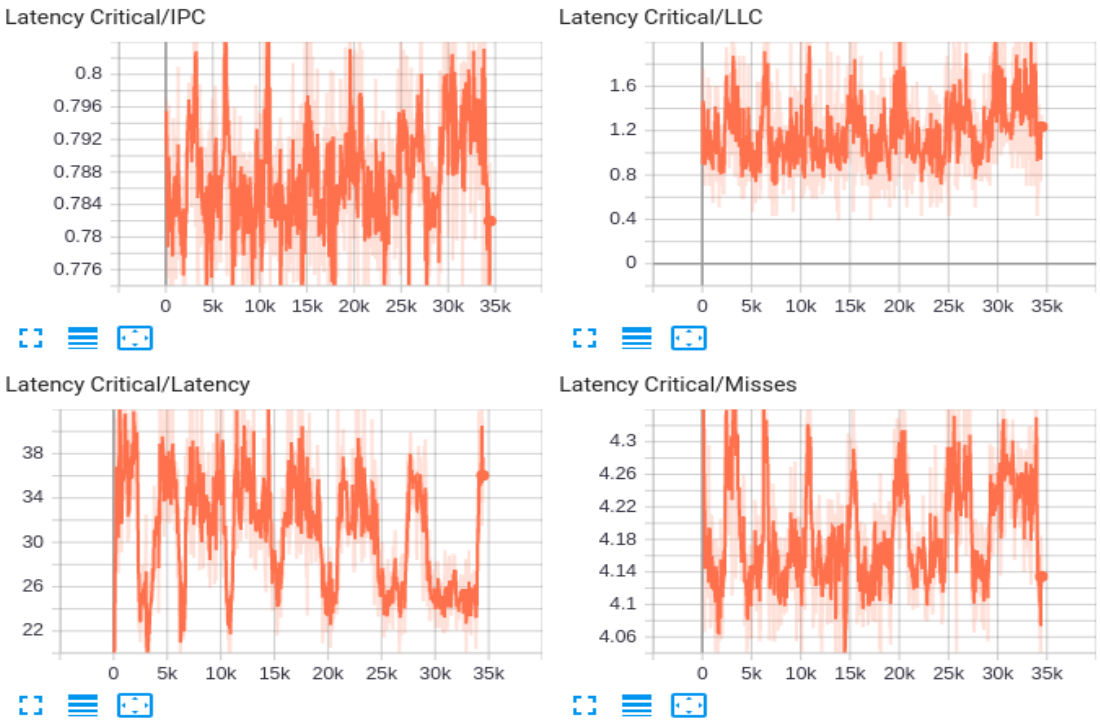
τη συνεκτέλεση της υπηρεσίας με το Spark Job Graph Analytics. Η εικόνα που παρατηρούμε είναι η ίδια με πριν, με τις φάσεις του Spark Job να επηρεάζουν και πάλι σαφέστατα όλες τις μετρικές της υπηρεσίας.

Τέλος, στην εικόνα 2.7 καταγράφουμε το bandwidth που χρησιμοποίησε η Memcached προς την κύρια μνήμη. Και για τα δύο παραδείγματα συνεκτελέσεων που παρουσιάζουμε, βλέπουμε πως οι τιμές του bandwidth είναι εξαιρετικά χαμηλές. Κάτι τέτοιο μπορεί να σημαίνει πως η υπηρεσία δεν κινδυνεύει έντονα από ενδεχόμενο κορεσμό αυτού του πόρου.

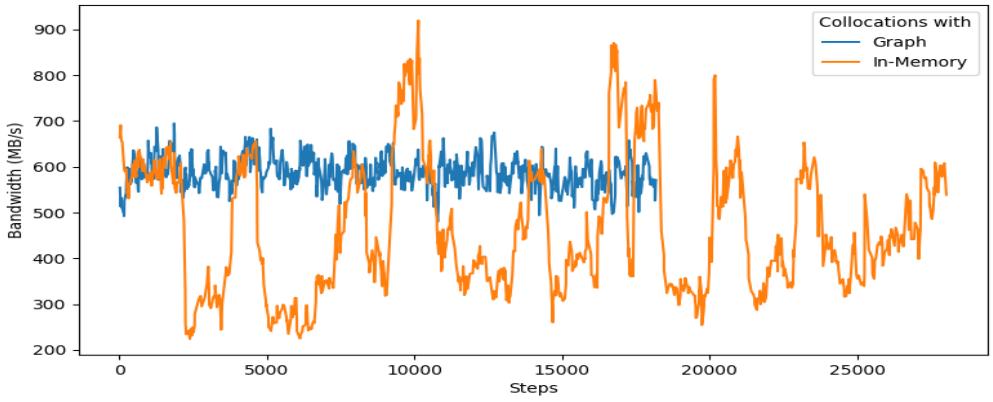
Σχήμα 2.4: Επηρεασμός της Memcached υπηρεσίας λόγω της συνεκτέλεσής της με την In-Memory Analytics. Απεικονίζονται οι μετρικές IPC, LLC χωρητικότητα, Q95 Latency και Misses ανά χίλιους κύκλους ρολογιού



Σχήμα 2.5: Συμπεριφορά του Spark job Graph Analytics όπως αυτή αποτυπώνεται από τις μετρικές IPC, LLC χωρητικότητα, Memory Bandwidth και Misses ανά χίλιους κύκλους ρολογιού



Σχήμα 2.6: Επηρεασμός της Memcached υπηρεσίας λόγω της συνεκτέλεσής της με την εφαρμογή Graph Analytics. Απεικονίζονται οι μετρικές IPC, LLC χωρητικότητα, Q95 Latency και Misses

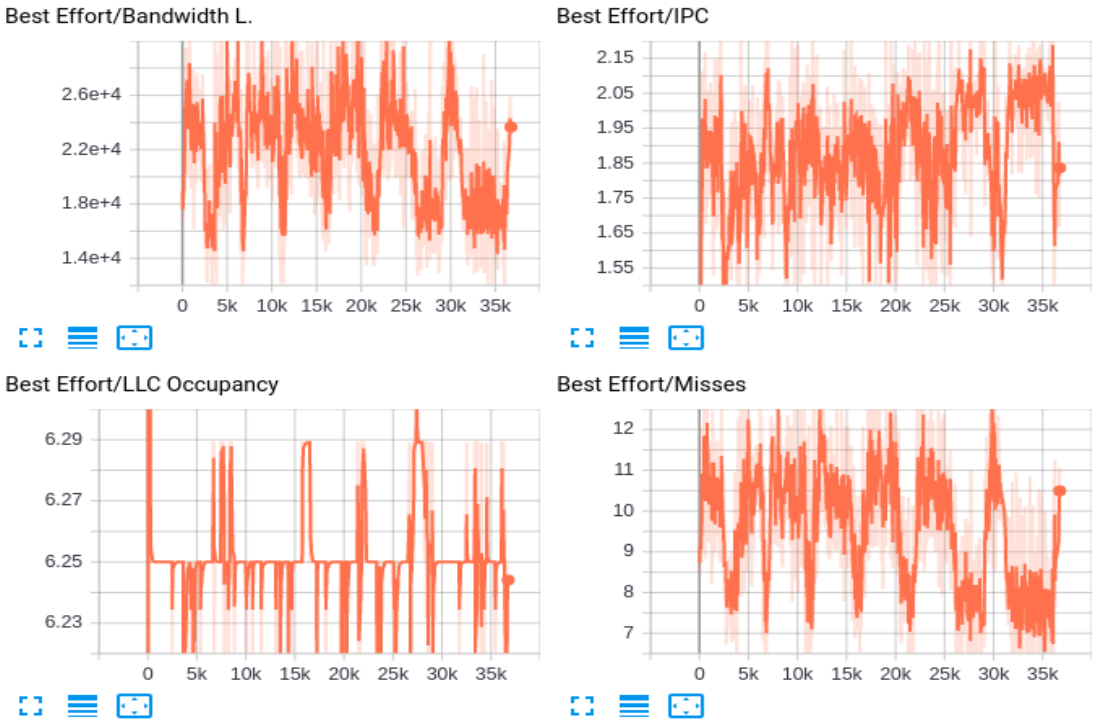


Σχήμα 2.7: Διακύμανση του bandwidth που χρησιμοποιεί η Memcached κατά την συνεκτέλεσή της με την In-Memory (πορτοκαλί γραφική) και με τη Graph Analytics (μπλε γραφική)

2.4.2 Συνεκτέλεση και στατική ανάθεση

Στο προηγούμενο σκέλος, φάνηκε ξεκάθαρα ότι η συνεκτέλεση χωρίς διαχωρισμό στη LLC επιδεινώνει σημαντικά την επίδοση της κρίσιμης υπηρεσίας. Στο σκέλος αυτό, μελετούμε το αν η διακύμανση στις μετρικές της υπηρεσίας και εν τέλει και στην επίδοση αυτής, εμφανίζεται ακόμα και στην περίπτωση που έχουμε δεσμεύσει αποκλειστικά για εκείνη ένα μεγάλο τμήμα της LLC.

Εκτελούμε και πάλι λοιπόν την υπηρεσία μαζί με το Spark Job Graphs Analytics. Όπως φανερώνει το σχήμα 2.2, η υπηρεσία καλύπτει τις ανάγκες της ακόμα και με 5 ways, παρόλα αυτά δεσμεύουμε για εκείνη τα 3/4 της LLC δηλαδή 15 ways.

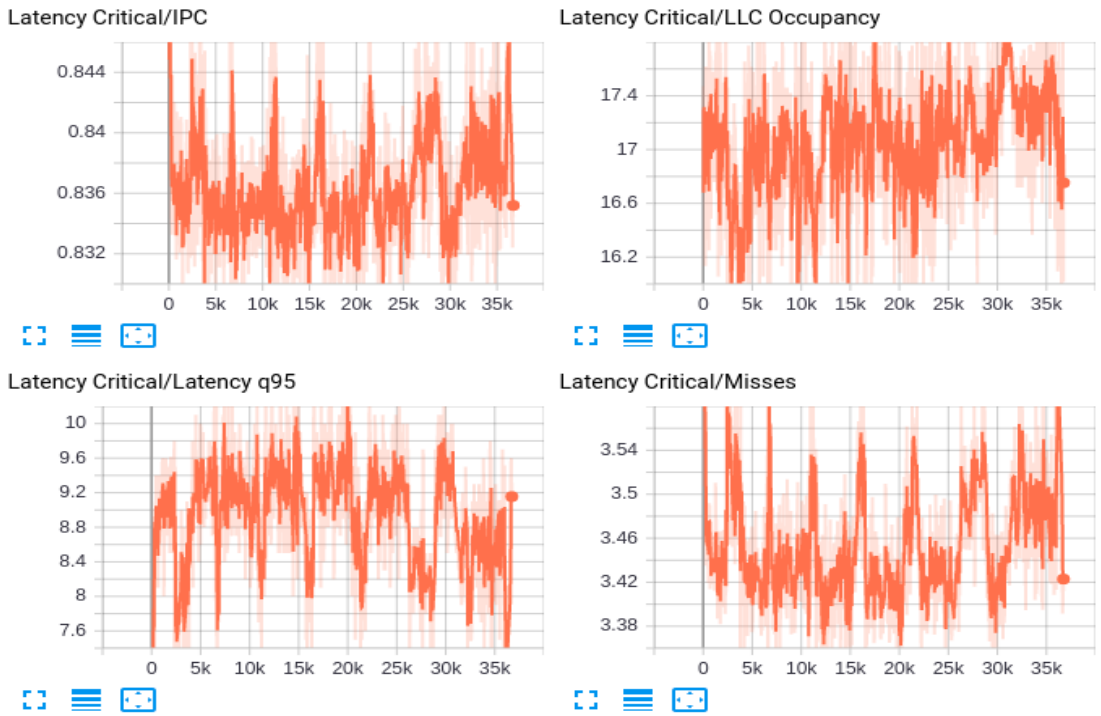


Σχήμα 2.8: Συμπεριφορά του Spark job Graph Analytics, όταν αυτό περιορίζεται μόνο στο 1/4 της LLC

Στο σχήμα 2.9 παραθέτουμε τις γνωστές πλέον μετρικές για την υπηρεσία υψηλής προτεραιότητας. Παρατηρούμε ότι παρόλο την αποκλειστική πρόσβαση σε επαρκές πλήθος ways, η συμπεριφορά της διαταράσσεται και πάλι από τις διάφορες φάσεις της εφαρμογής χαμηλής προτεραιότητας, τις οποίες επίσης παρουσιάζουμε στην εικόνα 2.8.

Σαφώς και η συνεισφορά της αποκλειστικής πρόσβασης στη LLC είναι σημαντική. Συγκρίνουμε τις εικόνες 2.6 και 2.9, που απεικονίζουν τις μετρικές για τη συνεκτέλεση της κρίσιμης υπηρεσίας με το Spark Job Graphs Analytics, όταν δεν υπάρχει έλεγχος και όταν εφαρμόζεται διαχωρισμός της LLC αντίστοιχα. Είναι εμφανές ότι η αποκλειστική πρόσβαση στα 3/4 της LLC μείωσε τα misses της κρίσιμης υπηρεσίας, μειώνοντας και το latency αυτής από τα 30 ms κατά μέση τιμή σε λιγότερα από 10 ms.

Ωστόσο οι φάσεις από τις οποίες διέρχεται η εφαρμογή χαμηλής προτεραιότητας, δημιουργούν διαφορετικό επίπεδο ανταγωνισμού, το οποίο είναι εμφανές στις μετρικές της υπηρεσίας υψηλής προτεραιότητας ακόμα και στην περίπτωση της επαρκούς δέσμευσης τμήματος της LLC.



Σχήμα 2.9: Επηρεασμός της Memcached υπηρεσίας λόγω της συνεκτέλεσής της με την εφαρμογή Graph Analytics, ακόμα και στην περίπτωση αποκλειστικής πρόσβασης της πρώτης στα 3/4 της LLC

Συνεπώς ακόμα και να ήμασταν σε θέση να γνωρίζουμε τη βέλτιστη στατική ανάθεση ways στα επιμέρους μέρη, κάτι που βέβαια απαιτεί off-line μελέτη όλων των εφαρμογών και πρακτικά είναι ανέφικτο, το αποτέλεσμα δε θα ήταν το βέλτιστο. Για παράδειγμα στην εικόνα 2.8 μεταξύ 25 και 30 χιλιάδων βημάτων τα misses της εφαρμογής μειώνονται, κάτι που επιφέρει και αντίστοιχη μείωση στο latency της υπηρεσίας (βλ. εικ όνα 2.9). Άρα σε αυτή τη φάση μπορούμε να διαχειριστούμε τον ανταγωνισμό ευκολότερα και πιθανώς να διαθέσουμε περισσότερα ways στις εφαρμογές χαμηλής προτεραιότητας. Παρουσιάζεται λοιπόν η ανάγκη για έναν εκλεπτυσμένο μηχανισμό που εκτός των άλλων, θα λαμβάνει υπόψιν του και τις διαφορετικές φάσεις των εφαρμογών βέλτιστης προσπάθειας.

Κεφάλαιο 3

Ενισχυτική Μάθηση

3.1 Εισαγωγή

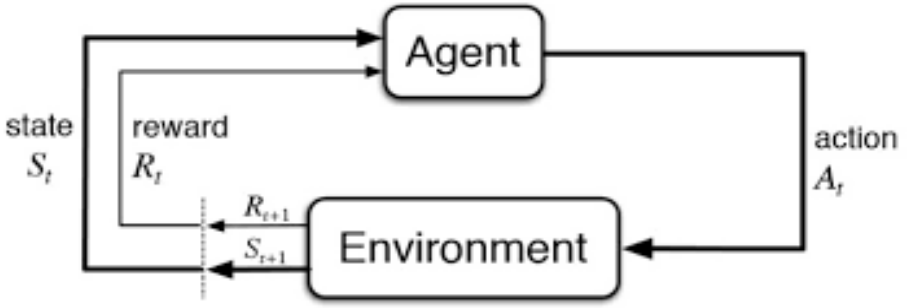
Η ενισχυτική μάθηση αποτελεί μία από τις τρεις βασικές περιοχές του επιστημονικού πεδίου που καλείται Μηχανική Μάθηση, με τις άλλες δύο να είναι η επιβλεπόμενη και η μη επιβλεπόμενη μάθηση. Η λειτουργία της διαφέρει από τις υπόλοιπες και θεωρείται ότι είναι εκείνη η τεχνική που προσεγγίζει περισσότερο τον τρόπο λειτουργίας της ανθρώπινης νοημοσύνης. Έναν τρόπο μάθησης δηλαδή, που βασίζεται στην αλληλεπίδραση του υποκειμένου με το περιβάλλον του και τη συσσώρευση εμπειρίας μέσα από τη διαδικασία αυτή. Κατά αυτό τον τρόπο, ο άνθρωπος εδραιώνει τη σχέση αιτίου-αποτελέσματος που διέπει τις δράσεις του και τις συνέπειες αυτών, ώστε χρησιμοποιώντας αργότερα τη γνώση αυτή, να είναι σε θέση, να αναπτύξει στρατηγικές, τέτοιες ώστε να επιτύχει τους σκοπούς του.

3.2 Βασικές έννοιες της Ενισχυτικής Μάθησης

Στην κλασσική περιγραφή της ενισχυτικής μάθησης ένας πράκτορας (agent), συνδεδεμένος σε ένα περιβάλλον (environment), αλληλεπιδρά με αυτό, μέσω συγκεκριμένων δράσεων (actions) που μπορεί να πραγματοποιήσει. Αυτές του οι ενέργειες έχουν σαν αποτέλεσμα τη μεταβολή της κατάστασης (state) του περιβάλλοντος και της απόδοσης σε αυτόν μιας ανταμοιβής (reward). Το προαναφερθέν μοντέλο απεικονίζεται παρακάτω, στο σχήμα 3.1.

Από τη διαδικασία αυτή ο πράκτορας προσπαθεί να αποκτήσει γνώση του περιβάλλοντος, προκειμένου να λάβει τις κατάλληλες αποφάσεις, που θα του επιτρέψουν να επιτύχει τους στόχους του. Στη συνέχεια περιγράφουμε αναλυτικότερα τις παρακάτω έννοιες:

- Το σύνολο S των καταστάσεων στις οποίες το περιβάλλον μπορεί να βρεθεί. Κάθε κατάσταση S_t αποτελείται από το σύνολο των δεδομένων που χρησιμοποιούμε για να περιγράψουμε το περιβάλλον. Για παράδειγμα, κατά την εκπαίδευση ενός πράκτορα σε ένα βιντεοπαιχνίδι Atari, ως κατάσταση θα επιλεγόταν οι εικόνες του βιντεοπαιχνιδιού αντί η εσωτερική κατάσταση του επεξεργαστή στον οποίο

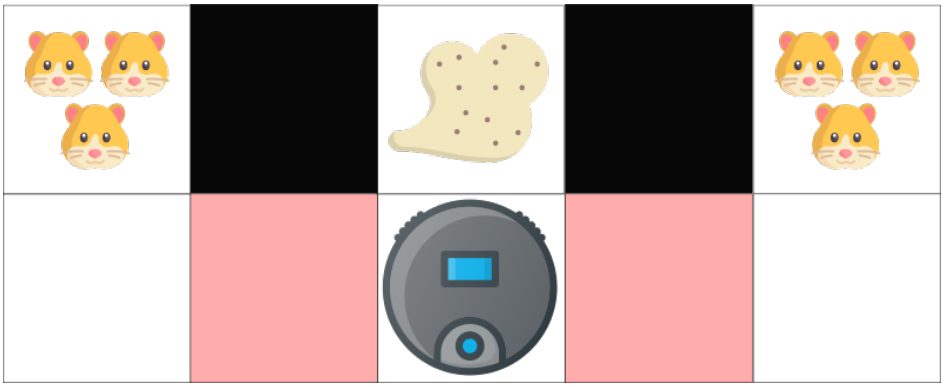


Σχήμα 3.1: Αλληλεπίδραση Πράκτορα - Περιβάλλοντος

εκτελείται. Έτσι μιλάμε για μερικώς παρατηρήσιμο περιβάλλον όπου ο πράκτορας παρατηρεί έμμεσα την κατάστασή του μέσω κάποιων μετρικών που ο σχεδιαστής επιλέγει.

Η επιλογή των μετρικών αυτών είναι κρίσιμη και αποτελεί ένα από τα λίγα σημεία παρέμβασης που έχουμε στη διαδικασία της μάθησης. Υπάρχουν περιπτώσεις όπου μπορεί να μην διαθέτουμε όλα τα απαραίτητα μεγέθη προκειμένου να περιγράψουμε σωστά κάποιο περιβάλλον. Αυτό πρακτικά σημαίνει, πως καταστάσεις στις οποίες χρειάζεται ο πράκτορας να λάβει διαφορετική απόφαση, στην αναπαράσταση που διαθέτουμε, εμφανίζονται ως ταυτόσημες.

Για παράδειγμα, στην εικόνα 3.2, όπου βλέπουμε μία ρομποτική συσκευή που προσπαθεί να φτάσει το σκονισμένο τετράγωνο στο κέντρο της εικόνας, αν η αναπαράσταση των καταστάσεων αποτελείται από τις θέσεις στις οποίες βρίσκεται τοίχος τότε τα δύο ροζ τετράγωνα αναπαρίστανται ακριβώς με τον ίδιο τρόπο δηλαδή: τοίχος επάνω και τοίχος κάτω. Ο πράκτορας όμως σε καθένα από αυτά καλείται να λάβει διαφορετική απόφαση κινούμενος στη μία περίπτωση δεξιά και στην άλλη αριστερά για να επιτύχει το στόχο του.



Σχήμα 3.2: Με ροζ χρώμα απεικονίζονται οι καταστάσεις με ταυτόσημη αναπαράσταση. Όμως ο πράκτορας ευρισκόμενος σε αυτές πρέπει να λάβει διαφορετική απόφαση

Από την άλλη μεριά, το γεγονός ότι μπορεί για κάποιο πρόβλημα να διαθέτουμε μεγάλο πλήθος μετρικών. δε σημαίνει ότι πρέπει να συμπεριληφθούν όλες στην κατάσταση, καθώς πλεονάζουσα πληροφορία μπορεί να δυσκολέψει τον πράκτορα να διακρίνει την ουσία και κατά συνέπεια να καθυστερήσει την διαδικασία της εκπαίδευσης.

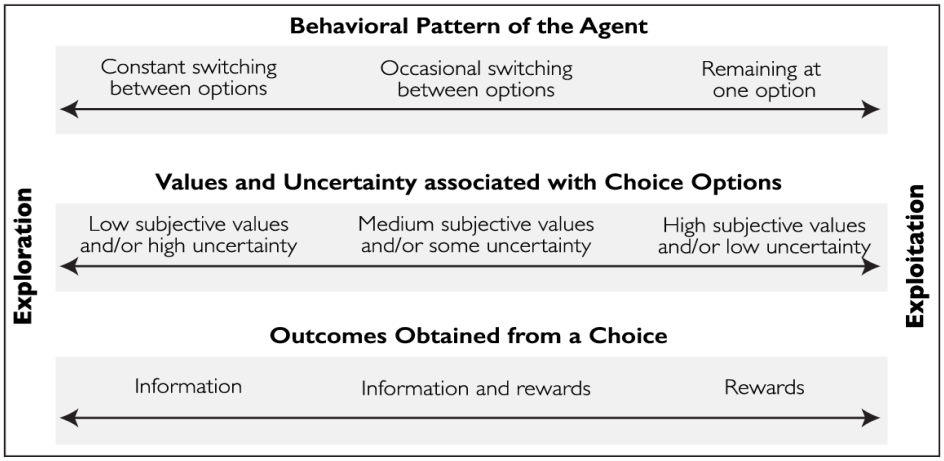
- Το σύνολο A των διαθέσιμων δράσεων που μπορεί να εκτελέσει ο πράκτορας στο περιβάλλον. Για κάποια προβλήματα μπορεί το σύνολο αυτό να μην είναι το ίδιο για όλες τις πιθανές καταστάσεις. Η δράση που λαμβάνεται στο βήμα t επηρεάζει την κατάσταση στο βήμα t+1.

- Τη συνάρτηση ανταμοιβής (reward function) που αποδίδει τις ανταμοιβές R_t σε κάθε βήμα t. Η συνάρτηση ανταμοιβής κατέχει εξέχουσα σημασία στη διαδικασίας της μάθησης, καθώς είναι η ανταμοιβή αυτή που προσπαθεί να μεγιστοποιήσει ο πράκτορας και άρα μόνο αυτή του προσδιορίζει το αν κινείται σωστά ή λανθασμένα στη προσπάθειά του να λύσει το πρόβλημα. Αν λοιπόν η συνάρτηση ανταμοιβής δεν έχει επιλεχθεί προσεκτικά μπορεί ο πράκτορας να μάθει μια εντελώς διαφορετική συμπεριφορά από την αναμενόμενη. Πολλά τέτοια φαινόμενα εντοπίζονται σε διάφορες ιστορικές περιόδους και αφορούν κίνητρα που προσπάθησε να δώσει κάποια κεντρική αρχή για την επίτευξη ενός στόχου.

  Παραδείγματος χάριν, αν για τη μείωση του πληθυσμού ενός επαχθούς οργανισμού σε ένα φυσικό περιβάλλον δίνεται αμοιβή για κάθε άτομο αυτού του οργανισμού που επιστρέφεται νεκρό, τότε οι διάφοροι πράκτορες (άνθρωποι) θα συνειδητοποιήσουν σύντομα πως η καλύτερη στρατηγική για να μεγιστοποιήσουν την ανταμοιβή τους θα είναι να εκτρέψουν άτομα του εν λόγω οργανισμού σε μεγάλους αριθμούς. Προφανώς κάτι τέτοιο οδηγεί στο ακριβώς ανάποδο αποτέλεσμα από το προσδοκώμενο ([9]).

- Το δίλημμα της "εξερεύνησης - εκμετάλλευσης". Όταν ο πράκτορας ξεκινά να αλληλεπιδρά με το περιβάλλον δε γνωρίζει τίποτα για αυτό και κατά συνέπεια δρα στην τύχη. Καθώς όμως συνεχίζεται αυτή η διαδικασία αρχίζει και συσσωρεύει κάποια γνώση για το περιβάλλον και την οποία είναι σε θέση να εκμεταλλευτεί ώστε να λάβει τις αποφάσεις του. Σε αυτό το σημείο υπεισέρχεται το ερώτημα σε ποιο βαθμό εκμεταλλεύεται την υφιστάμενη γνώση και σε ποιο συνεχίζει την εξερεύνηση;

  Από τη μία η άμεση εκμετάλλευση μπορεί να οδηγήσει σε μη βέλτιστες αποφάσεις καθώς είναι πιθανόν να υπάρχουν καταστάσεις του περιβάλλοντος που δεν έχουμε εξερευνήσει ακόμα και οι οποίες να οδηγούν σε καλύτερα αποτελέσματα. Από την άλλη η συνέχιση της εξερεύνησης σημαίνει πως θυσιάζουμε ένα ποσό ανταμοιβής που μπορούμε άμεσα να λάβουμε. Για την αποτελεσματική αντιμετώπιση αυτού του διλήμματος πολλές διαφορετικές μέθοδοι έχουν προταθεί στη βιβλιογραφία



Σχήμα 3.3: Επίδραση του διλήμματος εξερεύνησης-εκμετάλλευσης στη συμπεριφορά του πράκτορα, στις αξίες των καταστάσεων που εκτιμά και στα αποτελέσματα που συλλέγει.

([10]).

3.3 Μοντελοποίηση της Ενισχυτικής Μάθησης

Ένα πρόβλημα ενισχυτικής μάθησης μπορεί να περιγραφεί σαν μια Μαρκοβιανή Διαδικασία Αποφάσεων (Markov Decision Process - MDP). Η Μαρκοβιανή Διαδικασία Αποφάσεων είναι μια επέκταση της Αλυσίδας Markov όπου έχουν προστεθεί επιπλέον η δυνατότητα λήψης αποφάσεων και η απόδοση ανταμοιβών. Ο ορισμός της δίνεται παρακάτω:

Ορισμός 1 (Μαρκοβιανή Διαδικασία Αποφάσεων): *Η Μαρκοβιανή Διαδικασία αποφάσεων ορίζεται ως ένα σύνολο <S, A, P, R, γ> όπου:*

1. *S ένα πεπερασμένο σύνολο από καταστάσεις*
2. *A είναι ένα πεπερασμένο σύνολο από δράσεις*
3. *P είναι ένας πίνακας μεταβάσεων πιθανοτήτων,* $P^a_{ss} = Pr[S_t+1 = s'|S_t = s, A_t = a]$
4. *R είναι μια συνάρτηση ανταμοιβής,* $R^a_s = E[R_{t+1}|S_t = s, A_t = a]$
5. *γ είναι ένας παράγοντας έκπτωσης, όπου* $\gamma\epsilon[0, 1]$

Σαν επέκταση μιας Μαρκοβιανής διαδικασίας, οι καταστάσεις μιας MDP πληρούν τη μαρκοβιανή ιδιότητα κάτι που σημαίνει πως προκειμένου να λάβουμε κάποια μελλοντική απόφαση μπορούμε να βασιστούμε μόνο στην τωρινή κατάσταση του συστήματος, δηλαδή:

Ορισμός 2 (Μαρκοβιανή Ιδιότητα): *Το μέλλον είναι ανεξάρτητο του παρελθόντος δεδομένου του παρόντος.*

$$Pr[S_{t+1}|S_t] = Pr[S_{t+1}|S_1, ..., S_t] \tag{3.1}$$

Όπως αναφέραμε προηγουμένως, στόχος του πράκτορα είναι να μεγιστοποιήσει τη συνολική ανταμοιβή που λαμβάνει. Όμως η ενδεχόμενη αβεβαιότητα που μπορεί να υπάρχει στο περιβάλλον μας κάνει να αμφιβάλλουμε για τις μελλοντικές ανταμοιβές. Εκεί υπεισέρχεται ο παράγοντας γ του ορισμού, που δίνει μεγαλύτερη έμφαση στις πιο άμεσες επιβραβεύσεις. Έτσι λοιπόν, το μέγεθος που μας ενδιαφέρει τελικά είναι:

Ορισμός 3 (Συνάρτηση Επιστροφής): *Η συνάρτηση επιστροφής G_t ορίζεται ως το άθροισμα των μελλοντικών ανταμοιβών πολλαπλασιασμένες με το συντελεστή αβεβαιότητας γ.*

$$G_t = \sum_{n=0}^{\infty} \gamma^n R_{t+n+1} \tag{3.2}$$

Όσο αφορά τον τρόπο λήψης των αποφάσεων ορίζουμε την έννοια της πολιτικής:

Ορισμός 4 (Πολιτική): *Μια πολιτική π είναι μια κατανομή πάνω στις δράσεις δοθέντων των καταστάσεων.*

$$\pi(a|s) = P[A_t = a|S_t = s] \tag{3.3}$$

Η πολιτική π προσδιορίζει τις πιθανότητες λήψης μιας δράσης ανάλογα την κατάσταση που βρίσκεται ο πράκτορας και έτσι καθορίζει πλήρως τη συμπεριφορά του.

Σημαντικό ρόλο έχει η συνάρτηση τιμών-κατάστασης $V^{\pi}(s)$ (state-value function), η οποία με απλά λόγια είναι ένα μέτρο του πόσο καλό είναι να βρισκόμαστε στην κατάσταση s:

Ορισμός 5 (Συνάρτηση Τιμής-Κατάστασης): *Η συνάρτηση τιμής-κατάστασης $V_{\pi}(s)$ μιας MDP είναι η αναμενόμενη τιμή των επιστροφών ξεκινώντας από την κατάσταση s και ακολουθώντας την πολιτική π.*

$$V_{\pi}(s) = E_{\pi}[G_t|S_t = s] \tag{3.4}$$

Πέρα όμως από την αξία της ίδιας της κατάστασης μας ενδιαφέρει και η αξία της κάθε δράσης, γεγονός που θα προσδιορίσει και την απόφασή μας για το ποια δράση θα εκτελέσουμε. Ορίζουμε λοιπόν τη συνάρτηση τιμής-δράσης (action-value function).

Ορισμός 6 (Συνάρτηση Τιμής-Δράσης): *Η συνάρτηση τιμής-δράσης $Q_{\pi}(s, a)$ ορίζεται ως η αναμενόμενη τιμή των επιστροφών ξεκινώντας από την κατάσταση s, παίρνοντας την δράση a και ακολουθώντας έπειτα την πολιτική π.*

$$Q_{\pi}(s, a) = E_{\pi}[G_t|S_t = s, A_t = a] \tag{3.5}$$

Αυτό που πραγματικά μας ενδιαφέρει στην ενισχυτική μάθηση είναι να βρούμε τις βέλτιστες συναρτήσεις $V_*(s), Q_*(s,a)$, οπότε και πρακτικά θεωρούμε ότι έχουμε "λύσει" την MDP. Άπαξ και γνωρίζουμε τη βέλτιστη συνάρτηση τιμής-δράσης μπορούμε εύκολα να έχουμε και μία βέλτιστη πολιτική αποφάσεων, επιλέγοντας για την κατάσταση στην οποία βρισκόμαστε τη δράση με τη μεγαλύτερη τιμή.

Στη περίπτωση των Μαρκοβιανών Διαδικασιών Απόφασης ισχύει το παρακάτω θεώρημα:

Θεώρημα 1: *Για κάθε MDP:*

- *Υπάρχει βέλτιστη πολιτική* π_* *ώστε* $\pi_* \geq \pi, \forall \pi$
- *Κάθε βέλτιστη πολιτική επιτυγχάνει τη βέλτιστη state-value function,* $V\pi_*(s) = V_*(s)$
- *Κάθε βέλτιστη πολιτική επιτυγχάνει τη βέλτιστη action-value function,* $Q\pi_*(s,a) = Q_*(s,a)$

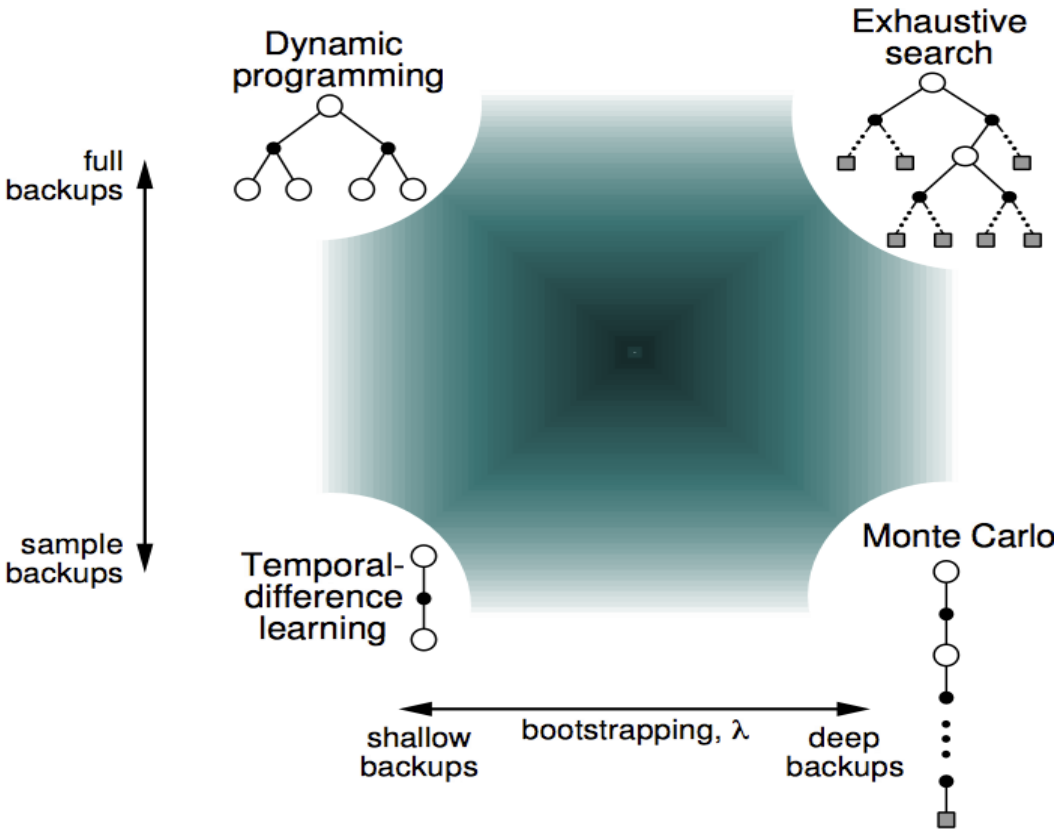
Χρησιμοποιώντας την εξίσωση του Bellman [11] μπορούμε να αποσυνθέσουμε τις παραπάνω εξισώσεις χωρίζοντάς τες σε δύο τμήματα, το κέρδος που λαμβάνουμε άμεσα και την τιμή των συναρτήσεων από εκείνο το σημείο και έπειτα.

$$V_\pi(s) = E_\pi[R_{t+1} + \gamma V_\pi(s_{t+1}) | S_t = s] \tag{3.6}$$

$$Q_\pi(s,a) = E_\pi[R_{t+1} + \gamma Q_\pi(s_{t+1}, A_{t+1}) | S_t = s, A_t = a] \tag{3.7}$$

Αρκετές επαναληπτικές μέθοδοι υπάρχουν για την εύρεση των παραπάνω βέλτιστων συναρτήσεων. Όταν ένα μοντέλο είναι διαθέσιμο για το πρόβλημα μας, δηλαδή γνωρίζουμε τις πιθανότητες μετάβασης P_{ss} μεταξύ των διάφορων καταστάσεων τότε μπορούμε να εφαρμόσουμε και μεθόδους δυναμικού προγραμματισμού.

Παρόλα αυτά είναι πολλές οι περιπτώσεις, ιδιαίτερα εκείνες στον πραγματικό κόσμο, όπου δεν έχουμε διαθέσιμο ένα πλήρες μοντέλο του προβλήματος, είτε το πλήθος των καταστάσεων είναι τόσο μεγάλο που καθιστά αδύνατη την εφαρμογή μεθόδων δυναμικού προγραμματισμού. Μια διαφορετική προσέγγιση για αυτές τις περιπτώσεις είναι η χρήση της μεθόδου Monte Carlo. Η μέθοδος Monte Carlo αλληλεπιδρά με το περιβάλλον και χρησιμοποιεί τον εμπειρικό μέσο για την εκτίμηση των value functions. Σε αντίθεση όμως με το δυναμικό προγραμματισμό, δε χρησιμοποιεί bootstraping κάτι που σημαίνει ότι περιμένει την ολοκλήρωση ενός στιγμιότυπου του προβλήματος (επεισόδιο), προκειμένου να αποδώσει αναδρομικά τις ανταμοιβές σε κάθε κατάσταση που έχει επισκεφτεί κατά τη διάρκεια αυτού.



Σχήμα 3.4: Συγκεντρωτική απεικόνιση των μεθόδων ενισχυτικής μάθησης

3.4 Q-Learning

Οι μέθοδοι ενισχυτικής μάθησης καλούνται να αντιμετωπίσουν κυρίως προβλήματα όπου το MDP είναι άγνωστο αλλά υπάρχει δυνατότητα αλληλεπίδρασης με το περιβάλλον και συλλογής γνώσης από αυτό. Ένας από τους πιο σημαντικούς αλγορίθμους του πεδίου αυτού ονομάζεται Q-learning.

3.4.1 Μάθηση Χρονικών Διαφορών

Ο αλγόριθμος αυτός χρησιμοποιεί τη μάθηση χρονικών διαφορών (Temporal Difference learning) για να ανανεώσει την εκτίμησή του για την action-value function. Καίριο σημείο της μεθόδου αυτής είναι ότι, η ανανέωση των εκτιμήσεων για την action-value συνάρτηση γίνεται εν μέρη βάσει των εκτιμήσεων που ήδη έχει μάθει ο πράκτορας, μέχρι εκείνη τη χρονική στιγμή(bootstrapping). Δεν απαιτεί λοιπόν την ύπαρξη ενός τέλους στο πρόβλημα προκειμένου να μπορέσει να αναπροσαρμόσει τις εκτιμήσεις του για τις value function όπως η μέθοδος Monte Carlo. Μπορεί έτσι να χρησιμοποιηθεί και σε περιβάλλοντα όπου δεν υπάρχει κάποια τερματική κατάσταση.

Αναλυτικότερα για κάθε απόφαση που παίρνει ο πράκτορας λαμβάνει μια ανταμοιβή και τη νέα κατάσταση στην οποία μετέβη το περιβάλλον. Υπολογίζουμε το Temporal Difference Target ως το άθροισμα της ανταμοιβής που λάβαμε και της εκτίμησής μας

για την αξία της νέας κατάστασης στην οποία μεταβήκαμε. Τέλος μετατοπίζουμε την τωρινή μας εκτίμηση για την αξία της κατάστασης S_t στην κατεύθυνση της διαφοράς του TD target από την τωρινή εκτίμηση.

$$V(s_t) = V(s_t) + a[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{3.8}$$

όπου α το βήμα μάθησης, που καθορίζει σε ποιο βαθμό λαμβάνουμε υπόψιν τις νέες ανανεώσεις.

3.4.2 Ε-Άπληστη Πολιτική

Όσο αφορά το δίλημμα της εκμετάλλευσης εξερεύνησης που αναφέραμε προηγουμένως, η μέθοδος Q-Learning χρησιμοποιεί συνήθως την ε-άπληστη πολιτική (ε-greedy). Σύμφωνα με αυτή σε κάθε βήμα ο αλγόριθμος με μια πιθανότητα ε λαμβάνει αποφάσεις τυχαία, ενώ διαφορετικά δρα άπληστα επιλέγοντας την δράση εκείνη που εκτιμά ότι προσφέρει τη μεγαλύτερη ανταμοιβή.

Με βάση τα παραπάνω είμαστε σε θέση να διατυπώσουμε πλέον τον αλγόριθμο Q-Learning:

Αλγόριθμος 1: Q-Learning

Initialize Q array with zero values
for *episode* $\leftarrow 1,\ M$ **do**
  Get the current state of the environment s_t
  for $t \leftarrow 1,\ T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = max_a Q^*(s_t, a)$
    Execute action a_t and observe reward r_t and the next state s_{t+1}
    Perform update rule:
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma max_a Q(s_{t+1}, a_t) - Q(s_t, a_t)]$
    Set $s_t = s_{t+1}$
  end
end

Ο αλγόριθμος αυτός χαρακτηρίζεται και ως Off-Policy καθώς χρησιμοποιεί διαφορετική πολιτική για τις νέες δράσεις που πραγματοποιεί (ε-greedy) και διαφορετική πολιτική προκειμένου να ανανεώσει την εκτίμησή του για τη state-value function (greedy).

Επίσης, παρόλο που συνηθέστερα ο αλγόριθμος διατυπώνεται σε επεισοδιακή μορφή, δηλαδή θεωρούμε ότι το περιβάλλον φτάνει σε μια τερματική κατάσταση από την οποία οδηγούμαστε και πάλι σε κάποια αρχική, κάτι τέτοιο δεν είναι απαραίτητο και κάλλιστα η αλληλουχία των αποφάσεων μας μπορεί να είναι συνεχής.

Συμπερασματικά, αν και εξαιρετικής σημασίας ο παραπάνω αλγόριθμος υποθέτει ένα πεπερασμένο πλήθος καταστάσεων για την αναπαράσταση του περιβάλλοντος, οι οποίες αναπαρίσταται με έναν πίνακα στη μνήμη. Γίνεται εύκολα αντιληπτό ότι κάτι τέτοιο είναι αδύνατο από πρακτικής άποψης για μεγάλους χώρους καταστάσεων. Επιπλέον η συγκεκριμένη λογική δεν εκμεταλλεύεται τις δυνατότητες γενίκευσης που υπάρχουν, επιχειρώντας μια σημειακή εκτίμηση της κάθε κατάστασης. Στην επόμενη ενότητα θα δούμε πως μπορούμε να επεκτείνουμε αυτή την ιδέα εκμεταλλευόμενοι τα πλεονεκτήματα που μας προσφέρει η προσέγγιση συναρτήσεων (function approximation) με τεχνικές όπως είναι τα νευρωνικά δίκτυα.

3.5 Βαθιά Ενισχυτική Μάθηση

Η συνάντηση του πεδίου της Ενισχυτικής Μάθησης με τις πρόσφατες εξελίξεις στην εκπαίδευση τεχνητών νευρωνικών δικτύων, οδήγησε σε μια σειρά από καινοτομίες στο πεδίο της τεχνητής νοημοσύνης, για τις οποίες επικράτησε ο όρος Βαθιά Ενισχυτική Μάθηση. Η αρχή έγινε από την εταιρία DeepMind το 2013, όταν αυτή κατόρθωσε για πρώτη φορά να χρησιμοποιήσει ένα μοντέλο βαθιάς μηχανικής μάθησης, για να εκπαιδεύσει έναν πράκτορα ενισχυτικής μάθησης, στο να παίζει Atari βιντεοπαιχνίδια, σε επίπεδο ανώτερο του ανθρώπου ([12]). Στη συνέχεια και πάλι η DeepMind εξέπληξε την ανθρωπότητα το 2016 όταν το σύστημα τεχνητής νοημοσύνης, AlphaGo, που είχε αναπτύξει κατόρθωσε να νικήσει σε μια σειρά αναμετρήσεων τον παγκόσμιο πρωταθλητή στο αρχαιότερο επιτραπέζιο παιχνίδι, Go ([13]). Το Go θεωρούνταν εξαιρετικά δύσκολο παιχνίδι για τους υπολογιστές, καθώς διαθέτει έναν τεράστιο χώρο καταστάσεων, όπου παραδοσιακές μέθοδοι τεχνητής νοημοσύνης, όπως αυτές που χρησιμοποιήθηκαν για τη νίκη έναντι πρωταθλητών στο σκάκι, δεν επαρκούσαν.

Η DeepMind συνέχισε την έρευνα στο πεδίο, αναπτύσσοντας ακόμα πιο εκλεπτυσμένα συστήματα που πλέον είχαν τη δυνατότητα να εκπαιδευτούν μόνα τους στο να μάθουν να κερδίζουν πολλά διαφορετικά παιχνίδια είτε επρόκειτο για Atari Games, είτε για επιτραπέζια. Και αν τα διάφορα παιχνίδια έδωσαν τροφή αρχικά στην ανάπτυξη αυτού του επιστημονικού πεδίου, στο σήμερα γίνονται πολλές προσπάθειες για την αποτελεσματική εφαρμογή του σε πρακτικά προβλήματα.

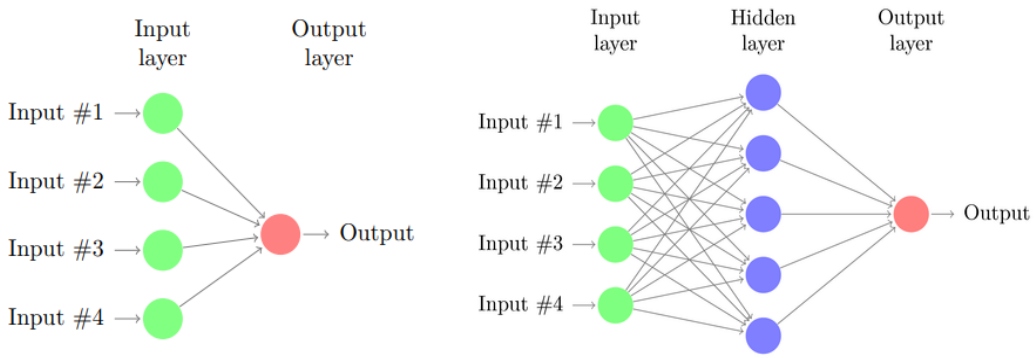
3.5.1 Τεχνητά Νευρωνικά Δίκτυα

Τα νευρωνικά δίκτυα είναι υπολογιστικά μοντέλα εμπνευσμένα από τη λειτουργία των βιολογικών νευρώνων των οργανισμών. Χρησιμοποιούνται ευρέως σε προβλήματα αναγνώρισης προτύπων, καθώς αποτελούν μια γενική μέθοδο προσέγγισης συναρτήσεων. Αν και η ανάπτυξη τεχνητών νευρωνικών δικτύων ξεκίνησε στα μέσα του περασμένου αιώνα, το πεδίο για μεγάλο διάστημα παρέμεινε στάσιμο, μη γνωρίζοντας ιδιαίτερη πρόοδο. Η κατάσταση αυτή άλλαξε δραματικά τα τελευταία χρόνια με τα νευρωνικά δίκτυα

να πετυχαίνουν εντυπωσιακά αποτελέσματα, σε τομείς όπως η αναγνώριση εικόνων, η αναγνώριση φωνής και η παραγωγή λόγου. Σε αυτή την αλλαγή καθοριστικότατη υπήρξε η ισχυροποίηση των υπολογιστικών συστημάτων. μιας και επέτρεψε τη χρήση βαθύτερων, δηλαδή πιο σύνθετων, νευρωνικών δικτύων.

Χαρακτηριστικά Νευρωνικών Δικτύων

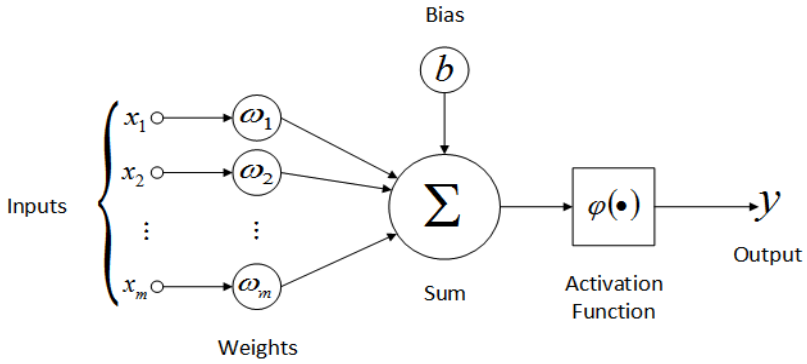
Ένα νευρωνικό δίκτυο λοιπόν, οργανώνεται σε επίπεδα τα οποία αποτελούνται από επιμέρους μονάδες, που καλούνται νευρώνες. Ο αριθμός των επιπέδων, των νευρώνων αλλά και ο τρόπος διασύνδεσης αυτών, καθορίζουν την αρχιτεκτονική του δικτύου.



Σχήμα 3.5: Απλές αρχιτεκτονικές Τεχνητών Νευρωνικών Δικτύων

Κάθε νευρωνικό δίκτυο διαθέτει σίγουρα ένα επίπεδο εισόδου, ένα επίπεδο εξόδου ενώ το πλήθος των κρυφών επιπέδων καθορίζεται ανάλογα το πρόβλημα. Όπως φαίνεται στην εικόνα 3.5 το επίπεδο εισόδου δέχεται την υπολογιστική αναπαράσταση των δεδομένων, ενώ το τελευταίο επίπεδο παράγει ένα σήμα εξόδου.

Ένας τεχνητός νευρώνας ουσιαστικά αποτελείται από έναν αθροιστή που υπολογίζει τον γραμμικό συνδυασμό των εισόδων του x_i με τα βάρη αυτού w_i και προσθέτει έναν σταθερό όρο b (bias). Αν η διαδικασία ολοκληρωνόταν εδώ θα μιλούσαμε για ένα γραμμικό μοντέλο καθώς οι έξοδοι των νευρώνων θα μπορούσαν να συνδυαστούν και έτσι να προκύψει ένας παράγοντας βάρους για κάθε είσοδο του δικτύου. Όπως είπαμε τα νευρωνικά δίκτυα αποτελούν μια γενική μέθοδο προσέγγισης συναρτήσεων. Προκειμένου να συμβαίνει αυτό εισάγεται μια μη γραμμικότητα στην έξοδο κάθε νευρώνα όπως φαίνεται και από την εικόνα 3.6. Ο όρος αυτός ονομάζεται συνάρτηση ενεργοποίησης και αποφασίζει για την ενεργοποίηση ή μη του νευρώνα. Διάφορες τέτοιες συναρτήσεις παίζουν αυτό το ρόλο με συνηθέστερες τη σιγμοειδή συνάρτηση, την υπερβολική εφαπτομένη και πιο πρόσφατα την ReLU (Rectified Linear Unit) που εκφράζεται από τον τύπο: $f(x) = x^+ = max(0, x)$.



Σχήμα 3.6: Τεχνητός Νευρώνας

Εκπαίδευση Νευρωνικών Δικτύων

Ένα νευρωνικό δίκτυο αποτελεί μια συνάρτηση εισόδου εξόδου: $\hat{y} = f(x;\theta)$, όπου θ τα βάρη του δικτύου. Στόχος της διαδικασίας της μάθησης είναι ο συσχετισμός των δεδομένων εισόδου x, με τις επιθυμητές εξόδους y. Θεωρούμε έτσι λοιπόν μια συνάρτηση απώλειας (loss function) $L(\hat{y}, y) = L(f(x;\theta), y)$, μεταξύ των πραγματικών τιμών y και των προβλέψεων $\hat{y}$ του δικτύου. Η συνάρτηση αυτή παίρνει την ελάχιστη τιμή της, όταν οι προβλέψεις μας είναι σωστές. Αναζητούμε λοιπόν τα βάρη εκείνα θ_* του δικτύου, που ελαχιστοποιούν τη συνάρτηση απώλειας δηλαδή :

$$\theta_* = \underset{\theta}{\operatorname{argmin}}\, L(\theta) \tag{3.9}$$

Όπως γίνεται αντιληπτό, πλέον το ζητούμενο είναι η προσαρμογή των βαρών του μοντέλου. Για το σκοπό αυτό χρησιμοποιείται η μέθοδος οπίσθιας διάδοσης (backpropagation), η οποία υπολογίζει διαδοχικά και ξεκινώντας από το τελευταίο επίπεδο, την παράγωγο της συνάρτησης απωλειών ως προς τα βάρη του δικτύου, με χρήση του κανόνα της αλυσίδας.

Ο αλγόριθμος αυτός αναφέρεται στον τρόπο υπολογισμού των παραγώγων των βαρών και όχι στο πως χρησιμοποιούνται αυτές για τη βελτιστοποίηση. Συνηθέστερα για το σκοπό αυτό αξιοποιείται ο αλγόριθμος καθόδου κλίσης (gradient descent) που ανανεώνει τα βάρη στην αντίθετη κατεύθυνση της παραγώγου, δηλαδή :

$$\theta = \theta - \eta \nabla_\theta L(\theta) \tag{3.10}$$

Πολλές πιο εκλεπτυσμένες παραλλαγές αυτής της μεθόδου έχουν προταθεί στη βιβλιογραφία, όπως είναι ο optimizer Adam ([14]).

3.5.2 DQN

Η προσαρμογή της μάθησης των νευρωνικών δικτύων στις ανάγκες της ενισχυτικής μάθησης δεν ήταν εύκολη υπόθεση καθώς αυτή εμφανίζει μια σειρά από ιδιαιτερότητες.

Όπως αναλύεται στη δημοσίευση [12] οι επιβραβεύσεις που λαμβάνουμε μπορεί να είναι σποραδικές, θορυβώδεις και καθυστερημένες. Ακόμα στην ενισχυτική μάθηση έχουμε στη διάθεσή μας ακολουθίες από ισχυρά συσχετισμένες καταστάσεις, τη στιγμή που η βαθιά μάθηση υποθέτει ανεξάρτητα και ομοιόμορφα κατανεμημένα δεδομένα. Τέλος η ίδια η κατανομή των δεδομένων αλλάζει, καθώς ο πράκτορας μαθαίνει νέες συμπεριφορές, ενώ η βαθιά μάθηση θεωρεί μια σταθερή υποκείμενη κατανομή.

Η συνεισφορά της εργασίας [12] έγκειται στη δημιουργία μιας παραλλαγής της μεθόδου Q-Learning που μπορεί αποτελεσματικά να χρησιμοποιήσει και να εκπαιδεύσει ένα νευρωνικό δίκτυο το οποίο προσεγγίζει τη βέλτιστη συνάρτηση action-value Q.

$$Q(s, a; \theta) \approx Q^*(s, a) \tag{3.11}$$

Το νευρωνικό αυτό, το ονομάζουμε Q-network $Q(s, a; \theta)$, όπου θ τα βάρη αυτού. Όπως συμβαίνει στην εκπαίδευση νευρωνικών δικτύων στόχος είναι η ελαχιστοποίηση μιας συνάρτησης απωλειών. Στην περίπτωση του DQN αυτή ορίζεται ως:

$$L_i(\theta_i) = E_\pi[(y_i - Q(s, a; \theta))^2] \tag{3.12}$$

όπου y_i είναι ο στόχος (target) κάθε επανάληψης και ο οποίος μπορεί να διαφέρει ανάλογα με το ποια μέθοδο μάθησης χρησιμοποιούμε.

Αλγόριθμος 2: DQN

Initialize Q-network with random weights θ
Initialize replay memory D to capacity N
for *episode* $\leftarrow 1,\ M$ **do**
  Get the current state of the environment s_t
  for $t \leftarrow 1,\ T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = max_a Q^*(s_t, a; \theta)$
    Execute action a_t and observe reward r_t and the next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1})
    Set $s_t = s_{t+1}$
    Sample random minibatch of k transitions (s_j, a_j, r_j, s_{j+1}) from D
    Set $y_j = \begin{cases} r_j & \text{for terminal states} \\ r_j + \gamma max_{a'} Q^*(s_{j+1}, a'; \theta) & \text{for non terminal states} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$
  end
end

Ο αλγόριθμος Q-Learning, όπως αναφέραμε στο 3.4.1, χρησιμοποιεί τη μάθηση χρονικών διαφορών, οπότε ο στόχος από τον οποίο υπολογίζουμε τη διαφορά με την εκτίμησή μας είναι $y_i = E_{s' \sim \epsilon}[R + \gamma \max_{a'} Q^*(s', a'; \theta_{i-1})|s, a]$. Οι παράμετροι του στόχου θ_{i-1} παραμένουν σταθερές καθώς βελτιστοποιήσουμε τη συνάρτηση απωλειών.

Το νευρωνικό δίκτυο Q λαμβάνει σαν είσοδο καταστάσεις και στην έξοδό του παίρνουμε τις εκτιμώμενες Q τιμές για κάθε πιθανή δράση. Ο αλγόριθμος DQN για την εξερεύνηση του περιβάλλοντος, ακολουθεί την e-greedy πολιτική που αναλύσαμε στο κεφάλαιο 3.4.2. Ακόμα, προκειμένου να αντιμετωπιστεί το ζήτημα της συσχέτισης μεταξύ διαδοχικών δειγμάτων, αυτά αποθηκεύονται σε μια προσωρινή μνήμη από την οποία και επιλέγεται τυχαία σε κάθε επανάληψη, ένα πλήθος. Η εκπαίδευση του δικτύου συμβαίνει επίσης σε κάθε επανάληψη και βασίζεται στο batch παρελθοντικών δειγμάτων που επιλέχθηκε από τη μνήμη. Αυτός ο μηχανισμός ονομάζεται αναπαραγωγή εμπειρίας (experience replay) και περιέχεται στον αλγόριθμο 2, που παρουσιάζουμε αναλυτικά.

3.5.3 Double DQN

Από τη στιγμή που ήρθε στο προσκήνιο ο αλγόριθμος DQN πολλές βελτιστοποιήσεις αυτού έχουν προταθεί. Παρουσιάζουμε στη συνέχεια κάποιες από αυτές που θεωρούνται στη βιβλιογραφία ως οι πιο σημαντικές και τις οποίες χρησιμοποιήσαμε και στην παρούσα διπλωματική εργασία, για την αντιμετώπιση του προβλήματός μας.

Αλγόριθμος 3: Double DQN

θ: initial network parameters, θ^- copy of θ
D: empty replay buffer, N: target network replacement freq
for *episode* $\leftarrow 1,\ M$ **do**
  Get the current state of the environment s_t
  for $t \leftarrow 1,\ T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = max_a Q(s_t, a; \theta)$
    Execute action a_t and observe reward r_t and the next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1})
    Set $s_t = s_{t+1}$
    Sample random minibatch of k transitions (s_j, a_j, r_j, s_{j+1}) from D
    Define $a^{max}(s_{j+1}; \theta) = argmax_{a'} Q(s_{j+1}, a'; \theta)$
    $$y_j = \begin{cases} r_j & \text{for terminal states} \\ r_j + \gamma Q(s_{j+1}, a^{max}(s_{j+1}; \theta); \theta)^- & \text{for non terminal states} \end{cases}$$
    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$
    Replace target parameters $\theta^- \leftarrow \theta$ every N steps
  end
end

Ο αλγόριθμος Q-Learning υπερεκτιμά την τιμή της συνάρτησης Q κάτι που οφείλεται στη χρήση της ίδιας συνάρτησης τόσο για την εκτίμηση του μέγιστου αλλά και για την επιλογή της δράσης. Στη δημοσίευση της DeepMind [15] προτείνεται η χρήση δύο ξεχωριστών, αλλά ίδιων προδιαγραφών νευρωνικών δικτύων, ενός policy που θα χρησιμοποιείται για την επιλογή του action και ενός νέου target για την εκτίμηση της

Q τιμής, με βάρη θ_t^-. Συνεπώς ο στόχος κάθε επανάληψης διαφοροποιείται σε σχέση με τον αλγόριθμο 2 ως εξής:

$$y_t = r_{t+1} + \gamma Q(s_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(s_{t+1}, a; \theta_t); \theta_t^-) \tag{3.13}$$

Τα βάρη του δεύτερου δικτύου θ_t^- διατηρούνται σταθερά κατά την βελτιστοποίηση και ανανεώνονται περιοδικά με τα βάρη του policy net ανά έναν αριθμό βημάτων.

3.5.4 Dueling DQN

Στη δημοσίευση [16] προτείνεται μια νέα αρχιτεκτονική για το νευρωνικό δίκτυο που χρησιμοποιεί ο αλγόριθμος DQN. Συγκεκριμένα διαχωρίζονται δύο ροές (streams), μία για την εκτίμηση αποκλειστικά της value function V(s) των καταστάσεων και μία για την advantage function A(s, a), δηλαδή για το πόσο καλύτερο είναι να λάβω μια δράση α έναντι όλων των άλλων, ευρισκόμενος σε συγκεκριμένη κατάσταση. Το πλεονέκτημα αυτής της μεθόδου είναι ότι μπορεί να γενικεύσει τη μάθηση πέραν των δράσεων και είναι χρήσιμη όταν υπάρχουν πολλές καταστάσεις με δράσης παρόμοιας αξίας.

Οι δύο ροές συνενώνονται με τον τρόπο που φαίνεται παρακάτω, ώστε να μην διαφοροποιηθεί η έξοδος του νευρωνικού δικτύου:

$$Q(s, a) = V(s) + A(s, a) - \underset{a\prime}{\operatorname{mean}}\, A(s, a\prime) \tag{3.14}$$

Συμπερασματικά αυτή η τεχνική, μέσω του διαχωρισμού των καταστάσεων από τις δράσεις, μας βοηθά να έχουμε πιο αξιόπιστες εκτιμήσεις της action-value συνάρτησης.

3.5.5 Prioritized Experience Replay

Σε πολλά προβλήματα, ιδιαίτερα σε αυτά που συναντάμε στον πραγματικό κόσμο, η αλληλεπίδραση με το περιβάλλον μπορεί να είναι αρκετά κοστοβόρα. Συνεπώς η αξιοποίηση των εμπειριών που συλλέγουμε, καθίσταται εξαιρετικά σημαντική. Στη δημοσίευση της DeepMind [17] διατυπώνεται η άποψη, πως δεν έχουν όλες οι εμπειρίες την ίδια αξία και συνεπώς είναι προς το συμφέρον μας, να δειγματοληπτούμε συχνότερα από τη μνήμη αναπαραγωγής εκείνες τις εμπειρίες, που είναι σημαντικότερες. Υποστηρίζεται ακόμα, ότι οι εμπειρίες που παράγουν το μεγαλύτερο Temporal Difference σφάλμα είναι αυτές που ταιριάζουν λιγότερο, με την αντίληψη που έχει σχηματίσει το μοντέλο μας για το περιβάλλον και άρα αυτές στις οποίες πρέπει να δώσουμε μεγαλύτερο βάρος.

Για το σκοπό αυτό προτείνεται η αναπαραγωγή εμπειριών με προτεραιότητες. Οι προτεραιότητες όμως, με τις οποίες οι εμπειρίες θα τοποθετούνται στη μνήμη, δεν μπορούν να είναι απλά το σφάλμα των χρονικών διαφορών p_i καθώς κάτι τέτοιο θα οδηγούσε στο να δειγματοληπτούμε πολύ περιορισμένο αριθμό εμπειριών. Εισάγουμε λοιπόν μια

παράμετρο α που συμμετέχει στη σύνθεση της πιθανότητα επιλογής:

$$P(i) = \frac{p_i^a}{\sum_k p_k^a} \tag{3.15}$$

Αν το α ισούται με μηδέν έχουμε τυχαία επιλογή ενώ αν ισούται με ένα έχουμε αποκλειστικά με βάση το TD σφάλμα. Από τη στιγμή που χρησιμοποιούμε δειγματοληψία με βάση τις προτεραιότητες εισάγεται μεροληψία (bias) στη διαδικασία της εκπαίδευσης καθώς τα δείγματα που έχουν υψηλή προτεραιότητα πιθανόν θα χρησιμοποιηθούν πολλές φορές. Για να απαλύνουμε αυτό το πρόβλημα εισάγουμε τα βάρη σημαντικότητας δειγματοληψίας (importance samling weights IS) τα οποία μειώνουν το βαθμό που συμμετέχουν στις ανανεώσεις τα συχνά εμφανιζόμενα δείγματα.

Αλγόριθμος 4: Priorited Experience Replay

θ: initial network parameters, θ^- copy of θ
D: empty replay buffer, N: target network replacement freq
for *episode* $\leftarrow 1$, M **do**
  Get the current state of the environment s_t
  for $t \leftarrow 1$, T **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = max_a Q(s_t, a; \theta)$
    Execute action a_t and observe reward r_t and the next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) with maximal priority
    Set $s_t = s_{t+1}$
    Sample random minibatch of k transitions (s_j, a_j, r_j, s_{j+1}) from D
     with $j \sim P(j) = p_j^a / \sum_i p_i^a$
    Compute importance-sampling weights $w_j = (N \cdot P(j))^{-\beta} / max_i w_i$
    Define $a^{max}(s_{j+1}; \theta) = argmax_{a'} Q(s_{j+1}, a'; \theta)$
    $$y_j = \begin{cases} r_j & \text{for terminal states} \\ r_j + \gamma Q(s_{j+1}, a^{max}(s_{j+1}; \theta); \theta)^- & \text{for non terminal states} \end{cases}$$
    Compute TD-errors $\delta_j = y_i - Q(s_j, a_j; \theta)$
    Update transition priorities $p_j \leftarrow |\delta_j|$
    Perform a gradient descent step on $(w_j \cdot (y_j - Q(s_j, a_j; \theta)))^2$
    Replace target parameters $\theta^- \leftarrow \theta$ every N steps
  end
end

Τα βάρη σημαντικότητας ορίζονται ως $(\frac{1}{N}\frac{1}{P(i)})^b$, όπου N είναι το μέγεθος της μνήμης και b μία παράμετρος που καθορίζει τη σημαντικότητα των βαρών αυτών. Η τιμή του b ξεκινάει από το 0 στην αρχή της εκπαίδευσης και ανέρχεται στο 1 κατά το τέλος της αυτής όπου και η αμερόληπτη φύση των ανανεώσεων είναι πιο σημαντική.

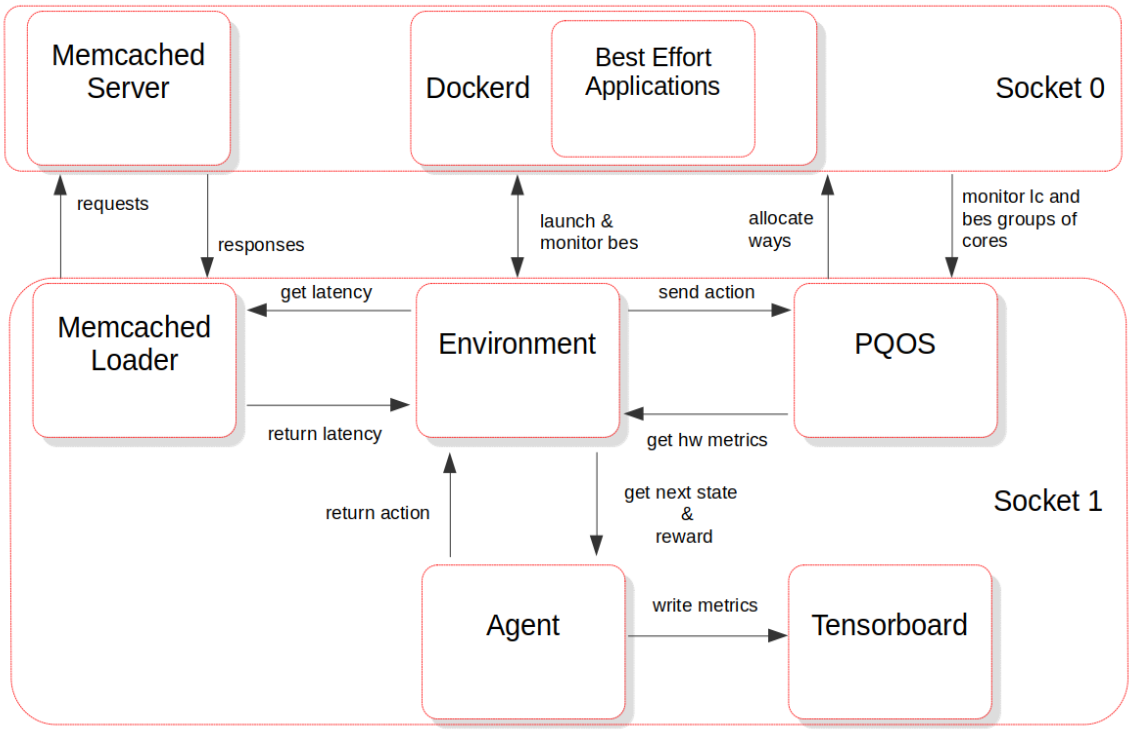
Για την υλοποίηση της μνήμης με προτεραιότητες χρειαζόμαστε μία αποδοτική δομή

δεδομένων. Τέτοια είναι η sumtree που μας επιτρέπει να ανανεώνουμε τις προτεραιότητες αλλά και να δειγματοληπτούμε σε $O(\log n)$. Πρόκειται για ένα δυαδικό δένδρο του οποίου τα φύλλα περιέχουν τις προτεραιότητες ενώ οι κόμβοι του το άθροισμα των τιμών των παιδιών τους. Στόχος μας είναι σε κάθε σύνολο δειγμάτων (batch) μεγέθους κ, που δειγματοληπτούμε, να λάβουμε δείγματα με διάφορες προτεραιότητες. Για αυτό χωρίζουμε το διάστημα [0, συνολική προτεραιότητα] σε κ διαστήματα και λαμβάνουμε τυχαία, για καθένα από αυτά, μια τιμή προτεραιότητας, την οποία και αναζητούμε στο δέντρο.

Κεφάλαιο 4

Περιγραφή Συστήματος

Σημαντική προσπάθεια καταβλήθηκε για τη σωστή οργάνωση όλων των τμημάτων που συναποτελούν το σύστημά μας, πάνω στο οποίο διεξήχθησαν μετέπειτα τα πειράματα. Στο σχήμα 4.1 απεικονίζονται τα διάφορα τμήματα και οι αλληλεπιδράσεις που αναπτύσσονται μεταξύ τους. Στη συνέχεια αναλύουμε καθένα από αυτά τα τμήματα.



Σχήμα 4.1: Τμήματα του συστήματος και αλληλεπιδράσεις αυτών

4.1 Υπηρεσία κρίσιμης απόκρισης

Ξεκινάμε με την υπηρεσία κρίσιμης απόκρισης (Latency Critical, LC) της οποίας την επίδοση θέλουμε να διασφαλίσουμε. Για το ρόλο αυτό, επιλέχτηκε η in-memory, key-value store βάση δεδομένων MemCached από τη σουίτα Cloudsuite [18]. Η συγκεκριμένη επιλογή δεν έγινε τυχαία καθώς πέρα από το ότι έχει χρησιμοποιηθεί ευρέως σε πλήθος εργασιών ([3, 6, 7]), κατέχει και σημαντική θέση στη βιομηχανία με μεγάλες

εταιρίες όπως το Facebook, το Netflix να την αξιοποιούν συστηματικά.

Ως μέτρο επίδοσης για τις διάφορες υπηρεσίες επιλέγεται το tail latency, δηλαδή ο χρόνος εξυπηρέτησης της μεγάλης μερίδας των εισερχόμενων αιτήσεων καθώς θεωρείται ότι αυτή η μετρική αποτυπώνει την εμπειρία του τελικού χρήστη. Το ποσοστό των αιτήσεων που θέλουμε να διεκπεραιώνεται σε συγκεκριμένο χρονικό όριο διαφέρει ανάλογα με τις απαιτήσεις κάθε υπηρεσίας με συνηθέστερες τιμές να είναι τα εκατοστημόρια (percentiles) 90, 95 και 99. Για τα πειράματά μας επιλέξαμε να ακολουθήσουμε για το Quality of Service (QoS) το χρονικό όριο που θέτει η Cloudsuite, δηλαδή τα 10 ms για το 95ο percentile των αιτήσεων.

4.2 Παραγωγή φορτίου και υπολογισμός απόκρισης

Όσο αφορά τις ανάγκες του προβλήματος λοιπόν, χρειαζόμαστε να έχουμε διαθέσιμο το tail latency της κρίσιμης υπηρεσίας, προκειμένου να είμαστε σε θέση να αποφανθούμε για το αν μια απόφασή μας, οδήγησε σε παραβίαση των ορίων που έχουμε θέσει ή όχι. Για να το πετύχουμε αυτό αφενός θα πρέπει να δημιουργήσουμε σημαντικό φορτίο προς τη βάση και στη συνέχεια να μπορέσουμε να μετρήσουμε το χρόνο που χρειάστηκε να διεκπεραιωθούν οι αιτήσεις που στείλαμε.

Για το σκοπό αυτό θα χρησιμοποιήσουμε τον loader της Cloudsuite. Η Cloudsuite είναι σουίτα για benchmarking υπηρεσιών νέφους, έτσι πέρα της ίδιας της MemCached παρέχεται και ένας client που ανταποκρίνεται στις ανάγκες που μόλις αναφέραμε. Αν και εκ πρώτης άποψης η μέτρηση του latency μοιάζει εύκολη υπόθεση, στην πράξη υπάρχουν πολλά σημεία που χρήζουν προσοχής, καθώς διαφορετικά κινδυνεύουμε να καταλήξουμε σε εντελώς αναξιόπιστες τιμές. Μια εξαιρετική μελέτη όλων των παγίδων στη διαδικασία της μέτρησης, πραγματοποιείται στο [19].

Αρχικά, στην πραγματική λειτουργία μιας βάσης δεδομένων τα διάφορα requests φτάνουν σε αυτή από ένα μεγάλο σε πλήθος διαφορετικών χρηστών. Αντίθετα αυτή η δυνατότητα δεν υπάρχει στους loader που παρέχουν οι διάφορες benchmarking σουίτες. Κάποιες από αυτές υιοθετούν μία closed-loop αρχιτεκτονική όπου ένα συγκεκριμένο πλήθος από νήματα είναι επιφορτισμένο με την αποστολή και λήψη των αιτήσεων/απαντήσεων. Καθένα από αυτά τα νήματα, προκειμένου να στείλει ένα νέο request, περιμένει την επιστροφή του προηγούμενου. Συνεπώς, όταν το latency της υπηρεσίας τείνει να αυξηθεί, τα νήματα περιμένουν περισσότερο χρόνο τις απαντήσεις, άρα καθυστερούν να στείλουν νέες αιτήσεις. Μοιραία, μειώνεται ο ρυθμός αποστολής των αιτήσεων από τον loader, με αποτέλεσμα αυτή η αύξηση του latency να μη συνεχίζεται και η τιμή του να υποτιμάται σε πολύ μεγάλο βαθμό.

Ο loader της Cloudsuite, που επιλέξαμε, ακολουθεί διαφορετική στρατηγική. Με βάση το φορτίο που ορίζουμε και το πλήθος των workers, δημιουργεί μία κατανομή για το χρόνο που γίνονται issue διαδοχικές αιτήσεις προς το server. Η κατανομή αυτή μπορεί

να είναι είτε ομοιόμορφη είτε εκθετική καθώς από τη θεωρία αναμονής γνωρίζουμε ότι από αυτή διέπεται ο χρόνος αφίξεων νέων πελατών σε έναν εξυπηρετητή. Έτσι λοιπόν δημιουργείται μία open-loop αρχιτεκτονική, όπου κάθε αίτημα αποστέλλεται ανεξάρτητα από την πορεία των προηγούμενων, σα να έχει προκύψει από διαφορετικό χρήστη.

Άλλο ένα σημαντικό θέμα που παρουσιάστηκε στις μετρήσεις μας, έχει να κάνει με την προέλευση του latency. Από τη στιγμή που πραγματοποιούμε *'από άκρη σε άκρη'* μετρήσεις, ένα τμήμα της μετρήσιμης τιμής οφείλεται στο latency της υπηρεσίας, ένα άλλο στο δικτυακό round trip του request/responce και τέλος στο latency του ίδιου του client που πραγματοποιεί τη μέτρηση. Στη δική μας περίπτωση αρχικά το latency του client έτεινε να κυριαρχεί των υπολοίπων. Αιτία για αυτό, ήταν το μεγάλο πλήθος αιτήσεων που χρειάζονταν, προκειμένου να κορεστεί έστω και ένα μόνο στιγμιότυπο (instance) του Memcached server και αυτό διότι η συγκεκριμένη υπηρεσία μπορεί να ανταποκριθεί με ευκολία σε μερικές εκατοντάδες χιλιάδες αιτήσεις το δευτερόλεπτο. Συνέπεια αυτού, ήταν ένα τεράστιο πλήθος απαντήσεων, να ανταγωνίζονται για το κλείδωμα στην μεριά του client, προκειμένου να προκύψουν τα συνολικά στατιστικά που χρειαζόμασταν.

Για την αντιμετώπιση αυτής της κατάστασης οι μηχανικοί της Facebook [19] ανέπτυξαν ένα πολύ εξελιγμένο σύστημα με lock free υλοποιήσεις και τεράστιο πλήθος loaders. Παρόλα αυτά στη δική τους περίπτωση στόχος είναι η μέτρηση του latency ενός παραγωγικού Memcached instance. Αντίθετα, εμάς δε μας ενδιαφέρει η μέγιστη επίδοση της εφαρμογής αυτής καθ' αυτής και έτσι παρατηρήσαμε ότι αν περιορίσουμε των αριθμό των νημάτων που έχει η βάση στη διάθεσή της, ώστε να ικανοποιεί εισερχόμενες αιτήσεις, από 4 σε 1, τότε ο loader της Cloudsuite μπορεί να επιτύχει τη κατάσταση κορεσμού, προτού παρουσιαστούν έντονα φαινόμενα ανταγωνισμού στη δική του πλευρά. Παράλληλα όμως διατηρήσαμε ένα επαρκές τμήμα (16 GB) κύριας μνήμης αποκλειστικά για τις ανάγκες της Memcached καθώς η λειτουργία της, προϋποθέτει και απαιτεί ότι το σύνολο των δεδομένων της βάσης μπορεί να χωρέσει στη μνήμη.

Μετά από δοκιμές υπολογίσαμε ότι ένα πλήθος αιτήσεων 90 χιλιάδων ανά δευτερόλεπτο, φέρνει τη Memcached υπηρεσία κοντά στα όρια που έχουμε θέσει, καθώς λαμβάνουμε τιμές για το q95 tail latency γύρω στα 4ms. Πράγματι το φαινόμενου κορεσμού του client δεν εμφανίζεται με ένα τέτοιου μεγέθους φορτίο, αφού αν σηκώσουμε έναν δεύτερο loader με πολύ μικρότερο φορτίο (άρα δεν υπάρχει πιθανότητα η μέτρηση αυτή να κυριαρχείται από client side latency), παίρνουμε παρόμοια τιμή μέτρησης με το instance του loader που αποστέλλει τον μεγάλο όγκο φορτίου.

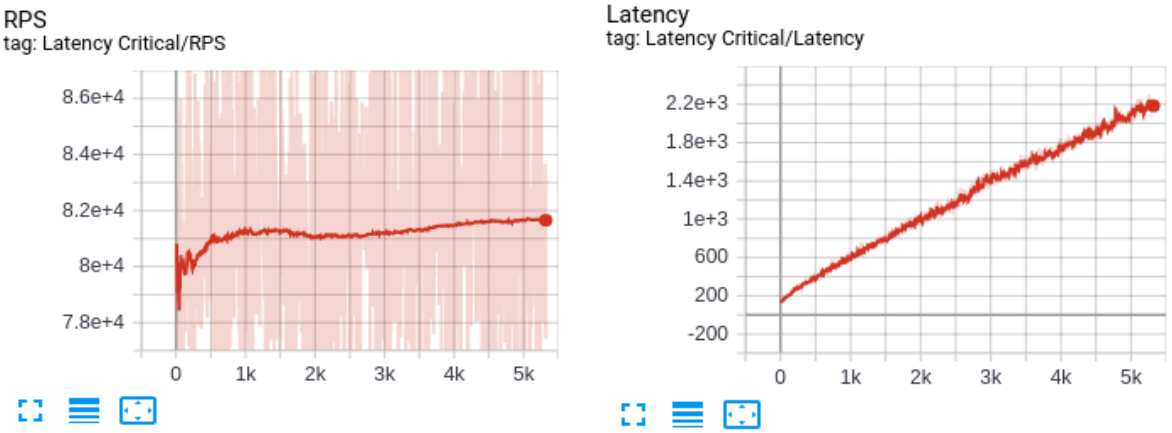
Τέλος, άλλο σημείο που έχρηζε προσοχής ήταν το είδος των αιτήσεων που αποστέλλουμε προς τη βάση. Το benchmark της ίδιας της Cloudsuite έθετε μια αναλογία 0,8 μεταξύ get και set requests. Ωστόσο όπως παρατηρήσαμε τα gets μπορούν να απαντηθούν από τη Memcached σε μικρότερο χρόνο σε σχέση με τα sets, κάτι που μοιάζει λογικό. Αυτό έχει σα συνέπεια ο χρόνος του tail lateny, να μονοπωλείται από τους χρόνους των sets, καθώς αυτά καθυστερούν περισσότερο. Συνεπώς για τη μέτρηση του latency δεν

έχει νόημα να στέλνεται κάποια αναλογία διαφορετικών αιτημάτων και συνεπώς προχωράμε στα πειράματά μας, αποκλειστικά με τα sets που δημιουργούν το μεγαλύτερο φόρτο στη βάση.

4.3 Εφαρμογές Βέλτιστης Προσπάθειας

Όσο αφορά τις εφαρμογές βέλτιστης προσπάθειας (Best Effort, BE) ή διαφορετικά χαμηλής προτεραιότητας, έγινε επιλογή τους από διάφορες πηγές. Αρχικά η σουίτα Cloudsuite πέρα από υπηρεσίες cloud διαθέτει και batch διεργασίες. Συγκεκριμένα διέθετε δύο Apache Spark εφαρμογές μία για in-memory και μία για graph analytics. Δεύτερη πηγή αποτέλεσαν τα παραδείγματα εφαρμογών Μηχανικής Μάθησης, που διαθέτει το ίδιο το Apache Spark [20]. Για την αξιοποίησή τους απαιτήθηκε η εύρεση ενός πραγματικού συνόλου δεδομένων από το αποθετήριο UCI [21] και η μετατροπή του στη μορφή LIBSVM [22]. Συνολικά χρησιμοποιήθηκαν 5 τέτοια batch jobs Μηχανικής Μάθησης.

Τέλος, για επιπλέον ποικιλία αξιοποιήθηκαν και εφαρμογές από τη σουίτα SPEC2006. Λόγω του ότι οι εφαρμογές αυτές, έχουν επιλεγεί στοχευμένα για να δημιουργούν φόρτο σε διάφορους πόρους του μηχανήματος, κατά τις συνεκτελέσεις με 9 ίδια στιγμιότυπα, παρατηρήσαμε ότι ο ανταγωνισμός είναι τέτοιος, που ο MemCached Server δε μπορεί να ανταποκριθεί στην εισερχόμενη κίνηση.



(a) Responses Per Second

(b) Tail Latency

Σχήμα 4.2: Προβλήματα κατά τη συνεκτέλεση με την εφαρμογή lbm. Παρατηρείται σταθερή υστέρηση 10% στον ρυθμό των απαντήσεων από την Memcached υπηρεσία. Ως αποτέλεσμα το tail latency αυτής εμφανίζεται να αυξάνει γραμμικά με το χρόνο.

Συγκεκριμένα, παρατηρήσαμε σημαντική απόρριψη αιτήσεων. Όπως φαίνεται στην εικόνα 4.2, από τις 90 χιλιάδες αιτήσεις που αποστέλλονται το δευτερόλεπτο, εξυπηρετούνται μόνο οι 81 χιλιάδες περίπου. Συνεπώς η κρίσιμη υπηρεσία παρουσιάζει μια

σταθερή υστέρηση της τάξης του 10% στην εξυπηρέτηση της εισερχόμενης κίνησης, γεγονός που επιφέρει γραμμική αύξηση ως προς το χρόνο του tail latency. Με βάση τον τρόπο που έχουμε ορίσει το πρόβλημα δεν μπορούμε να κάνουμε λόγο για tail latency, τη στιγμή που δε είναι εφικτό να εξυπηρετηθεί η οριζόμενη κίνηση. Εξάλλου όπως δηλώνεται στο [23], τα παραδοσιακά workstattion benchmarks όπως αυτά της σουίτας SPEC, δεν συμβαδίζουν με τα batch jobs που απαντώνται στα κέντρα δεδομένων, αφού απαιτούν ένα πολύ μεγαλύτερο αριθμό από ways για να ικανοποιήσουν τις ανάγκες του. Ως εκ τούτου, από τις 5 SPEC εφαρμογές που ετοιμάσαμε, αναγκαστήκαμε να απορρίψουμε τις 3 από αυτές.

Οι εφαρμογές που η σουίτα Cloudsuite περιέχει χρησιμοποιούν docker containers ([24]) για την εύκολη και άμεση χρήση τους. Docker container είναι ένα ελαφρύ, αυτόνομο, εκτελέσιμο πακέτο λογισμικού που περιλαμβάνει όλα όσα χρειάζονται για να τρέξει μια εφαρμογή: κωδικά, περιβάλλον εκτέλεσης, εργαλεία συστήματος, βιβλιοθήκες συστήματος και ρυθμίσεις. Με τη χρήση τους αποφεύγουμε δηλαδή να εγκαταστήσουμε κάθε ανάγκη όλων αυτών των εφαρμογών στο σύστημά μας καθώς αυτές εμπεριέχονται στο εκάστοτε container. Για τη συνολική διαχείριση των εφαρμογών εντάξαμε και τις υπόλοιπες σε containers.

4.4 PQOS

Έχοντας καταλήξει με τις εφαρμογές που θα συμμετέχουν στη συνεκτέλεση, χρειαζόμαστε και ένα τρόπο παρακολούθησης αυτών αλλά και ορισμού του τμήματος της Last Level Cache στο οποίο μπορούν να έχουν πρόσβαση. Θα χρησιμοποιήσουμε την τεχνολογία RDT της Intel που αναφέραμε προηγουμένως (2.2) μέσω της προγραμματιστικής διεπαφής που προσφέρει το PQOS [25]. Η Intel διαθέτει και έναν Python wrapper της διεπαφής, τον οποίο και χρησιμοποιούμε, καθώς η υπόλοιπη υλοποίησή μας έχει επίσης αναπτυχθεί σε Python.

Χρησιμοποιούμε την τεχνολογία RDT σε επίπεδο πυρήνων. Δημιουργούμε δύο ομάδες πυρήνων (groups), ένα με τους πυρήνες που χρησιμοποιεί η υπηρεσία κρίσιμης απόκρισης και ένα με όσους αναθέτουμε στις εφαρμογές χαμηλής προτεραιότητας. Για την εκτέλεση των πειραμάτων μας, μόνο ένας πυρήνας ανατέθηκε στην MemCached και οι υπόλοιποι 9, που διαθέτει το ένα από τα δύο socket του server μας, χρησιμοποιήθηκαν για τις ανάγκες των BEs. Να αναφέρουμε σε αυτό το σημείο πως δεν κάνουμε χρήση του hyperthreading, καθώς διαφορετικά θα εμφανιζόταν ανταγωνισμός και στις ιδιωτικές cache των πυρήνων. Βασιζόμαστε μόνο στους 10 φυσικούς πυρήνες της κάθε επεξεργαστικής μονάδας.

Το PQOS μας δίνει τη δυνατότητα παρακολούθησης, για κάθε group, μετρικών όπως το μέγεθος της LLC που καταλαμβάνει, το bandwidth που χρησιμοποιεί αλλά και το IPC και τα misses μέσω του εργαλείου perf ([26]). Όσο αφορά το allocation των ways της LLC, αντιστοιχούμε ένα Class of Services (COS) με καθένα από τα groups. Στο ένα

group αντιστοιχούμε ways πάντοτε ξεκινώντας από τα αριστερά ενώ στο άλλο από τα δεξιά. Σε κάθε group θα αντιστοιχεί τουλάχιστον ένα way, ενώ λόγω περιορισμών του PQOS στο group που λαμβάνει ways από αριστερά, αντιστοιχίζονται τουλάχιστον δύο ways. Επιλέχθηκε αυτό το group να είναι της κρίσιμης υπηρεσίας. Κάθε φορά λοιπόν μετατρέπουμε την απόφαση που παίρνουμε από τον πράκτορα σε bitmasks, ξένα μεταξύ τους, και τα εφαρμόζουμε στα δύο επιλεγμένα Class of Services.

4.5 Περιβάλλον

Στρατηγική θέση στην υλοποίησή μας κατέχει το περιβάλλον. Ο ρόλος του είναι να κρύβει την πολυπλοκότητα όλων των τμημάτων που περιγράφηκαν έως τώρα και να παρουσιάζει μια συγκεκριμένη διεπαφή προς τον πράκτορα. Ακολουθούμε το πρότυπο διεπαφής που εισήγαγε η ευρέως χρησιμοποιούμενη βιβλιοθήκη της OpenAI, Gym ([27]). Έτσι εύκολα μπορούν στο μέλλον να συνδεθούν νέοι αλγόριθμοι, αρκεί αυτοί να είναι συμβατοί με τη συγκεκριμένη διεπαφή. Στη συνέχεια περιγράφουμε τις ενέργειες που επιτελεί το περιβάλλον.

Αρχικά λοιπόν εκκινείται ο loader της Memcached βάσης δεδομένων, η οποία τρέχει ήδη στο μηχάνημα. Στη συνέχεια, ανάλογα και με τις παραμέτρους που έχουμε θέσει, εκκινεί ένα πλήθος από εφαρμογές χαμηλής προτεραιότητας που θα συνεκτελεστούν με την κύρια υπηρεσία. Όταν πλέον έχει εκκινήσει την λειτουργία του ο πράκτορας, τον τροφοδοτεί με την κατάσταση του περιβάλλοντος και λαμβάνει την επόμενη δράση του. Αυτή την εφαρμόζει με χρήση του PQOS στο φυσικό μηχάνημα και στη συνέχεια πρέπει για αυτή τη νέα απόφαση, να συλλέξει τις τιμές των μετρικών που επηρεάζονται. Κατόπιν, σχηματίζει την επόμενη κατάσταση, υπολογίζει την ανταμοιβή του πράκτορα και τα αποστέλλει πίσω σε αυτόν.

Επειδή ο loader της Memcached που χρησιμοποιούμε από τη σουίτα Cloudsuite είναι υλοποιημένος στη γλώσσα προγραμματισμού C, η επικοινωνία με το περιβάλλον γίνεται μέσω sockets. Το κύριο πρόγραμμα ξεκινά την επικοινωνία ζητώντας από τον loader τις software μετρικές της Memcached. Εκείνος με το που λάβει ένα τέτοιο αίτημα μηδενίζει τα στατιστικά που διατηρεί και πέφτει σε sleep για ένα μικρό διάστημα (ώστε να επιστρέψει τιμές που έχουν προκύψει, αφότου έχει εφαρμοστεί η νέα απόφαση). Ξυπνώντας αποστέλει πίσω τις μετρικές στο περιβάλλον. Παρόμοια, δηλαδή με μηδενισμό των μετρητών, συλλέγονται και οι hardware μετρικές μέσω του PQOS, χωρίς να απαιτείται όμως διεργασιακή επικοινωνία λόγω της ενσωματομένης υλοποίησης σε Python.

Τέλος, να αναφέρουμε ότι σε συχνά διαστήματα το περιβάλλον ελέγχει αν έχουν ολοκληρωθεί οι εφαρμογές βέλτιστης προσπάθειας και ανάλογα με το πείραμα που εκτελούμε, μπορεί να προγραμματίσει για εκτέλεση νέες. Ουσιαστικά για τη διαχείριση των εφαρμογών υλοποιούμε έναν scheduler που μας επιτρέπει να προσδιορίσουμε ποιες εφαρμογές θα εκτελεστούν, πόσες φορές θα επαναληφθεί η κάθε μία ή ακόμα, μας δίνει την δυνατότητα να θέσουμε κάποιο seed και η επιλογή αυτών να γίνεται τυχαία.

4.6 Πράκτορας Ενισχυτικής Μάθησης

Σημαντικότερο τμήμα του όλου συστήματος είναι ο πράκτορας. Ο πράκτορας συνδέεται με το περιβάλλον, προκειμένου να λάβει την κατάσταση αυτού και να εφαρμόσει τις αποφάσεις του, όπως φαίνεται στο σχήμα 3.1. Από τις μετρικές που είχαμε διαθέσιμες συμπεριλάβαμε στην κατάσταση του περιβάλλοντος το πλήθος των Misses που συμβαίνουν κανονικοποιημένο ως προς το χρόνο (kilo cycles), καθώς και τα ways που είναι διαθέσιμα για τις BE εφαρμογές.

Όσο αφορά την κατάσταση που λαμβάνουμε, θυμίζουμε ότι στόχος μας είναι, να αποτυπώνει το επίπεδο του ανταγωνισμού, που δέχεται η κρίσιμη υπηρεσία από τις εφαρμογές βέλτιστης προσπάθειας, ώστε βάση αυτού να αποφαίνεται ο πράκτορας για τα ways που πρέπει να διαθέσει στα BEs. Όπως παρατηρήσαμε και περιγράψαμε στο κεφάλαιο 2.4, οι διακυμάνσεις των misses των best effort εφαρμογών, αποτυπώνονται στο latency της κρίσιμης εφαρμογής, για αυτό και επιλέγουμε τη συγκεκριμένη μετρική. Συμπεριλάβαμε επίσης τα ways που ανατίθενται σε κάθε βήμα, ώστε ο πράκτορας να είναι σε θέση να μπορεί να διαχωρίσει το επίπεδο ανταγωνισμού από την απόφαση που λαμβάνει. Δηλαδή μία μεγάλη τιμή Misses όταν έχουν ανατεθεί πολύ λίγα ways στα BEs μπορεί να συνιστά μικρότερο ανταγωνισμό σε σχέση με μία μικρότερη τιμή Misses με πολύ περισσότερα ways.

Δε συμπεριλάβαμε επιπλέον μετρικές στην κατάσταση καθώς όπως φαίνεται και στις εικόνες 2.3, 2.5 τα πρότυπα που εμφανίζονται τόσο στο IPC όσο και στο Bandwidth είναι πανομοιότητα με αυτό των Misses και συνεπώς δεν προσδίδουν κάποια σημαντική επιπλέον πληροφορία στον πράκτορα. Για να αποτυπωθούν πρότυπα περισσοτέρων μετρικών, απαιτείται ένα μεγαλύτερο νευρωνικό δίκτυο, κάτι που έχει σημαντικές συνέπειες στο χρόνο που χρειάζεται για την ολοκληρωμένη εκπαίδευση του μοντέλου. Στο συγκεκριμένο πρόβλημα δεν έχουμε την πολυτέλεια του χρόνου, καθώς ο πράκτορας πρέπει άμεσα να αρχίσει να παίρνει σωστές αποφάσεις.

Για την υλοποίηση του πράκτορα χρησιμοποιούμε τον αλγόριθμο DQN και τις βελτιστοποιήσεις αυτού που παρουσιάστηκαν στο κεφάλαιο 3.5. Το νευρωνικό δίκτυο του αλγορίθμου αποτελείται από δύο κρυφά επίπεδα ενώ δοκιμάστηκαν διάφορες αρχιτεκτονικές κατά τη διάρκεια των πειραμάτων. Ως συνάρτηση ενεργοποίησης των νευρώνων χρησιμοποιούμε την ELU ([28]). Ως συνάρτηση απωλειών χρησιμοποιούμε το μέσο τετραγωνικό σφάλμα και τέλος ως optimizer τη μέθοδο Adam ([14]). Η υλοποίηση αυτού έγινε με χρήση της βιβλιοθήκης νευρωνικών δικτύων Pytorch ([29]) που διανέμει ως λογισμικό ανοικτού κώδικα η Facebook και τα τελευταία χρόνια έχει υιοθετεί σε μεγάλο βαθμό από την ερευνητική κοινότητα και όχι μόνο.

Όπως προείπαμε, το πρόβλημα βελτιστοποίησης που καλούμαστε να λύσουμε είναι διττό. Από τη μία πλευρά προσπαθούμε να προστατεύσουμε την επίδοση της υπηρεσίας και από την άλλη να αυξήσουμε τη χρήση του μηχανήματος εκτελώντας παράλληλα όσο καλύτερα γίνεται εφαρμογές χαμηλής προτεραιότητας. Αυτή η κατάσταση πρέπει να αποτυπώνεται στη συνάρτηση ανταμοιβής του πράκτορα, την οποία και παρουσιάζουμε

παρακάτω:

$$R = \begin{cases} \text{num of ways allocated to best effort apps}, & \text{if latency} < \text{threshold} \\ -\text{penalty coef} \cdot \max \text{ ways}, & \text{otherwise} \end{cases} \tag{4.1}$$

Με βάση το παραπάνω σχήμα λοιπόν, ανταμοίβουμε τον πράκτορα, όταν αυτός δεν παραβιάζει το όριο του latency, όχι με κάποια σταθερή ποσότητα, αλλά ανάλογα με το με πόσους λιγότερους πόρους κατάφερε κάτι τέτοιο, ή ισοδύναμα με το πόσα ways διέθεσε στις εφαρμογές καλύτερης προσπάθειας. Θεωρούμε κατά αυτό τον τρόπο ότι η πρόοδος των BEs είναι συνάρτηση των ways που τους αποδίδονται. Σε διαφορετική περίπτωση τον τιμωρούμε με μία μεγάλη αρνητική ανταμοιβή που τίθεται ίση με το μέγιστο πλήθος ways προς διάθεση, επί κάποιον συντελεστή. Ο συντελεστής αυτός καθορίζει την ανοχή του πράκτορα σε παραβιάσεις του ορίου και είναι μία παράμετρος με την οποία πειραματιστήκαμε.

Όσο αφορά τώρα το δίλημμα εξερεύνησης εκμετάλλευσης ο αλγόριθμος χρησιμοποιεί την ε-άπληση πολιτική που αναφέραμε στην ενότητα 3.4.2, με τη διαφορά ότι σε κάθε βήμα ελαττώνουμε εκθετικά την πιθανότητα ε με την οποία εξερευνούμε (4.2).

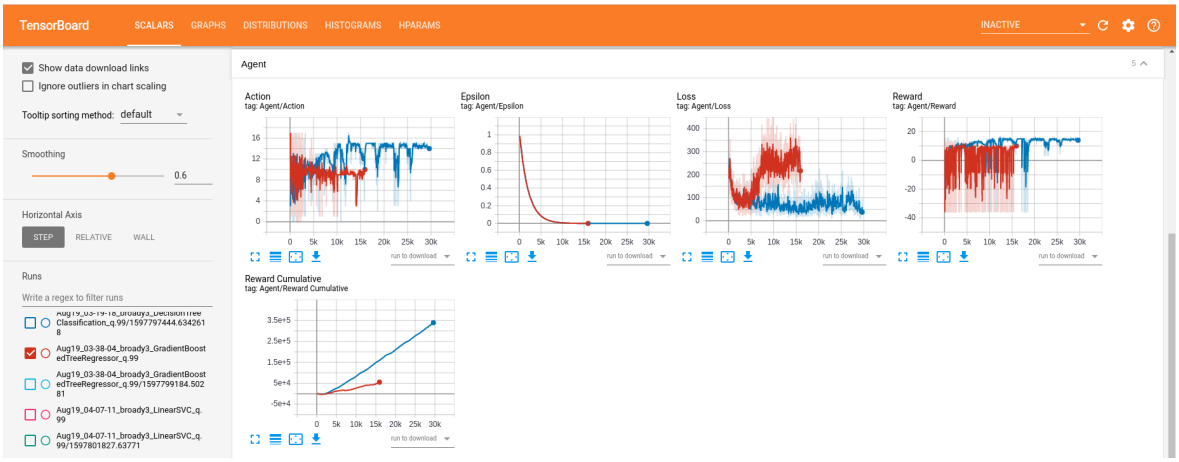
$$\epsilon = \epsilon_{end} + (\epsilon_{start} - \epsilon_{end}) \cdot e^{-steps \cdot \epsilon_{decay}} \tag{4.2}$$

Έτσι λοιπόν στην αρχή η επιλογή των δράσεων γίνεται στην τύχη ενώ προοδευτικά βασιζόμαστε όλο και περισσότερο στη γνώση που έχουμε συσσωρεύσει. Ο ρυθμός με τον οποίο μειώνουμε το ποσοστό εξερεύνησης, επιλέχθηκε έτσι ώστε στον περιορισμένο χρόνο που διαρκούν τα πειράματα μας, ο πράκτορας από τη μία να εξερευνήσει αρκετά τις φάσεις των προγραμμάτων, από την άλλη όμως να εκμεταλλευτεί και τη γνώση που αποκτά, ώστε να φανεί και η επίδραση της λειτουργίας του στο τελικό αποτέλεσμα. Τέλος για την τεχνική αναπαραγωγής εμπειριών χρησιμοποιούμε μία μνήμη μεγέθους 10 χιλιάδων παρελθοντικών στιγμιοτύπων (από εκεί και έπειτα διατηρούνται τα πιο πρόσφατα).

4.7 Tensorboard

Τελευταίο κομμάτι του συστήματός μας είναι το Tensorboard που χρησιμοποιούμε για την απεικόνιση διάφορων μετρικών. Είναι γεγονός πως αν και έχει αναπτυχθεί ένα πλήθος βιβλιοθηκών υψηλού επιπέδου που καθιστούν εξαιρετικά εύκολη τη δημιουργία και τη χρήση ενός νευρωνικού δικτύου, δεν είναι το ίδιο ομαλή η διαδικασία της εκπαίδευσής τους. Τα νευρωνικά δίκτυα πολλές φορές λόγω της πολυπλοκότητάς τους μοιάζουν σαν ένα μαύρο κουτί όπου αδυνατούμε να δούμε τι γίνεται στο εσωτερικό τους ώστε να παρέμβουμε κατάλληλα στη διαδικασία της εκπαίδευσης. Για να αντιμετωπίσει αυτό το πρόβλημα η Google ανέπτυξε για τις ανάγκες της δικής της βιβλιοθήκης νευρωνικών δικτύων (Tensorflow) το Tensorboard ([30]), μια εργαλειοθήκη απεικόνισης διαφόρων

πτυχών των νευρωνικών δικτύων. Η χρήση του διαδόθηκε μεταξύ της κοινότητας της βαθιάς μηχανικής μάθησης και υιοθετήθηκε και από την Pytorch.



Σχήμα 4.3: Απεικόνιση μετρικών στο Tensorboard για δύο διαφορετικά πειράματα

Στην παρούσα εργασία χρησιμοποιούμε το Tensorboard για να απεικονίσουμε τις μεταβολές των βαρών των δικτύων, μια σειρά από υπερπαραμέτρους, αλλά και πτυχές του περιβάλλοντος όπως τις hardware και software μετρικές των συνεκτελούμενων εφαρμογών. Η συμβολή του στη διεξαγωγή των πειραμάτων ήταν καθοριστική καθώς μας επέτρεψε να έχουμε άμεσα διαθέσιμη όλη την απαραίτητη πληροφορία που χρειαζόμασταν για την αξιολόγηση των διαφόρων δοκιμών. Συνέβαλλε επίσης στην έγκαιρη αντιμετώπιση προβλημάτων και ανωμαλιών ακόμα και από εξωγενείς παράγοντες, που διαφορετικά θα ήταν εξαιρετικά δύσκολο να εντοπιστούν τόσο άμεσα.

Κεφάλαιο 5

Πειραματική Μελέτη

Στο παρόν κεφάλαιο εκθέτουμε και αναλύουμε τα αποτελέσματα των πειραμάτων μας. Προκειμένου να αξιολογήσουμε διεξοδικά την επίδοση του πράκτορα, εκτελέσαμε μια σειρά από διαφορετικά σενάρια.

5.1 Παράμετροι Πειραμάτων

Το πρόβλημά μας χαρακτηρίζεται από ένα μεγάλο αριθμό παραμέτρων, που αφορούν είτε πτυχές του περιβάλλοντος, είτε του αλγορίθμου μάθησης. Καταγράφουμε συγκεντρωτικά παρακάτω τις επιλογές που κάναμε (πίνακας 5.1 και πίνακας 5.2).

Παράμετροι Περιβάλλοντος	Τιμές
id_cores_lc	0
id_cores_bes	1-9
cores per be	1
service requests per second	90,000
type of requests	sets
target	10ms
quantile	q95
penanlty coefficient	2
time interval	50ms

Πίνακας 5.1: Παράμετροι του Περιβάλλοντος

Για πολλές από τις συγκεκριμένες παραμέτρους πειραματιστήκαμε, προκειμένου να διαπιστώσουμε ποιες τιμές μας δίνουν την καλύτερη συμπεριφορά. Στις επόμενες ενότητες παρουσιάζουμε κάποιες από αυτές τις μελέτες, που θεωρούμε ότι παρουσιάζουν ενδιαφέρον, όπως η επιρροή του διαστήματος απόφασης και η συνεισφορά των βελτιστοποιήσεων του αλγορίθμου DQN.

Παράμετροι Πράκτορα	Τιμές
hidden layers	2
neurons per layer	[24, 48]
input feature	MPKC
network architecture	Dueling
algorithm	Double DQN
memory type	Classic
learning rate	0.001
batch size	64
target_net update period	100 steps
discount factor γ	0,95
memory size	10,000
epsilon_start	1
epsilon_end	0
epsilon_decay	0.001

Πίνακας 5.2: Παράμετροι του Πράκτορα

5.2 Χρήση του πράκτορα σε συνεκτελέσεις

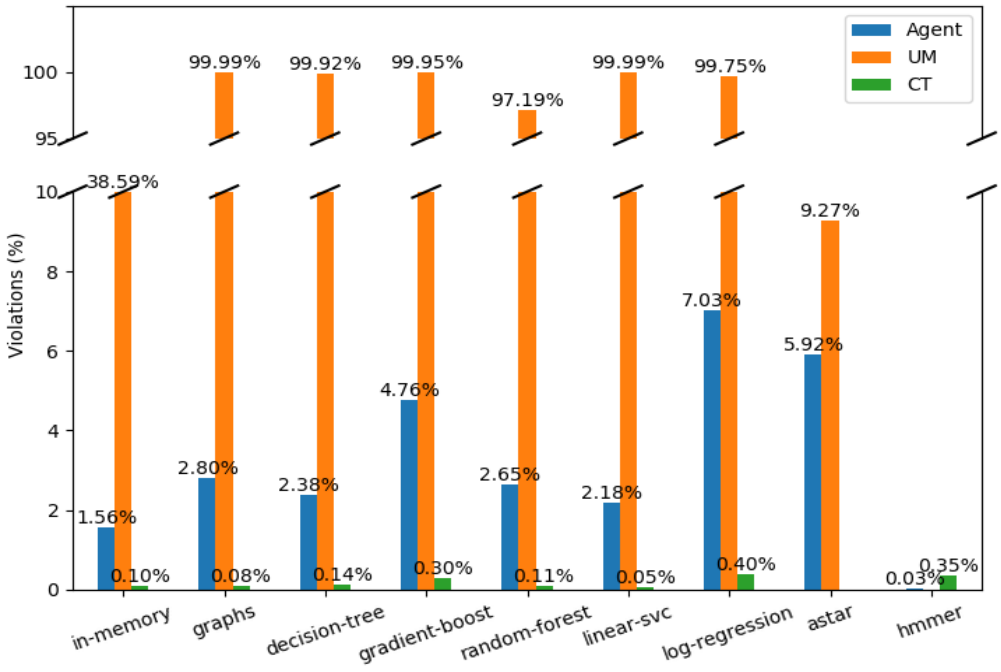
Πρώτα απ' όλα παρουσιάζουμε τη λειτουργία του πράκτορα, σε συνεκτελέσεις της κρίσιμης υπηρεσίας με ένα πλήθος διαφορετικών εφαρμογών χαμηλής προτεραιότητας. Σε κάθε συνεκτέλεση τοποθετούνται μαζί με την κρίσιμη υπηρεσία, 9 στιγμιότυπα της ίδιας BE εφαρμογής. Κάθε πείραμα διαρκεί μέχρι να ολοκληρωθούν όλα τα στιγμιότυπα και ως ότου συμβεί αυτό, όσα τελειώνουν επανεκκινούνται. Συγκρίνουμε την απόδοση σε σχέση με το να διαθέταμε το σύνολο των ways πλην ενός, στην κρίσιμη υπηρεσία (Cache Takeover, CT), προσέγγιση που προσπαθεί να διασφαλίσει κατά το δυνατόν περισσότερο, ότι δεν πρόκειται η υπηρεσία να ξεπεράσει τα όρια της. Από την άλλη συγκρίνουμε και με την κατάσταση Unmanaged (UM), όπου δεν λαμβάνεται κανένας περιορισμός, ώστε να έχουμε ένα μέτρο σύγκρισης, του κατά πόσο επιβραδύνει ο πράκτορας την εκτέλεση των εφαρμογών χαμηλής προτεραιότητας.

Από τον πίνακα 5.3, φαίνεται πως η πολιτική CT, πετυχαίνει να διαφυλάξει την επίδοση της υπηρεσίας σε όλες τις περιπτώσεις. Μάλιστα τα ποσοστά παραβιάσεων κυμαίνονται σε επίπεδα αρκετά μικρότερα του 1%. Ωστόσο ο χρόνος που χρειάζεται για την ολοκλήρωση των πειραμάτων είναι πολλαπλάσιος εκείνου που χρειάστηκε κατά την εκτέλεση χωρίς απομονωμένη πρόσβαση στη κρυφή μνήμη. Ενδεικτικά ο χρόνος εκτέλεσης για την εφαρμογή in-memory υπερτετραπλασιάζεται. Από την άλλη, τα ποσοστά παραβιάσεων, του ορίου για το latency της κρίσιμης υπηρεσίας, στις συνεκτελέσεις χωρίς παρέμβαση, αγγίζουν το 100% για την πλειοψηφία των εφαρμογών.

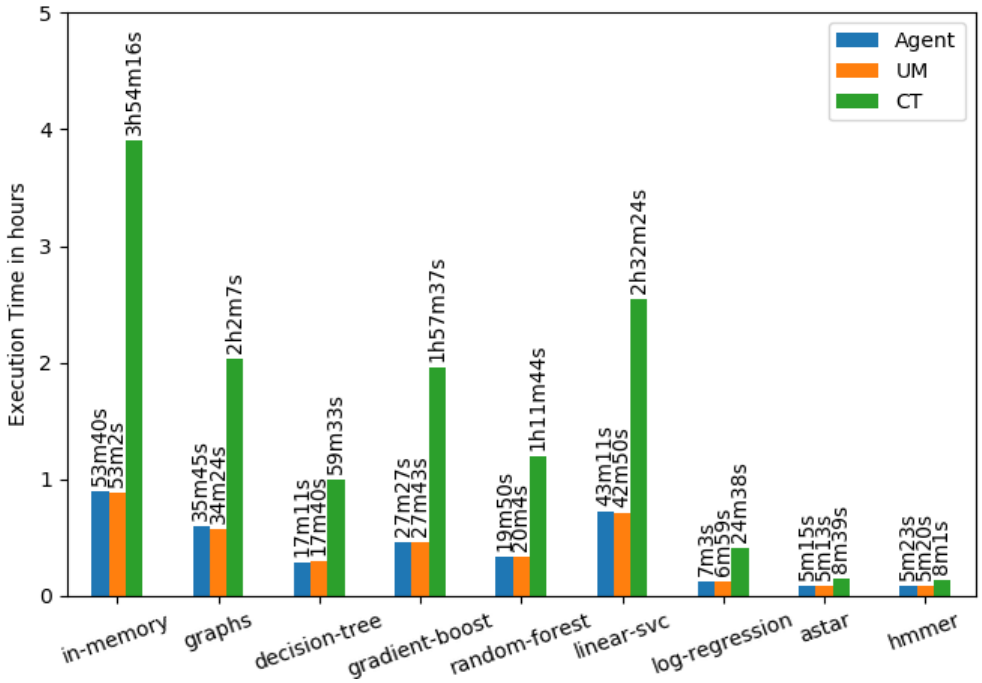
Περνώντας τώρα στον πράκτορα, παρατηρούμε πως καταφέρνει να συγκεράσει τις

BE apps	Agent			CT		UM	
	Viol.	Post Exp.	Time	Viol.	Time	Viol.	Time
in-memory	1.56%	0.94%	53m41s	0,10%	234m16s	38.59%	53m03s
graphs	2.80%	1.15%	35m46s	0.08%	122m07s	99.99%	34m25s
decision-tree	2.38%	0.52%	17m11s	0,14%	59m34s	99.92%	17m40s
gradient-boost	4.76%	4.29%	27m27s	0.30%	117m37s	99,95%	27m44s
random-forest	2.65%	1.50%	19m51s	0.11%	71m45s	97.19%	20m05s
linear-svc	2.18%	1.60%	43m11s	0.05%	152m24s	99.99%	42m51s
log-regression	7.03%	-	07m04s	0.40%	24m39s	99.75%	06m59s
astar	5.92%	-	05m16s	0%	08m39s	9.27%	05m13s
hmmer	0.03%	-	05m24s	0.35%	08m02s	0.07%	05m20s

Πίνακας 5.3: Συγκεντρωτικά αποτελέσματα επιδόσεων του πράκτορα έναντι της διάθεσης ενός way (CT) και του μη διαχωρισμού της LLC



Σχήμα 5.1: Σύγκριση των επιδόσεων του πράκτορα έναντι των πολιτικών CT και UM, ως προς τις παραβιάσεις του ορίου της κρίσιμης υπηρεσίας



Σχήμα 5.2: Σύγκριση των επιδόσεων του πράκτορα έναντι των πολιτικών CT και UM, ως προς το χρόνο ολοκλήρωσης των εφαρμογών χαμηλής προτεραιότητας

δύο καταστάσεις. Αφενός τα ποσοστά παραβιάσεων που πετυχαίνει, είναι χαμηλά, μη ξεπερνώντας το 5%. Εξαίρεση αποτελούν οι περιπτώσεις συνεκτέλεσης με τα BEs log-regression και astar, των οποίων η διάρκεια είναι πολύ μικρή και έτσι ο πράκτορας δεν ολοκληρώνει καν την αρχική φάση όπου κυριαρχεί η εξερεύνηση. Αφετέρου οι χρόνοι που πετυχαίνει για τα BEs, πλησιάζουν πάρα πολύ εκείνους της εκτέλεσης χωρίς διαμοιρασμό. Συνεπώς καταφέρνει να προφυλάξει την υπηρεσία υψηλής προτεραιότητας χωρίς να βλάψει ουσιαστικά τις εφαρμογές βέλτιστης προσπάθειας.

Μια πιο εποπτική απεικόνιση των εν λόγω αποτελεσμάτων γίνεται στις εικόνες 5.1 και 5.2, όπου συγκρίνονται τα ποσοστά παραβιάσεων και οι χρόνοι περάτωσης των εφαρμογών αντίστοιχα.

Για να ποσοτικοποιήσουμε τώρα, το πως καταφέρνει ο πράκτορας να ανταποκριθεί και στους δύο στόχους του, ορίζουμε ένα συνδυαστικό μέτρο (Combined Metric, CI) ως εξής:

$$Speedup = \frac{Time^{BE}_{alone}}{Time^{BE}} \tag{5.1}$$

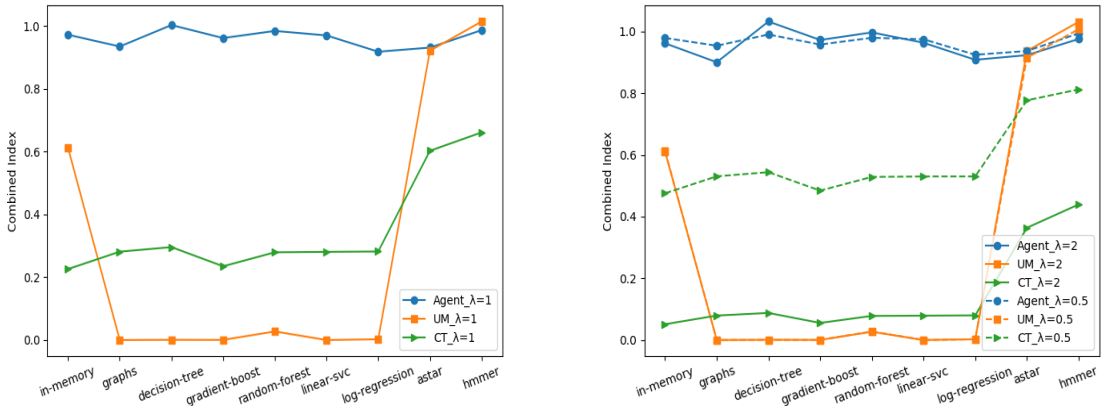
$$CI = (1 - violations) * Speedup^{\lambda} \tag{5.2}$$

Η μετρική αυτή απαρτίζεται από δύο συνιστώσες που αντιστοιχούν στην χρησιμοποίηση του server και στο σεβασμό του ορίου της υπηρεσίας. Καταρχάς όσο αφορά τη

χρησιμοποίηση του server υπολογίζουμε το κλάσμα του χρόνου που χρειάστηκε για να ολοκληρωθεί εκτελούμενη μόνη της η BE εφαρμογή, προς τον αντίστοιχο χρόνο κατά τη συνεκτέλεση με την κύρια υπηρεσία και χρήση του εκάστοτε μηχανισμού. Το αποτέλεσμα αυτό πολλαπλασιάζεται με το συμπληρωματικό του ποσοστού παραβιάσεων του QoS ως προς τη μονάδα. Κατά αυτό τον τρόπο λαμβάνουμε υπόψιν μας τα όποια χρονικά οφέλη, μόνο κατά το ποσοστό που δεν παραβιάστηκαν τα όρια.

Τέλος, να αναφέρουμε ότι η παράμετρος λ επηρεάζει τη σημαντικότητα των δύο μερών, ώστε ανάλογα με τις απαιτήσεις του κάθε προβλήματος να αξιολογείται και η επίδοση των διάφορων μηχανισμών. Για λ=1 η βαρύτητα και των δύο συνιστωσών είναι η ίδια. Αν η χρησιμοποίηση του εξυπηρετητή είναι πιο σημαντική για εμάς, το λ πρέπει να είναι μεγαλύτερο του 1, αλλιώς αν μας ενδιαφέρει περισσότερο η τήρηση των ορίων πρέπει το λ να κρατηθεί μικρότερο της μονάδας.

Στο σχήμα 5.3 παρουσιάζουμε τις τιμές αυτού του μέτρου για τις τρεις μεθόδους και για όλες τις συνεκτελέσεις. Ακόμα δοκιμάζουμε επιπλέον τιμές του λ, 0,5 και 2.



(a) λ=1 (b) λ=0.5 and λ=2

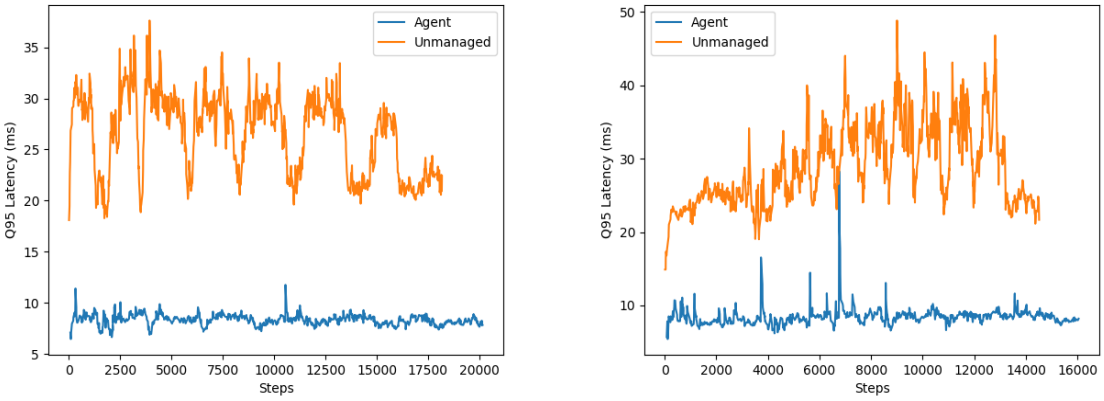
Σχήμα 5.3: Αξιολόγηση της επίτευξης των στόχων των πολιτικών του πράκτορα και των μεθόδων CT, Unmanaged με βάση το συνδυαστικό μέτρο. Εξετάζεται και η επιρροή της παραμέτρου λ

Είναι εμφανές πως ο πράκτορας καταφέρνει να πετύχει πολύ υψηλή επίδοση στο συγκεκριμένο δείκτη, πλησιάζοντας το μέγιστο για το σύνολο των best effort εφαρμογών. Οι πολιτικές CT και UM μένουν πολύ πιο πίσω γιατί αποτυγχάνουν εντελώς να ικανοποιήσουν τον έναν από τους δύο στόχους, η μεν CT τη χρησιμοποίηση του server, η δε UM την προστασία της υπηρεσίας.

Στην εικόνα 5.3(b) βλέπουμε πως η διαφοροποίηση της τιμής του λ δεν επηρεάζει ιδιαίτερα παρά μόνο το CT, το οποίο επωφελείται ουσιαστικά από τον περιορισμό της βαρύτητας του χρόνου εκτέλεσης, όταν η παράμετρος λ τίθεται ίση με 0.5.

Στη συνέχεια παραθέτουμε κάποιες γραφικές για να εξηγήσουμε καλύτερα τη λειτουργία του πράκτορα και να καταδείξουμε τα αποτελέσματα που πετυχαίνει. Στην ει-

κόνα 5.4 βλέπουμε την επίδραση που έχει η λειτουργία του πράκτορα στη συγκράτηση εντός ορίων του tail latency. Οι εικόνες αφορούν τα BEs graph analytics και gradient boosted, ωστόσο παρόμοια είναι η εικόνα και για τα υπόλοιπα.



(a) Graph Analytics

(b) Gradient Boosted

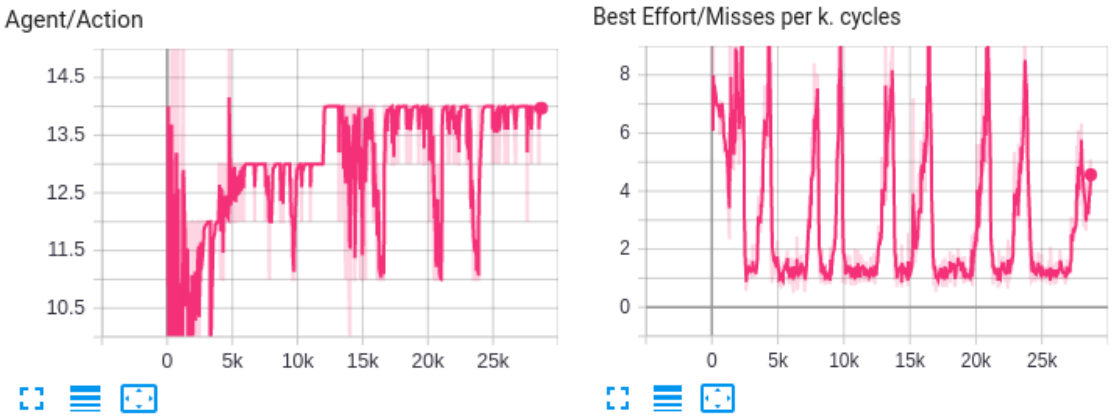
Σχήμα 5.4: Συγκράτηση του tail latency όταν χρησιμοποιείται ο πράκτορας έναντι της συνεκτέλεσης χωρίς περιορισμό, για τις εφαρμογές graph analytics και gradient-boosted

Επίσης, σε αντίθεση με τη στατική ανάθεση που παρουσιάστηκε στην ενότητα 2.4.2, οι φάσεις της εφαρμογής χαμηλής προτεραιότητας δε φαίνεται να αποτυπώνονται ιδιαίτερα στη μορφή του tail latency. Αυτό συμβαίνει διότι ο πράκτορας προσαρμόζει κάθε φορά την απόφασή του στις φάσεις των εφαρμογών χαμηλής προτεραιότητας αποσοβώντας έτσι τους κραδασμούς που αυτές δημιουργούν.

Η εικόνα 5.5 αποτυπώνει ακριβώς αυτό. Δεξιά βλέπουμε τα misses που πραγματοποιεί η εφαρμογή in-memory analytics και αριστερά την προσαρμογή των αποφάσεων του πράκτορα. Αύξηση των misses συνεπάγεται αύξηση του ανταγωνισμού , οπότε ο πράκτορας σπεύδει να μειώσει τα ways που διαθέτει στην εφαρμογή χαμηλής προτεραιότητας. Ένα άλλο σημείο που χρήζει σχολιασμού είναι πως η προσαρμογή αυτή, δε συμβαίνει άμεσα καθώς ο πράκτορας κατά τα πρώτα βήματα δρα ουσιαστικά στην τύχη και προοδευτικά αρχίζει να εκμεταλλεύεται τη γνώση που συσσωρεύει.

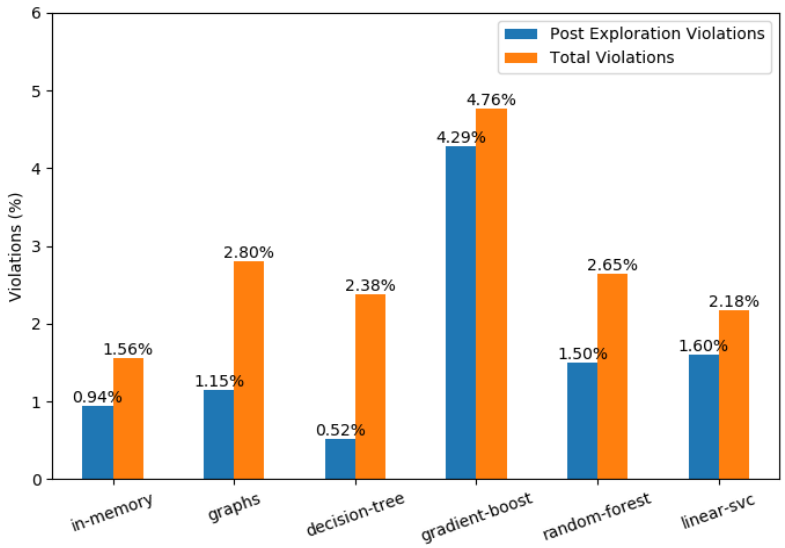
Λόγω αυτού, στον πίνακα 5.3 πέρα από το συνολικό ποσοστό παραβιάσεων που πετυχαίνει ο πράκτορας, διαχωρίζουμε την εκτέλεση και παρουσιάζουμε επιπλέον τις παραβιάσεις, αν εξαιρεθεί το διάστημα εξερεύνησης (Post Exploration Violations).

Γίνεται εύκολα αντιληπτό από την εικόνα 5.6, πως τα ποσοστά παραβίασης του ορίου, είναι σαφώς χαμηλότερα αν δεν λάβουμε υπόψιν μας τη φάση αυτή. Όμως η εξερεύνηση είναι αναγκαστικά μία διαδικασία από την οποία πρέπει να περάσει ο πράκτορας, προκειμένου να μάθει να δρα ορθά στο περιβάλλον. Στη συνέχεια επεξεργαζόμαστε τρόπους ώστε να περιορίσουμε το κόστος της εξερεύνησης.



(a) Agent's Actions (b) In-Memory Analytics Misses

Σχήμα 5.5: Απεικόνιση των φάσεων της εφαρμογής in-memory analytics (δεξιά) και της προσαρμογής των αποφάσεων του πράκτορα σε αυτές (αριστερά)



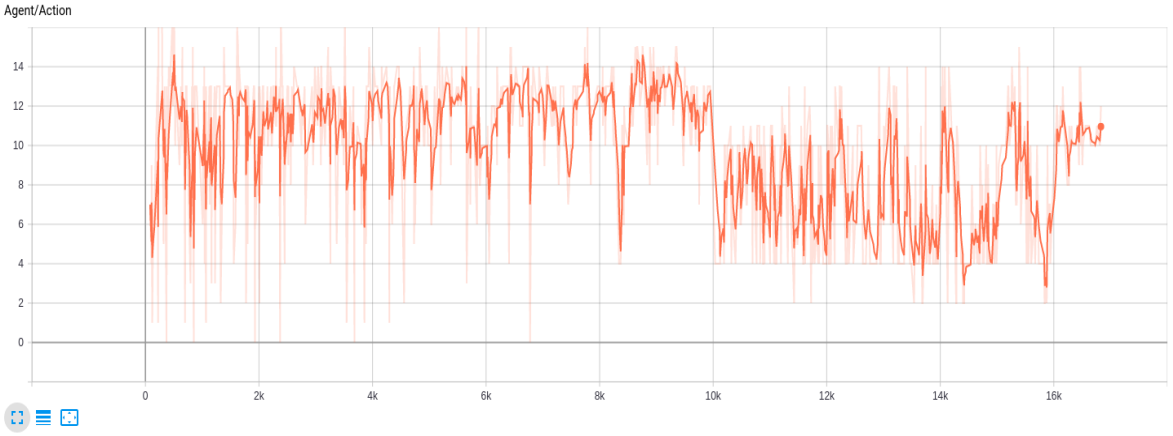
Σχήμα 5.6: Αντιπαραβολή του ποσοστού παραβιάσεων, μετά τη συμβατική λήξη του διαστήματος εξερεύνησης, με το συνολικό. Σε όλες τις συνεκτελέσεις καταγράφονται χαμηλότερες τιμές

5.3 Μεταφορά γνώσης

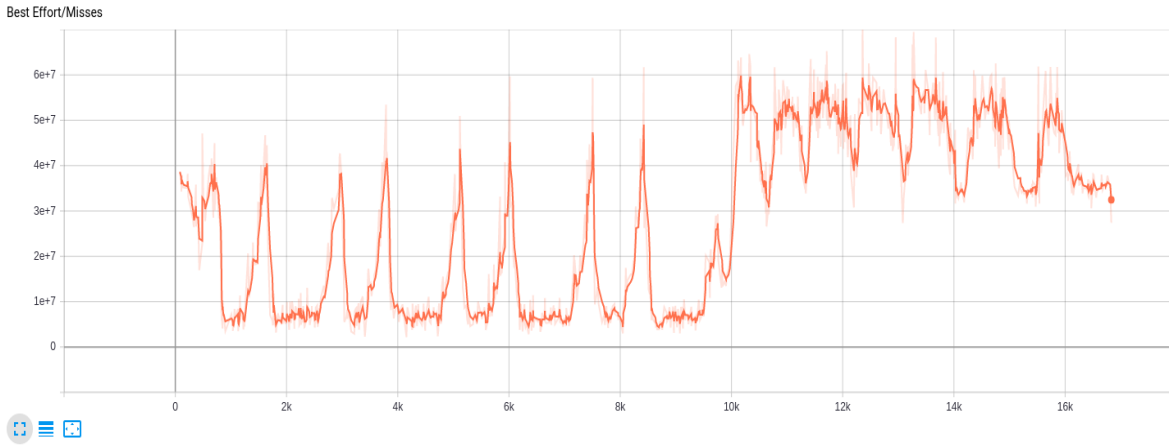
Κατά τη λειτουργία ενός κέντρου δεδομένων, θα συνεκτελεστούν διαδοχικά με την κρίσιμη υπηρεσία, πολλές διαφορετικές εφαρμογές χαμηλής προτεραιότητας. Γεννάται λοιπόν το ερώτημα αν χρειάζεται να εξερευνούμε από την αρχή κάθε συνεκτέλεση ή αν μπορούμε να εκμεταλλευτούμε κάποια πρότερη γνώση πάνω στο πρόβλημα.

Η λογική πίσω από τη συγκεκριμένη συλλογιστική είναι πως ο πράκτορας θέλουμε να μαθαίνει να κωδικοποιεί τον ανταγωνισμό που εμφανίζεται στο περιβάλλον και να δρα

ορθά βάση αυτού. Κατά την συνεκτέλεσή του, με μία εφαρμογή χαμηλής προτεραιότητας εκτίθεται στα πρότυπα ανταγωνισμού που αυτή δημιουργεί. Η γνώση που αποκτά ενδέχεται να είναι χρήσιμη για τη συνεκτέλεση με κάποια άλλη εφαρμογή, με παρεμφερή πρότυπα ανταγωνισμού. Έτσι λοιπόν αντί να ξεκινήσει την εξερεύνηση από το μηδέν, ο πράκτορας θα μπορούσε να εκμεταλλευτεί τη γνώση που έχει συσσωρεύσει και να συνεχίσει να μαθαίνει πάνω σε αυτή.



(a) Agent's Actions



(b) Misses

Σχήμα 5.7: Μεταφορά γνώσης. Περί τα 10 χιλιάδες βήματα ολοκληρώνεται η εκτέλεση της in-memory και ξεκινά η graphs analytics. Ο πράκτορας ανταποκρίνεται στην αλλαγή και βαθμιαία προσαρμόζεται στις φάσεις της νέας εφαρμογής

Στην εικόνα 5.7 παρουσιάζονται δύο διαδοχικές εκτελέσεις. Αρχικά ο πράκτορας εξερευνά το περιβάλλον συνεκτέλεσης με την εφαρμογή in-memory analytics και στα 10 χιλιάδες βήματα περίπου, αυτή ολοκληρώνεται και εκκινούν 9 στιγμιότυπα της εφαρμογής graphs analytics. Στην εικόνα 5.7(b) παρατηρούμε πως οι συμπεριφορές των δύο εφαρμογών, εν γένη διαφέρουν. Το ελάχιστο των misses της graph analytics κινείται στο ίδιο επίπεδο με το μέγιστο των misses της in-memory. Παρόλα αυτά, βλέπουμε πως

ο πράκτορας ανταποκρίνεται στην αλλαγή διαφοροποιώντας την περιοχή όπου κινούνται οι αποφάσεις του. Μάλιστα, καθώς αλληλεπιδρά περισσότερο με τη νέα εφαρμογή καταφέρνει να προσαρμοστεί και στις φάσεις αυτής.

Το αποτέλεσμα είναι πολύ ενθαρρυντικό και μας προέτρεψε να συνεχίσουμε να κινούμαστε σε αυτή την κατεύθυνση. Στην ιδανική περίπτωση λοιπόν, αν ο πράκτορας εκτιθόταν σε ένα μεγάλο πλήθος εφαρμογών, που καλύπτουν το σύνολο των προτύπων ανταγωνισμού, θα θέλαμε να μπορεί να ανταποκριθεί σε κάθε εφαρμογή χαμηλής προτεραιότητας που θα εμφανιζόταν κατά τη συνεκτέλεση.

Για το σκοπό αυτό, χωρίσαμε τις εφαρμογές χαμηλής προτεραιότητας, που έχουμε στη διάθεσή μας, σε δύο ομάδες. Εκπαιδεύσαμε τον πράκτορα στην πρώτη ομάδα, ενώ αξιολογήσαμε την επίδοσή του στις εφαρμογές της δεύτερης.

Η εκπαίδευση στην πρώτη ομάδα γίνεται διαδοχικά. Κάθε ένα στιγμιότυπο της εφαρμογής που ολοκληρώνεται αντικαθίσταται με ένα της επόμενης. Η εξερεύνηση σε εκείνο το σημείο ξεκινά και πάλι, προκειμένου να συγκεντρωθεί γνώση και για τη νέα εφαρμογή. Όταν τελειώσουν οι διαδοχικές εκτελέσεις, αποθηκεύουμε τα βάρη του νευρωνικού δικτύου, δημιουργώντας ένα checkpoint. Μπορούμε κατόπιν, να φορτώσουμε τα βάρη αυτά σε ένα νευρωνικό δίκτυο ίδιων προδιαγραφών και να συνεχίσουμε να χρησιμοποιούμε τον πράκτορα.

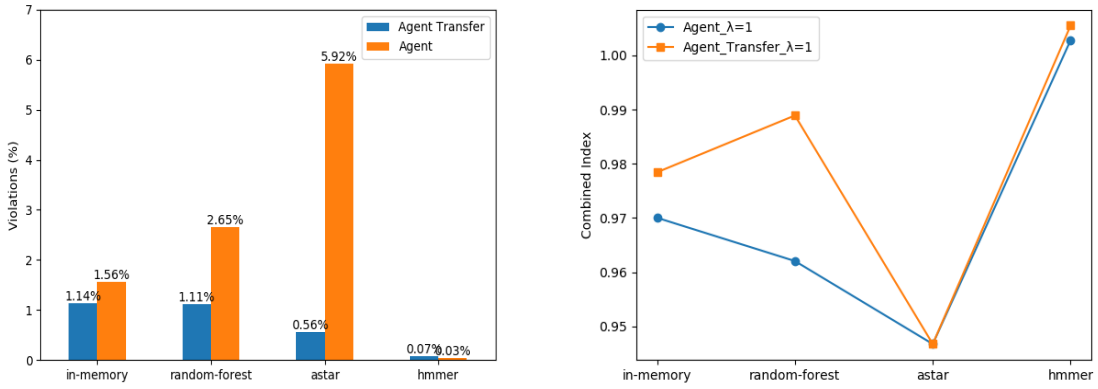
Αν θεωρήσουμε επαρκή την έκθεση του πράκτορα σε διαφορετικά πρότυπα, τότε μπορούμε να εκκινήσουμε τη διαδικασία της εξερεύνησης από πολύ μικρότερες τιμές, είτε αυτή μπορεί να διαρκέσει πολύ λιγότερο. Κατά τη δική μας αξιολόγηση, θέσαμε την αρχική πιθανότητα εξερεύνησης ίση με 1% έναντι 100% που χρησιμοποιούσαμε όταν δεν διαθέταμε απολύτως καμία γνώση για το πρόβλημα.

Η ομάδα εκπαίδευσης αποτελούνταν από 5 εφαρμογές ενώ η αξιολόγηση έγινε στις υπόλοιπες τέσσερις. Τα αποτελέσματα αυτά παρουσιάζονται στον πίνακα 5.4.

BE apps	Agent Transfer		Agent	
	Viol.	Time	Viol.	Time
in-memory	1.14%	53m27s	1.56%	53m41s
random-forest	1.11%	19m37s	2.65%	19m51s
astar	0.56%	5m34s	5.92%	5m16s
hmmer	0.07%	5m23s	0.03%	5m24s

Πίνακας 5.4: Αποτελέσματα μεταφοράς γνώσης. Η αξιολόγηση γίνεται σε τέσσερις εφαρμογές που δεν έχουν συμμετάσχει στην εκπαίδευση του πράκτορα

Βλέπουμε πως η μεταφορά γνώσης έχει θετικά αποτελέσματα στην περαιτέρω μείωση των παραβιάσεων. Τα ποσοστά κυμαίνονται κοντά ή ακόμα και χαμηλότερα από τα επίπεδα που καταγράψαμε προηγουμένως και αφορούσαν τις παραβιάσεις που συμβαίνουν μετά το πέρας της φάσης εξερεύνησης. Επιπλέον η μεταφορά γνώσης βοηθά και στις περιπτώσεις Best Effort εφαρμογών μικρής διάρκειας, όπως η astar, όπου η online



(a) Violations

(b) Combined Metric

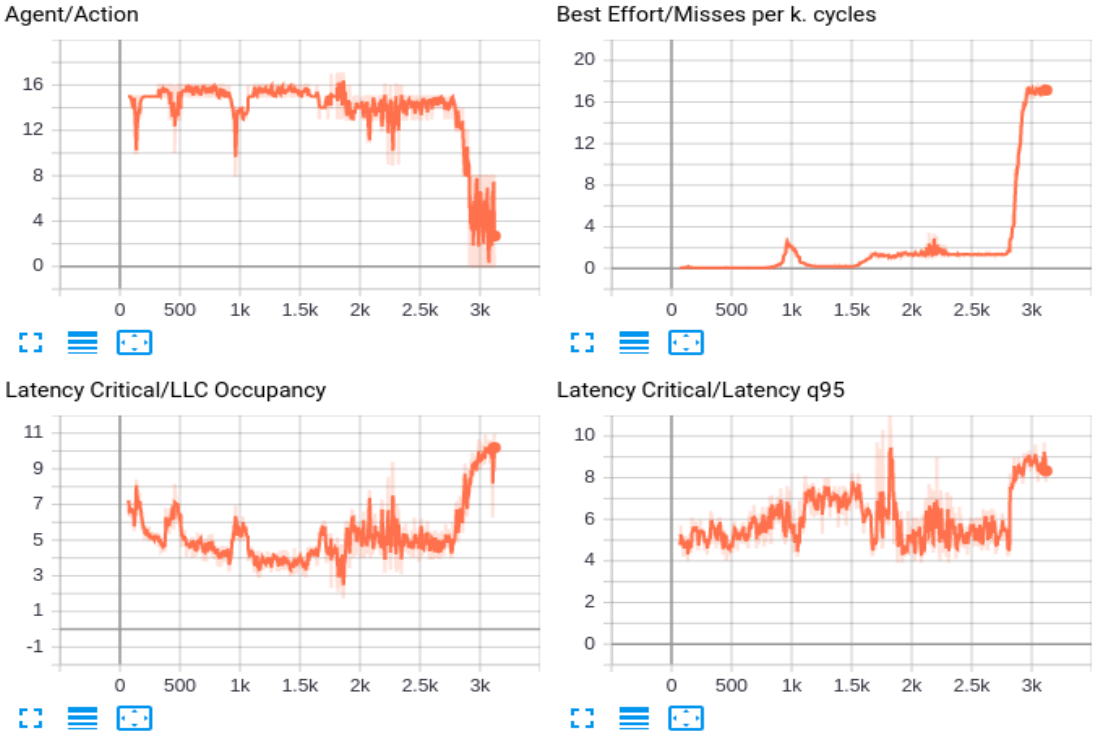
Σχήμα 5.8: Σύγκριση των επιδόσεων του πράκτορα με μεταφορά μάθησης και χωρίς. Σημειώνεται σημαντική μείωση του ποσοστού παραβιάσεων κάτι που αποτυπώνεται και στη συνδυαστική μετρική

χρήση του πράκτορα δεν προλαβαίνει να αποδώσει καρπούς. Τέλος, λειτουργεί καλύτερα και σε εφαρμογές με μη επαναλαμβανόμενες φάσεις, αφού η έκθεση σε πολλά πρότυπα μπορεί να καλύπτει κάποιες από αυτές τις φάσεις.

Σχήμα 5.9: Μετρικές της εφαρμογής in-memory analytics όταν ο πράκτορας αποφεύγει την εξερεύνηση και λαμβάνει αποφάσεις βασιζόμενος σε πρότερη γνώση που έχει αποκτήσει κατά την offline εκπαίδευσή του

Στην εικόνα 5.9, που αφορά χρήση μεταφοράς γνώσης στη συνεκτέλεση με την εφαρμογή in-memory analytics, βλέπουμε πως σε αντίθεση με την εικόνα 5.5 η προσαρμογή των αποφάσεων του πράκτορα στις φάσεις της εφαρμογής είναι πολύ πιο άμεση, αφού δεν ξοδεύονται αποφάσεις για την εξερεύνηση της συνεκτέλεσης.

Από την άλλη στην εικόνα 5.10 απεικονίζονται τα αντίστοιχα μεγέθη για την εφαρμογή astar, στην οποία ο πράκτορας κατά την online χρήση του, δυσκολευόταν να αντεπεξέλθει. Αυτό οφειλόταν και στη σύντομη διάρκειά της αλλά και στην απότομη μεταβολή των misses που εμφανίζει στο τέλος της. Πλέον βλέπουμε, πως ο πράκτορας καταφέρνει να διαφοροποιήσει την απόφασή του στην απότομη μεταβολή, παρόλο που δεν έχει αλληλεπιδράσει ξανά με τη συγκεκριμένη εφαρμογή. Με την αντίδρασή του αυτή, προλαβαίνει να κρατήσει εντός ορίων το latency της κρίσιμης υπηρεσίας.



Σχήμα 5.10: Μετρικές της εφαρμογής astar όταν γίνεται χρήση μεταφοράς γνώσης. Ο πράκτορας αντιδρά στην απρόβλεπτη αλλαγή φάσης της εφαρμογής και προλαβαίνει την αύξηση του latency, προτού αυτή ξεπεράσει το όριο

Τα παραπάνω αποτελέσματα καταδεικνύουν, ότι η γνώση που αποκομίζει ο πράκτορας, από συνεκτελέσεις στις οποίες έχει εκτεθεί, συμβάλλει θετικά στην προσπάθειά του να αλληλεπιδράσει με νέες εφαρμογές. Έτσι ένα σύστημα διαμοιρασμού της κρυφής μνήμης μπορεί να εκπαιδευτεί offline, σε ένα σύνολο εφαρμογών χαμηλής προτεραιότητας που θεωρείται αντιπροσωπευτικό και κατόπιν να χρησιμοποιηθεί online σε νέες συνεκτελέσεις.

5.4 Πειραματικές Μελέτες

Στη συνέχεια αυτού του κεφαλαίου, παρουσιάζουμε κάποιες μελέτες που κάναμε, προκειμένου να καταλήξουμε στην τελική μορφή του αλγορίθμου και των παραμέτρων αυτού. Οι μελέτες αυτές έγιναν σε δύο από τις εφαρμογές χαμηλής προτεραιότητας. Ο λόγος για αυτό είναι ότι στην κανονική λειτουργία του, ένας τέτοιος πράκτορας, θα κληθεί να αλληλεπιδράσει με ένα τεράστιο πλήθος διαφορετικών best effort εφαρμογών.

Είναι πιθανό το σύνολο των παραμέτρων που έχει επιλεγεί να μην λειτουργεί αποτελεσματικά για όλες αυτές, αλλά παρά μόνο για τις εφαρμογές που συμμετείχαν στην επιλογή. Έτσι λοιπόν κρατήσαμε το σύνολο των εφαρμογών επιλογής παραμέτρων μικρό, ώστε να δούμε πως ανταποκρίνεται ο πράκτορας και σε εφαρμογές με τις οποίες δεν έχει ξανά αλληλεπιδράσει.

Από τα αποτελέσματα του πίνακα 5.3, μπορούμε να πούμε πως πετυχαίνουμε το στόχο μας στην επιλογή παραμέτρων, καθώς βλέπουμε παρόμοιες τιμές και για τις εφαρμογές που δε συμμετείχαν σε αυτή. Μικρή εξαίρεση αποτελούν οι εφαρμογές μικρής διάρκειας για τις οποίες, η αναλογία εξερεύνησης εκμετάλλευσης που έχουμε επιλέξει δεν είναι ταιριαστή. Ωστόσο η φύση της προσέγγισης που μελετούμε απαιτεί έναν σημαντικό βαθμό αλληλεπίδρασης με το περιβάλλον και έτσι επιλέξαμε να αντιμετωπίσουμε αυτό το ζήτημα με χρήση της μεταφοράς γνώσης.

5.4.1 Μελέτη της συνεισφοράς των βελτιστοποιήσεων του αλγορίθμου DQN

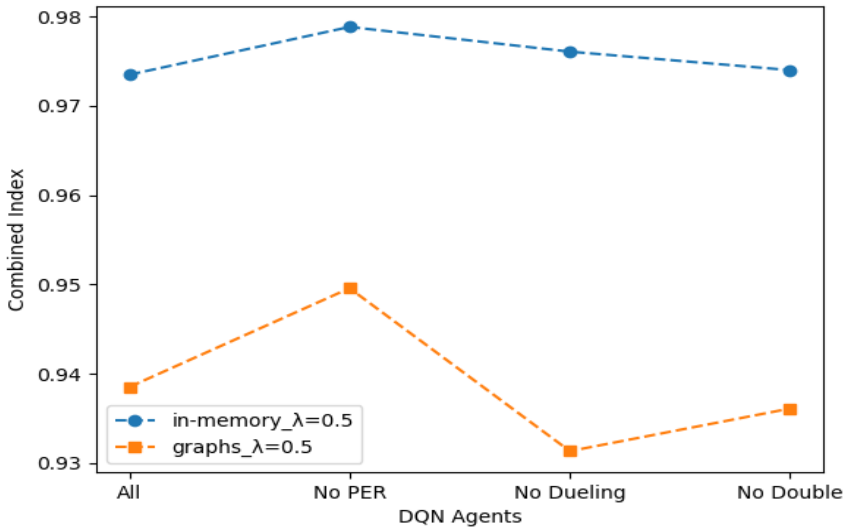
Όπως αναφέραμε στην ενότητα 3.5, υλοποιούμε τρεις βασικές βελτιστοποιήσεις του αλγορίθμου DQN. Στόχος μας εδώ, είναι να δούμε τη συνεισφορά κάθε μιας στο συγκεκριμένο πρόβλημα. Συνεπώς εκτελούμε τον πράκτορα, απενεργοποιώντας κάθε φορά μία από τις βελτιστοποιήσεις και συγκρίνουμε με το μοντέλο που τις περιλαμβάνει όλες.

BE apps	All		No Per		No Dueling		No Double	
	Viol.	Time	Viol.	Time	Viol.	Time	Viol.	Time
in-memory	2,1%	53m30s	1,7%	53m21s	2,3%	53m	2.4%	53m7s
graphs	4,7%	35m56s	3.8%	35m46s	5,1%	36m11s	4,9%	35m58s

Πίνακας 5.5: Αποτελέσματα συνεκτελέσεων απενεργοποιώντας διαδοχικά μία εκ των βελτιστοποιήσεων του αλγορίθμου DQN, με στόχο τη μελέτη της συνεισφοράς που έχει η καθεμιά

Από τις παραπάνω εκτελέσεις βλέπουμε πως η μη χρησιμοποίηση δύο νευρωνικών δικτύων (Double DQN), ενός για την λήψη απόφασης και ενός για την εκτίμηση της αξίας αυτής, επιφέρει μεγάλη επιδείνωση στην απόδοση του πράκτορα. Όπως αναφέραμε στην ενότητα 3.5.3 η σύμπλεξη απόφασης και εκτίμησης υπερεκτιμά την αξία της κατεύθυνσης στην οποία κινείται ο πράκτορας.

Στη συνέχεια, χειρότερα είναι και τα αποτελέσματα, αν δεν χρησιμοποιηθεί η αρχιτεκτονική Dueling (3.5.4), η οποία αποπλέκει την εκτίμηση της αξίας καταστάσεων και αξιών. Τέλος, καλύτερα παρουσιάζονται τα αποτελέσματα όταν δεν συμμετέχει στο μοντέλο η βελτιστοποίηση PER (Prioritized Experience Replay 3.5.5). Παρόλο που δοκιμάστηκαν διάφορες παράμετροι για το μηχανισμό αυτό, το αποτέλεσμα δεν είναι ενθαρρυντικό, οπότε στις τελικές δοκιμές μας δεν συμπεριλάβαμε αυτή τη βελτιστοποίηση. Η παραπάνω ανάλυση τεκμηριώνεται και από την εικόνα 5.11, όπου παρουσιάζουμε τη συνδυαστική μετρική για λ=0.5, δηλαδή δίδοντας έμφαση στο ποσοστό παραβίασης του QoS. Κάτι τέτοιο είναι συμβατό και με τη συνάρτηση ανταμοιβής του πράκτορα, αφού τιμωρείται διπλά κάθε παραβίαση του ορίου.



Σχήμα 5.11: Απεικόνιση της συνδυαστικής μετρικής για τις παραλλαγές του DQN αλγορίθμου. Παρουσιάζονται οι τιμές για τα BEs in-memory και graph analytics

5.4.2 Μελέτη της ευαισθησίας στο χρονικό παράθυρο απόφασης

Το χρονικό παράθυρο στο οποίο ο πράκτορας λαμβάνει την απόφασή του, αποτελεί σημαντική παράμετρο για τη λειτουργία του. Από τη μία, ένα μεγαλύτερο παράθυρο θα προσέφερε υψηλότερη αξιοπιστία στις τιμές που συλλέγονται, από την άλλη διατρέχουμε τον κίνδυνο η υπηρεσία να μείνει για αρκετό χρόνο σε κατάσταση όπου παραβιάζει τα όρια που έχουν τεθεί.

Επιπλέον, γνωρίζουμε πως για να εφαρμοστεί από το PQOS η απόφαση που λαμβάνει ο agent, απαιτείται τουλάχιστον ένα διάστημα μερικών millseconds. Άρα πιθανώς να προσμετρηθούν στη μέτρηση τιμές, που αφορούν την προηγούμενη απόφαση. Όσο μι-

κρότερο είναι το διάστημα που χρησιμοποιούμε τόσο μεγαλύτερη θα είναι η συνεισφορά αυτών των τιμών. Ακόμα, σε ένα μικρό διάστημα αντιστοιχεί και μικρότερο πλήθος αιτήσεων άρα στην εκτίμηση της τιμής του latency μπορεί να συμμετέχει ένας ανεπαρκής αριθμός απαντήσεων. Παρόλα αυτά δε διατρέχουμε τέτοιο κίνδυνο στην περίπτωση της Memcached υπηρεσίας καθώς αποστέλλεται ένα τεράστιο πλήθος μετρήσεων ανά δευτερόλεπτο (90k/sec).

Πέρα όμως από τα ζητήματα αυτά, η επιλογή του διαστήματος επηρεάζει και τη λειτουργία του πράκτορα. Είναι γνωστό πως τα νευρωνικά δίκτυα απαιτούν ένα πολύ μεγάλο πλήθος δεδομένων για την εκπαίδευσή τους. Ιδιαίτερα στην βαθιά ενισχυτική μάθηση, οι περισσότερες εργασίες που έχουν παρουσιαστεί, αφορούν προβλήματα προσομοίωσης, για παράδειγμα βιντεοπαιχνίδια Atari, όπου η αλληλεπίδραση με το περιβάλλον είναι ακαριαία. Έτσι συλλέγεται γρήγορα ένα τεράστιο πλήθος δεδομένων, κάτι που σε περιβάλλοντα πραγματικού κόσμου είναι πολλές φορές μη εφικτό.

Με βάση τις περιγραφές των αλγορίθμων που έχουμε δώσει στην ενότητα 3.5 σε κάθε βήμα αλληλεπίδρασης με το περιβάλλον, συλλέγουμε ένα νέο δείγμα, αλλά ταυτόχρονα εκτελούμε και ένα βήμα βελτιστοποίησης. Η επιλογή ενός μεγαλύτερου διαστήματος συνεπάγεται μείωση και των συλλεγόμενων δειγμάτων και των βημάτων βελτιστοποίησης, κάτι που περιμένουμε να επηρεάσει αρνητικά τη διαδικασία της εκπαίδευσης.

Για να εντοπίσουμε το καλύτερο διάστημα λήψης απόφασης, εκτελέσαμε τον πράκτορα σε συνεκτελέσεις με δύο διαφορετικές εφαρμογές χαμηλής προτεραιότητας και χρόνους όπου πέφτει σε sleep 50, 100 και 200ms. Το πραγματικό διάστημα απόφασης θα είναι κάθε φορά λίγο μεγαλύτερο καθώς πέρα από το χρόνο που 'κοιμάται' προκειμένου να συλλέξει μετρήσεις, πρέπει να λάβει την απόφαση του, με χρήση του νευρωνικού δικτύου (inference), αλλά και να πραγματοποιήσει ένα βήμα βελτιστοποίησης (backward pass).

Σε μετρήσεις που κάναμε βρήκαμε ότι το διάστημα του inference είναι αμελητέο (απαιτείται λιγότερο από 1ms), ενώ για την εκπαίδευση, χρειαζόταν ένα σαφώς μεγαλύτερο διάστημα κοντά στα 50ms. Παρόλα αυτά, η εκπαίδευση του νευρωνικού δικτύου θα μπορούσε να συμβεί παράλληλα με το διάστημα που αναμένουμε τη συλλογή το στατιστικών, καθώς δεν υπάρχει εξάρτηση μεταξύ των δύο λειτουργιών.

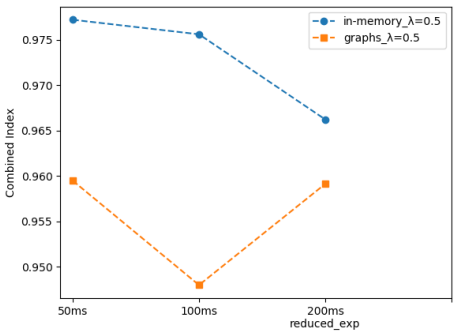
BE apps	50ms		100ms		200ms	
	Viol.	Time	Viol.	Time	Viol.	Time
in-memory	1.56%	53m41s	1.63%	53m47s	2.47%	53m54s
graphs	2.8%	35m46s	3.34%	36m14s	3.04%	35m37s

Πίνακας 5.6: Επίδοση του πράκτορα όταν χρησιμοποιείται διαφορετικό διάστημα απόφασης

Παρατηρούμε στον πίνακα 5.6, ότι οι χρόνοι ολοκλήρωσης των εφαρμογών είναι παρεμφερείς σε όλες τις περιπτώσεις. Αντίθετα μικρότερα παρουσιάζονται τα ποσοστά

παραβίασης για το διάστημα των 50ms, ενώ για τα 100 και τα 200ms η απόδοση του πράκτορα αρχίζει να πλήττεται.

Φαίνεται λοιπόν ο πράκτορας να μην επηρεάζεται τόσο από ενδεχόμενη αστάθεια των μετρήσεων για μικρά διαστήματα απόφασης. Είναι γνωστό εξάλλου, πως η ύπαρξη θορύβου στα δεδομένα επιδρά θετικά στη διαδικασίας της μάθησης, καθώς επιτρέπει στα νευρωνικά δίκτυα να γενικεύουν, αντί να υπερπροσαρμόζονται (overfitting) σε συγκεκριμένες τιμές. Μάλιστα σε πολλές περιπτώσεις προστίθεται τεχνητά θόρυβος στα δεδομένα.



Σχήμα 5.12: Συνδυαστική Μετρική για λ=0.5

Επιπλέον, το όποιο ενδεχόμενο όφελος ίσως υπάρχει από την μείωση του θορύβου, φαίνεται να αντισταθμίζεται πλήρως από το γεγονός πως επιλέγοντας ένα μεγαλύτερο διάστημα απόφασης, μειώνονται τα βήματα βελτιστοποίησης που πραγματοποιούμε. Η επίδραση είναι πιο ορατή στο spark job graph analytics λόγω της μικρότερης διάρκειας αυτού, καθώς με τη μείωση των βημάτων, μεγαλύτερο ποσοστό του χρόνου εκτέλεσης ξοδεύεται στην εξερεύνηση. Μάλιστα στην περίπτωση των 200ms για να είναι αντιπροσωπευτική η σύγκριση, μειώσαμε στο μισό το διάστημα της εξερεύνησης, κάτι που όπως φαίνεται στην εικόνα 5.12, όπου απεικονίζεται η συνδυαστική μετρική, έδρασε θετικά για το BE graphs.

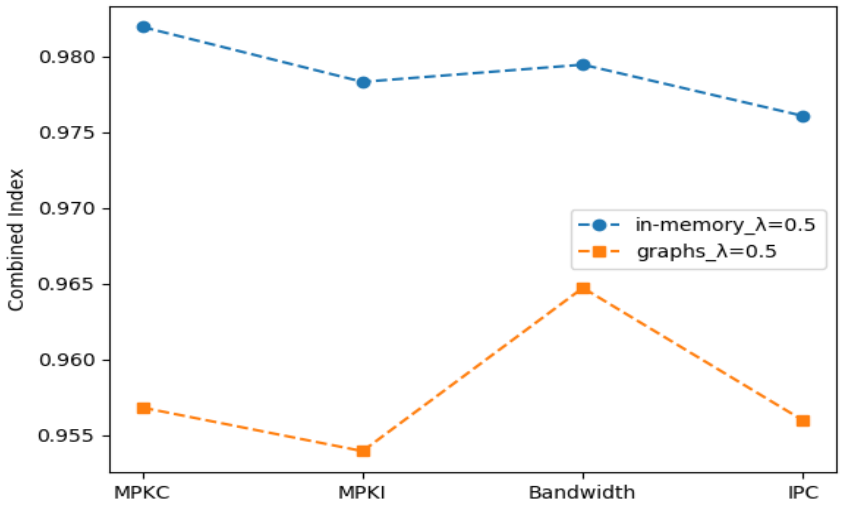
5.4.3 Μελέτη συνεισφοράς διαφορετικών μετρικών

Σε αυτή τη μελέτη, πειραματιζόμαστε με τις διάφορες μετρικές που έχουμε στη διάθεσή μας. Στόχος μας ήταν να αποφανθούμε για το ποια πρέπει να δοθεί σαν είσοδος στον αλγόριθμό μας. Όπως έχουμε αναφέρει και στην ενότητα 4.6, τα πρότυπα που εμφανίζονται στις μετρικές Misses, IPC και Bandwidth έχουν μεγάλο βαθμό ομοιότητας.

Συμπεριλαμβάνουμε κάθε φορά στο μοντέλο μας μία από αυτές τις μετρικές, μαζί με τα ways που διατίθενται εκείνη τη στιγμή για τις εφαρμογές χαμηλής προτεραιότητας. Όσο αφορά τα Misses, κανονικοποιούμε είτε ως προς το χρόνο (Misses per kilo cycles, MPKC), είτε ως προς το πλήθος των εντολών που εκτελέστηκαν (Misses per kilo instructions, MPKI).

BE apps	MPKC		MPKI		IPC		Bandwidth	
	Viol.	Time	Viol.	Time	Viol.	Time	Viol.	Time
in-memory	1,36%	53m23s	1,77%	53m20s	1.61%	53m23s	1.95%	53m23s
graphs	2,82%	35m57s	2.91%	36m06s	2.2%	35m49s	3.29%	35m40s

Πίνακας 5.7: Επίδραση διαφορετικών μετρικών στην απόδοση του αλγορίθμου

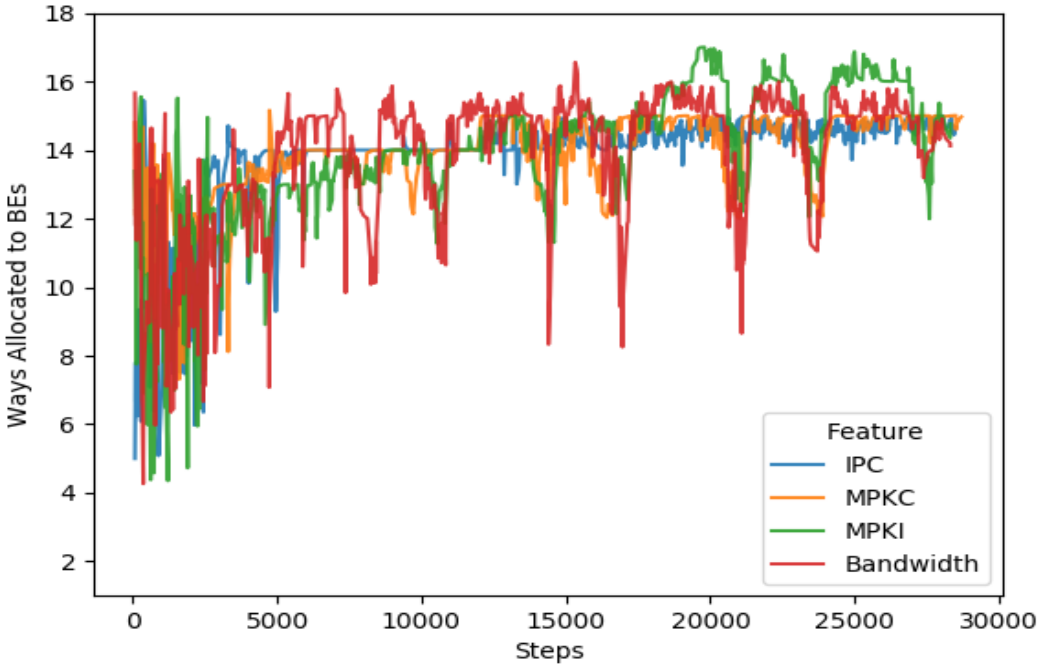


Σχήμα 5.13: Απεικόνιση του combined metric για τις εκδοχές του πράκτορα, όπου χρησιμοποιείται διαφορετική μετρική σαν είσοδος

Από τα αποτελέσματα του πίνακα 5.7 και της εικόνας 5.13 βλέπουμε, πως δεν υπάρχει κάποια μετρική που να ξεχωρίζει. Την καλύτερη επίδοση ως προς τις παραβιάσεις του QoS για τη συνεκτέλεση με την εφαρμογή in-memory, την πετυχαίνει ο πράκτορας χρησιμοποιώντας τη μετρική MPKC, ενώ για τη συνεκτέλεση με την BE graphs καλύτερη φαίνεται να είναι η χρήση του Bandwidth. Ως προς τους χρόνους ολοκλήρωσης τα αποτελέσματα κινούνται σε πανομοιότυπα επίπεδα. Τέλος, όσο αφορά τη μετρική MPKI, η μικρή υστέρησή της σε σχέση με την MPKC ίσως οφείλεται στο γεγονός, ότι εμπλέκει τα misses με τις εντολές που έχουν εκτελεστεί.

Στην εικόνα 5.14 παρουσιάζουμε τις αποφάσεις που λαμβάνει ο πράκτορας, καθώς χρησιμοποιεί διαφορετικές μετρικές σε συνεκτελέσεις με την εφαρμογή in-memory analytics. Έτσι έχουμε την ευκαιρία να δούμε πως προσαρμόζεται ο πράκτορας στις φάσεις της εφαρμογής, με χρήση κάθε διαφορετικής μετρικής. Παρατηρούμε ότι έχουμε την πιο σταθερή συμπεριφορά όταν χρησιμοποιείται το IPC. Ο ανταγωνισμός στην κύρια μνήμη αντικατοπτρίζεται έμμεσα σε αυτή τη μετρική. Όταν αυξάνεται ο αριθμός των Misses μειώνεται το IPC, καθώς καταναλώνονται κύκλοι για τον ερχομό των δεδομένων από την κύρια μνήμη. Ωστόσο το IPC διαφοροποιείται και λόγω των υπολογιστικών φάσεων της εφαρμογής και πιθανώς για αυτό το λόγο δε προσαρμόζονται τόσο οι δράσεις του πράκτορα στη μετρική αυτή.

Από την άλλη, έντονες διαφοροποιήσεις παρουσιάζει η μετρική MPKI. Ίσως κάτι τέτοιο οφείλεται στο ότι, ενδεχόμενη αύξηση των misses μειώνει και τις εντολές που εκτελούνται με αποτέλεσμα η μετρική να αυξάνεται και για τους δύο λόγους. Εικάζουμε ότι αυτό μπορεί να οδηγήσει σε υπερεκτίμηση ή, στην αντίθετη περίπτωση που έχουμε μείωση των misses, σε υποεκτίμηση του ανταγωνισμού για κρυφή μνήμη. Τέλος, έντονες διαφοροποιήσεις παρουσιάζει και το bandwidth ενώ ηπιότερες είναι αυτές που εμφανίζει



Σχήμα 5.14: Διαφοροποίηση της συμπεριφοράς του πράκτορα, κατά τη συνεκτέλεση με την εφαρμογή χαμηλής προτεραιότητας in-memory analytics, όταν δίνονται ως είσοδοι διαφορετικές μετρικές

η μετρική MPKC.

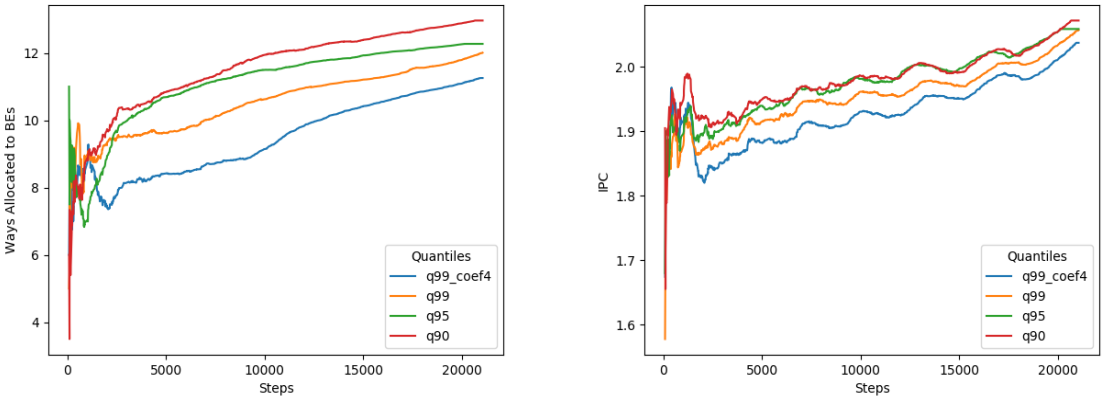
5.4.4 Μελέτη της συμπεριφοράς του πράκτορα σε διαφορετικούς στόχους

Κατά την περιγραφή του συστήματος, στην ενότητα 4.1, αναφέραμε ότι διαφορετικοί στόχοι μπορούν να τεθούν, ανάλογα με τις ανάγκες της κάθε υπηρεσίας. Ερχόμαστε λοιπόν να μελετήσουμε εδώ τη συμπεριφορά του πράκτορα, όταν καλείται να εξυπηρετήσει κάτω από το ίδιο όριο (10ms), διαφορετικό ποσοστό των αιτήσεων που λαμβάνει. Συγκεκριμένα τίθενται ως στόχοι τα εκατοστημόρια q90, q95, q99.

BE apps	q90		q95		q99	
	Viol.	Time	Viol.	Time	Viol.	Time
in-memory	1.41%	53m01s	1.70%	53m21s	2.46%	53m37s
graphs	3.48%	35m40s	3.76%	35m46s	6.69%	36m25s

Πίνακας 5.8: Συγκεντρωτικά αποτελέσματα της εκτέλεσης του πράκτορα με διαφορετικούς στόχους. Παρουσιάζεται το ποσοστό παραβιάσεων και ο χρόνος ολοκλήρωσης για τα εκατοστημόρια q90, q95 και q99

Όπως είναι αναμενόμενο, όσο αυξάνεται το εκατοστημόριο που μας ενδιαφέρει τόσο δυσκολεύεται ο πράκτορας να πετύχει το στόχο του. Ωστόσο όπως βλέπουμε στην εικόνα 5.15(a) ο πράκτορας προσαρμόζεται κάθε φορά στις διαφορετικές συνθήκες του περιβάλλοντος και αποδίδει λιγότερα ways στις εφαρμογές χαμηλής προτεραιότητας προκειμένου να μειώσει τις παραβιάσεις του ορίου.



(a) Agent's Actions Smoothed

(b) IPC of BEs Smoothed

Σχήμα 5.15: Διαφοροποίηση των δράσεων του πράκτορα και του IPC, ανάλογα με το εκατοστημόριο που τίθεται ως στόχος. Απεικονίζονται οι μετρικές για το job Graph Analytics και για τα εκατοστημόρια q90, q95, q99, και q99 με διπλάσιο συντελεστή ποινής

Εκεί οφείλεται και η αύξηση της διάρκειας εκτέλεσης, η οποία παρατηρείται καθώς αυξάνεται το εκατοστημόριο που προσπαθούμε να καλύψουμε. Η μείωση των αποδιδόμενων ways επιφέρει ανάλογη πτώση στο IPC των εφαρμογών χαμηλής προτεραιότητας, όπως φαίνεται στην εικόνα 5.15(b). Να σημειώσουμε, ότι στο σχήμα αυτό έχουμε πραγματοποιήσει εκθετική ομαλοποίηση, προκειμένου να καταδείξουμε την γενική εικόνα που διέπει τις εκτελέσεις αυτές.

5.4.5 Επίδραση του συντελεστή ποινής στη συνάρτηση ανταμοιβής

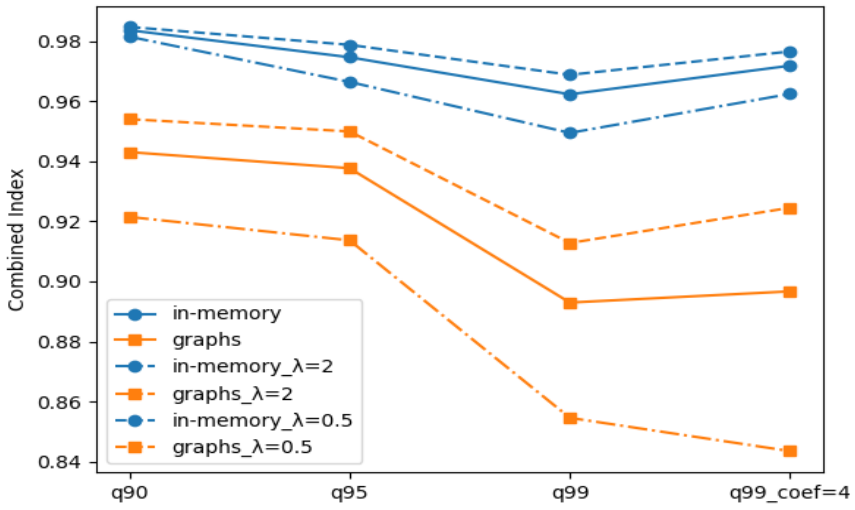
Ιδιαίτερα τώρα για το εκατοστημόριο 99, φαίνεται η αύξηση των παραβιάσεων να είναι σημαντική. Σε αυτή την περίπτωση θα εξετάσουμε την επίδραση της παραμέτρου ποινής στη συνάρτηση ανταμοιβής. Όπως αναφέραμε στην ενότητα 4.6, η παράμετρος αυτή επηρεάζει την ανοχή του πράκτορα στις παραβιάσεις και τον βοηθά να σταθμίσει τις αποφάσεις του, μεταξύ βραχυπρόθεσμων και μακροπροθέσμων ανταμοιβών.

Στον πίνακα 5.9 βλέπουμε πως πράγματι ο διπλασιασμός της παραμέτρου ποινής κατάφερε να μειώσει σημαντικά τα αυξημένα ποσοστά παραβιάσεων που σημειώθηκαν για το εκατοστημόριο q99. Ιδιαίτερα για την εφαρμογή graph analytics που είναι και

BE apps	q99, penatly coef=4		q99, penatly coef=2	
	Viol.	Time	Viol.	Time
in-memory	1.86%	53m25s	2.46%	53m37s
graphs	4.67%	37m3s	6.69%	36m25s

Πίνακας 5.9: Επίδοση του πράκτορα, όταν ως στόχος τίθεται το εκατοστημόριο q99 και η παράμετρος ποινής διπλασιάζεται

η πιο επιθετική από τις δύο, το ποσοστό παραβιάσεων μειώθηκε κατά δύο ποσοστιαίες μονάδες πλησιάζοντας το αντίστοιχο ποσοστό για το εκατοστημόριο q95%. Η βελτίωση αυτή είχε ως αποτέλεσμα την αύξηση του χρόνου ολοκλήρωσης του πειράματος αφού ο πράκτορας αναθέτει ακόμα λιγότερα ways στις εφαρμογές χαμηλής προτεραιότητας όπως φαίνεται και στην εικόνα 5.15. Το αποτέλεσμα είναι σημαντικό καθώς καταδεικνύει πως ο πράκτορας με κατάλληλη ρύθμιση των παραμέτρων του, μπορεί να προσαρμοστεί σε διαφορετικές προδιαγραφές που τίθενται.



Σχήμα 5.16: Απεικόνιση της συνδυαστικής μετρικής καθώς ο στόχος του πράκτορα δυσκολεύει. Παρουσιάζονται οι τιμές για τα BEs in-memory και graph analytics και για τιμές του λ: 0.5, 1 και 2

Τέλος, η αυξανόμενη δυσκολία του στόχου αποτυπώνεται και στις τιμές του συνδυαστικού συντελεστή όπως απεικονίζονται στην εικόνα 5.16. Για όλες τις τιμές του λ παρατηρούμε ότι η απόδοση πλήττεται. Ωστόσο πιο απότομη είναι η πτώση όταν λαμβάνεται περισσότερο υπόψιν ο χρόνος ολοκλήρωσης (λ=2).

Κάτι τέτοιο είναι λογικό μιας και ο στόχος του πράκτορα, όπως εκφράζεται από τη συνάρτηση ανταμοιβής αυτού, είναι πρώτα να προστατεύσει το QoS και έπειτα να διαθέσει ways στις BE εφαρμογές. Θυμίζουμε ότι τιμωρείται διπλά (penalty coefficient = 2)

όταν υπάρχει παραβίαση ορίου. Όσο αφορά τον περαιτέρω διπλασιασμό της παραμέτρου ποινής βλέπουμε, στην περίπτωση του BE graphs, πως αυτός συνεπάγεται περιορισμό των απωλειών του combined metric, όταν σταθμίζεται περισσότερο το ποσοστό παραβιάσεων ενώ επιδείνωση αυτού, όταν μας ενδιαφέρει κυρίως ο ρυθμός διεκπεραίωσης των εφαρμογών χαμηλής προτεραιότητας.

Κεφάλαιο 6

Επίλογος

6.1 Σύνοψη και Συμπεράσματα

Στην παρούσα διπλωματική εργασία, αξιοποιώντας τεχνικές βαθιάς ενισχυτικής μάθησης, υλοποιούμε έναν πράκτορα λογισμικού, ο οποίος αλληλεπιδρώντας με ένα περιβάλλον συνεκτέλεσης, μαθαίνει να προστατεύει την υπηρεσία κρίσιμης απόκρισης αλλά και να αυξάνει τη χρησιμοποίηση του επεξεργαστή, διαθέτοντας πόρους στις χαμηλής προτεραιότητας εφαρμογές.

Ο μηχανισμός βασίζεται στις τεχνολογίες επίβλεψης και καταμερισμού της κρυφής μνήμης δεδομένων της Intel, αλλά αξιοποιεί και μετρικές λογισμικού όπως το tail latency που εμφανίζει η υπηρεσία, ώστε να λαμβάνει κάθε στιγμή απόφαση για το διαμοιρασμό της LLC. Ακόμα για τις ανάγκες του προβλήματος δημιουργήθηκε ένα πολύπλοκο σύστημα με πολλά διαφορετικά τμήματα ώστε να διεκπεραιώνονται αποτελεσματικά όλες οι απαραίτητες λειτουργίες, όπως η συλλογή των μετρικών και η επιβολή των αποφάσεων.

Συμπερασματικά, μπορούμε να πούμε πως η βαθιά ενισχυτική μάθηση αποτελεί ένα σημαντικό εργαλείο για την αντιμετώπιση του εν λόγω προβλήματος. Καταδείξαμε ότι με αυτοματοποιημένο τρόπο, μπορούμε να μάθουμε τα πρότυπα που διέπουν τις συνεκτελέσεις και να δράσουμε βάση αυτών. Παρουσιάσαμε ότι είναι εφικτή η διατήρηση του επιπέδου λειτουργίας μιας κρίσιμης υπηρεσίας και η ταυτόχρονη ικανοποίηση ενός πλήθους εφαρμογών χαμηλής προτεραιότητας με πολύ μικρή επιβράδυνσή τους.

Δείξαμε ακόμα, ότι ο μηχανισμός μας μπορεί να προσαρμόζει τη λειτουργία του σε διαφορετικούς στόχους και προδιαγραφές με κατάλληλη ρύθμιση των παραμέτρων του, ισορροπώντας έτσι μεταξύ της προστασίας της υπηρεσίας κρίσιμης απόκρισης και του ρυθμού διεκπεραίωσης των εφαρμογών χαμηλής προτεραιότητας. Επιπλέον, είδαμε ότι πέρα από τις επιλογές που κάναμε, είναι σε θέση να λειτουργήσει αποτελεσματικά με διάφορες hardware μετρικές αλλά και για διαφορετικά διαστήματα απόφασης.

Τέλος προχωρήσαμε ένα βήμα πιο πέρα και παρουσιάσαμε τις δυνατότητες που προσφέρει η μάθηση νευρωνικών δικτύων για μεταφορά γνώσης. Καταφέραμε να εκπαιδεύσουμε ένα στιγμιότυπο του πράκτορα offline σε ένα ενδεικτικό πλήθος διαφορετικών εφαρμογών και να εκμεταλλευτούμε τη γνώση που απέκτησε σε νέες, προηγουμένως άγνωστες στον πράκτορα εφαρμογές. Φάνηκε ότι ο πράκτορας μπόρεσε να ανταποκριθεί επιτυχώς στις νέες εφαρμογές χωρίς την ανάγκη για επιπλέον εξερεύνηση των

συνεκτελέσεων αυτών, κάτι που προηγουμένως έθετε ένα άνω όριο στις επιδόσεις που μπορούσαμε να πετύχουμε.

Ακόμα, με αυτόν τον τρόπο η χρήση του πράκτορα είναι επωφελής και για εφαρμογές μικρής διάρκειας, στις οποίες προηγουμένως δεν είχε το χρόνο να μάθει να δρα αποτελεσματικά αλλά και σε εφαρμογές που παρουσιάζουν μεμονωμένες αναπάντεχες διαφοροποιήσεις στις φάσεις τους, όπου δυσκολευόταν να ανταποκριθεί, αφού ποτέ δεν είχε εκτεθεί σε παρόμοιες συμπεριφορές.

6.2 Επεκτάσεις

Η συγκεκριμένη διπλωματική εργασία θα μπορούσε να επεκταθεί σε διάφορες κατευθύνσεις. Αρχικά, η μελέτη μας εστιάζει στη διαχείριση του ανταγωνισμού, όπως αυτός δημιουργείται από τις διάφορες φάσεις των εφαρμογών χαμηλής προτεραιότητας. Μία άλλη πτυχή του προβλήματος αποτελεί και το φορτίο που δέχεται κάθε στιγμή η υπηρεσία υψηλής προτεραιότητας, κάτι που προφανώς επηρεάζει και τις ανάγκες της για κοινόχρηστους πόρους. Για το σκοπό αυτό απαιτείται η επέκταση του συστήματος με έναν loader που να μπορεί να δημιουργεί, συνθετικά και να προσομοιώνει πραγματικά, μοτίβα δικτυακής κίνησης. Απαιτείται ακόμα ο εμπλουτισμός της κατάστασης του περιβάλλοντος, ώστε να περιλαμβάνονται και μετρικές, είτε από το λογισμικό, είτε από το υλικό, που να αποτυπώνουν τη διαφοροποίηση στην κίνηση. Τέτοιες μπορεί να είναι το πλήθος των αιτήσεων που περιμένουν κάθε στιγμή στις ουρές της κρίσιμης υπηρεσίας ή το πλήθος των επιτυχημένων polls για νέα πακέτα που πραγματοποιεί ο επεξεργαστής προς την κάρτα δικτύου.

Συνεχίζοντας, άλλη μία πορεία επέκτασης θα ήταν η αξιολόγηση του μηχανισμού με περισσότερες υπηρεσίες κρίσιμης απόκρισης. Η δυσκολία για το συγκεκριμένο ζητούμενο έγκειται στο ότι δεν υπάρχουν αρκετές σουίτες για υπηρεσίες νέφους, που να παρέχουν επιπλέον τη δυνατότητα δημιουργίας φορτίου και συγκέντρωσης διαφόρων μετρικών. Επιπλέον, όσο αφορά το αλγοριθμικό σκέλος, η βαθιά ενισχυτική μάθηση, εξελίσσεται ταχύτατα τα τελευταία χρόνια με νέες προσεγγίσεις να έχουν αναπτυχθεί. Θα μπορούσαν λοιπόν να εφαρμοστούν και στο συγκεκριμένο πρόβλημα νέες μέθοδοι όπως είναι αυτή του Δράστη-Κριτή ([31]).

Ενδιαφέρον θα είχε ακόμη, η ολιστική διαχείριση του ζητήματος του ανταγωνισμού σε όλους τους κοινόχρηστους πόρους. Αξιοποιώντας την τεχνολογία MBA (Memory Bandwidth Allocation) της Intel, θα μπορούσε ο μηχανισμός να λαμβάνει αποφάσεις και για το bandwidth προς την κύρια μνήμη, που διατίθεται σε κάθε ομάδα πυρήνων. Κάτι τέτοιο ίσως αποδεικνυόταν κρίσιμο, ιδιαίτερα στις περιπτώσεις εκείνες όπου και η υπηρεσία έχει υψηλές απαιτήσεις για bandwidth. Η αρχιτεκτονική του νευρωνικού δικτύου θα χρειαζόταν να διαφοροποιηθεί προκειμένου να έχει πολλές εξόδους μία για κάθε πόρο, κατά τον τρόπο που περιγράφεται στην εργασία [4].

Τέλος, φαίνεται πολλά υποσχόμενη η περαιτέρω ενασχόληση με την τεχνική της

μεταφοράς γνώσης. Είδαμε ότι τα οφέλη που προσφέρει είναι σημαντικά και μπορεί να ξεκλειδώσει καταστάσεις, όπου είναι δύσκολη η online εφαρμογή της ενισχυτικής μάθησης. Ωστόσο, αν και σε άλλες περιοχές της μηχανικής μάθησης, όπως η αναγνώριση εικόνων, αυτή η τεχνική χρησιμοποιείται ευρέως, λίγα πράγματα έχουν συμβεί στο πεδίο της ενισχυτικής, χωρίς να υπάρχει ένα καθιερωμένο πλαίσιο χρήσης αυτής.

Βιβλιογραφία

[1] Konstantinos Nikas, Nikela Papadopoulou, Dimitra Giantsidi, Vasileios Karakostas, Georgios Goumas, and Nectarios Koziris. Dicer: Diligent cache partitioning for efficient workload consolidation. In *Proceedings of the 48th International Conference on Parallel Processing*, ICPP 2019, New York, NY, USA, 2019. Association for Computing Machinery.

[2] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Heracles: Improving resource efficiency at scale. *SIGARCH Comput. Archit. News*, 43(3S):450–462, June 2015.

[3] Yuhao Li, Dan Sun, and Benjamin C. Lee. Dynamic colocation policies with reinforcement learning. *ACM Trans. Archit. Code Optim.*, 17(1), March 2020.

[4] Rajiv Nishtala, Vinicius Petrucci, Paul Carpenter, and Magnus Själander. Twig: Multi-agent task management for colocated latency-critical cloud services. 12 2019.

[5] Intel resource director technology. `https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html`. Accessed: 2020-08-01.

[6] Laiping Zhao, Yanan Yang, Kaixuan Zhang, Xiaobo Zhou, Tie Qiu, Keqiu Li, and Yungang Bao. Rhythm: Component-distinguishable workload deployment in datacenters. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

[7] R. Nishtala, P. Carpenter, V. Petrucci, and X. Martorell. Hipster: Hybrid task manager for latency-critical cloud workloads. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 409–420, 2017.

[8] Memcached: high-performance, distributed memory object caching system. `http://memcached.org/about`. Accessed: 2020-08-14.

[9] Wikipedia cobra effect. `https://en.wikipedia.org/wiki/Cobra_effect`. Accessed: 2020-08-05.

[10] David Silver lecture 9: Exploration and exploitation. https://www.davidsilver.uk/wp-content/uploads/2020/03/XX.pdf. Accessed: 2020-08-03.

[11] Wikipedia bellman equation. https://en.wikipedia.org/wiki/Bellman_equation. Accessed: 2020-08-06.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. 12 2013.

[13] Wikipedia deepmind alphago. https://en.wikipedia.org/wiki/AlphaGo. Accessed: 2020-08-07.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[15] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. 09 2015.

[16] Ziyu Wang, Nando Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. 11 2015.

[17] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2016.

[18] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2012.

[19] Y. Zhang, D. Meisner, J. Mars, and L. Tang. Treadmill: Attributing the source of tail latency through precise load testing and statistical inference. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 456–468, 2016.

[20] Apache Spark mllib. https://spark.apache.org/mllib/. Accessed: 2020-08-08.

[21] UCI machine learning repository. https://archive.ics.uci.edu/ml/datasets.php. Accessed: 2020-08-02.

[22] LIBSVM data classification. https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html. Accessed: 2020-08-02.

[23] Why intel added cache partitioning. https://danluu.com/intel-cat/. Accessed: 2020-08-28.

[24] Docker what is a container? https://www.docker.com/resources/what-container. Accessed: 2020-08-11.

[25] Pqos/intel rdt utility. https://github.com/intel/intel-cmt-cat/tree/master/pqos. Accessed: 2020-08-11.

[26] Perf linux profiling with performance counters. https://perf.wiki.kernel.org/index.php/Main_Page. Accessed: 2020-08-11.

[27] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[28] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2015.

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8026–8037. Curran Associates, Inc., 2019.

[30] TensorBoard tensorflow's visualization toolkit. https://www.tensorflow.org/tensorboard. Accessed: 2020-08-12.

[31] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[32] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, HotNets '16, page 50–56, New York, NY, USA, 2016. Association for Computing Machinery.