ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Βελτιστοποίηση κατανομής LLC σε συστηματα Kubernetes

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΑΛΕΞΑΝΔΡΟΥ ΟΡΦΑΝΟΥΔΑΚΗ

Επιβλέπων: Γεώργιος Γκούμας
Επίκουρος καθηγητής ΕΜΠ

Αθήνα, Ιούνιος 2023

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και
Υπολογιστών

Βελτιστοποίηση κατανομής LLC σε συστηματα Kubernetes

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΑΛΕΞΑΝΔΡΟΥ
ΟΡΦΑΝΟΥΔΑΚΗ

Επιβλέπων: Γεώργιος Γκούμας
Επίκουρος καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 14η Ιουλίου 2023

.....	.....	.....
Γεώργιος Γκούμας	Νεκτάριος Κοζύρης	Διονύσιος Πνευματικάτος
Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ

Αθήνα, Ιούνιος 2023

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

Περίληψη

Η διασφάλιση υψηλής ποιότητας υπηρεσιών και η εκτέλεση εργασιών υψηλής προτεραιότητας στα σύγχρονα κέντρα δεδομένων είναι ζωτικής σημασίας. Στο παρελθόν, οι πάροχοι Υπηρεσιών Cloud το πετύχαιναν αυτό απομονώνοντας εφαρμογές σε πολυεπεξεργαστές, με αποτέλεσμα τη σημαντική υποαξιοποίηση των υπολογιστικών συστημάτων. Αυτή η απομόνωση είναι απαραίτητη επειδή η συνεκτέλεση των εφαρμογών οδηγεί σε διαμάχη για κοινόχρηστους πόρους, όπως η κρυφή μνήμη τελευταίου επιπέδου, η οποία βλάπτει την απόδοση και την ποιότητα της υπηρεσίας. Ωστόσο, το εργαλείο Workload Collocation Agent (WCA) της Intel που βασίζεται στις τεχνολογίες Cache-Monitoring και Cache-Allocation του Resource Director Technology (RDT) επίσης της Intel, επιτρέπει πλέον τη διαχείριση του επιπέδου ιεραρχίας της μνήμης σε kubernetes clusters, επιτρέποντας την εφαρμογή δυναμικών μηχανισμών με ανάδραση για τη βελτίωση της ποιότητας υπηρεσίας για εφαρμογές υψηλής προτεραιότητας και τη μείωση του χρόνου καθυστέρησης για εφαρμογές βέλτιστης προσπάθειας. Αυτή η διπλωματική εργασία μελετά και εκμεταλλεύεται αυτές τις τεχνολογίες για την προστασία εφαρμογών υψηλής προτεραιότητας ενώ παράλληλα διατηρεί υψηλά τε επίπεδα αξιοποίησης του υποκείμενου υπολογιστικού συστήματος. Ο μηχανισμός που θα αναπτυχθεί αξιολογείται χρησιμοποιώντας εφαρμογές πολλαπλών νημάτων μέσα σε ένα σύμπλεγμα Kubernetes που φιλοξενείται σε ένα μηχάνημα που χρησιμοποιεί τον επεξεργαστή Intel Xeon E5-2630 v4, αποδεικνύοντας την καταλληλότητά του για συνεκτέλεση εφαρμογών υψηλής προτεραιότητας με ευαισθησία στην κρυφή μνήμη και σε περιπτώσεις κορεσμού εύρους ζώνης μνήμης.

Λέξεις κλειδιά: Διαμοιρασμός κρυφής μνήμης, Καταμερισμός κρυφής μνήμης, Κρυφή μνήμη τελευταίου επιπέδου, Ποιότητα Υπηρεσίας, Εφαρμογή υψηλής προτεραιότητας, Διαχείριση κοινών πόρων, Intel RDT, Δυναμικός μηχανισμός προστασίας επίδοσης, Kubernetes, Συνεκτέλεση εφαρμογών

Abstract

Ensuring high-quality service and performance of high-priority tasks in modern data centers is crucial. In the past, Cloud Services providers achieved this by isolating applications in multi-processors, resulting in significant underutilization of computing systems. This isolation is necessary because co-execution of applications leads to contention for shared resources, such as the last level shared cache (LLC) memory, which harms performance and quality of service (QoS). However, Intel's Workload Collocation Agent (WCA) tool which leverages Intel's RDT Cache-Monitoring and Cache-Allocation technologies now enable management of the memory hierarchy level, allowing for the implementation of dynamic mechanisms with feedback to improve quality of service for high-priority applications and reduce time latency for best effort (BE) applications. This Diploma Thesis studies and exploits these technologies to protect high-priority applications and evaluates the mechanism using multi-threaded applications inside a Kubernetes cluster hosted in a machine using the Intel Xeon E5-2630 v4 processor, demonstrating its suitability for co-execution with cache-sensitive high-priority applications and in cases of memory bandwidth saturation.

Key words: Cache sharing, Cache allocation, Last Level Cache, Quality of service, High priority task, Intel RDT, Dynamic performance protecting mechanism, Kubernetes, Co-running applications

Ευχαριστίες

Πρώτα απ΄ όλα, θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή αυτής της διπλωματικής κ. Γεώργιο Γκούμα, για την ευκαιρία που μου έδωσε, να ασχοληθώ με το συγκεκριμένο θέμα και να επεκτείνω τις γνώσεις μου πάνω σε αυτό. Ιδιαίτερες ευχαριστίες θα ήθελα να αποδώσω στους μεταδιδακτορικούς ερευνητές Κωνσταντίνο Νίκα και Ιωάννη Παπαδάκη για την καθοδήγησή και τη διαρκή και άμεση στήριξη τους κατά τη διάρκεια της διπλωματικής.
Τέλος θα ήθελα να ευχαριστήσω την οικογένεια και τους φίλους μου για την αμέριστη συμπαράσταση και τη συνεχή στήριξή τους.

Περιεχόμενα

Ευρετήριο εικόνων

Ευρετήριο πινάκων

1 Εισαγωγή

1.1 Σύγχρονα πολυεπεξεργαστικά συστήματα

Την τελευταία δεκαετία η χρήση πολυπύρηνων επεξεργαστών (Chip Multiprocessors, CMP) έχει πλέον εδραιωθεί σε επεξεργαστικά συστήματα για υψηλή επίδοση, όπως για παράδειγμα στους εξυπηρετητές (servers) των σύγχρονων κέντρων δεδομένων (Data Centers). Σε αντίθεση με τα παλαιότερα μονολιθικά συστήματα, τα οποία αποτελούνταν αποκλειστικά και μόνο από μία επεξεργαστική μονάδα και υλοποιούσαν τεχνικές παραλληλοποίησης μόνο σε επίπεδο εντολών (Instruction Level Parallelism), τα σύγχρονα πολυπύρηνα επεξεργαστικά συστήματα παρέχουν τη δυνατότητα παράλληλης εκτέλεσης πολλαπλών εφαρμογών και ταυτόχρονα εξασφαλίζουν βελτιωμένη απόδοση στο σύστημα. Ωστόσο, σε τέτοια συστήματα προκύπτει το πρόβλημα ανταγωνισμού για τους κοινούς πόρους, ο οποίος ανταγωνισμός συνεπάγεται σημαντικές επιπτώσεις στην παροχή ποιότητας υπηρεσίας (Quality of Service, QoS).

Οι διαμοιραζόμενοι πόροι (shared resources), για τους οποίους ανταγωνίζονται οι συνεκτελούμενες εφαρμογές, περιλαμβάνουν την κοινόχρηστη κρυφή μνήμη (on-chip shared cache memory), το εύρος του διαύλου δεδομένων προς την μνήμη (memory bandwidth), ελεγκτές E/E (I/O controllers), κτλ.. Ο ανταγωνισμός, λοιπόν, για τους κοινούς αυτούς πόρους αποτελεί εμπόδιο στην επίτευξη ντετερμινιστικής και βέλτιστης εκτέλεσης των συνεκτελούμενων εφαρμογών, με αποτέλεσμα να παρατηρείται απρόβλεπτη εκτέλεση και κακή ποιότητα υπηρεσίας. Ως αποτέλεσμα, η παροχή ποιότητας υπηρεσίας σε ένα περιβάλλον συνεκτελούμενων εφαρμογών αποτέλεσε, και συνεχίζει ακόμα και σήμερα, να αποτελεί σημείο ενδιαφέροντος πολλών ερευνητών ([11], [8], [12], [4], [10]). Απώτερος στόχος των σχετικών ερευνών είναι ο αποτελεσματικός διαμοιρασμός των κοινόχρηστων πόρων και συγκεκριμένα του τελευταίου επιπέδου της κρυφής μνήμης (last level cache, LLC) του πολυεπεξεργαστή στις εφαρμογές με στόχο την παροχή ποιότητας υπηρεσίας και την ντετερμινιστική τους εκτέλεση.

1.2 Προκλήσεις στα πολυεπεξεργαστικά συστήματα

Η βελτίωση ποιότητας υπηρεσίας είναι πρωταρχικό ζήτημα σε όλα τα σύγχρονα πολυεπεξεργαστικά συστήματα. Είτε πρόκειται για ατομική και προσωπική χρήση υπολογιστικών συστημάτων, είτε για μεγάλες συστοιχίες υπολογιστών (clusters) ή κέντρα δεδομένων μεταξύ άλλων, οι χρήστες και οι διαχειριστές εκτελούν ταυτόχρονα πολλές εφαρμογές, απαιτώντας την βέλτιστη και, ανεξάρτητη από τις άλλες, εκτέλεσή τους. Μάλιστα, η ταχεία ανάπτυξη των Cloud εφαρμογών, σε συνδυασμό με την τάση για εικονικοποίηση (virtualization), αυξάνει τις υπολογιστικές απαιτήσεις, καθιστώντας την προστασία της ποιότητας υπηρεσίας ζωτικής σημασίας. Η εικονικοποίηση, με όλα τα πλεονεκτήματα που φέρει, εισήγαγε νέα προβλήματα στην διαχείριση της κατανομής κοινόχρηστων πόρων εφαρμογών σε πολυκομβικά υπολογιστικά συμπλέγματα. Τα εργαλεία ενορχήστρωσης των πακεταρισμένων εφαρμογών τους αποδίδουν πτητικό χαρακτήρα, καθώς οι εφαρμογές πλέον επανεκκινούνται συνεχώς και μετακινούνται μεταξύ των κόμβων για να επιτευχθεί καλύτερη αξιοποίηση των μηχανημάτων, εξισορρόπιση εισερχόμενου φορτίου, αυτόματη κλιμάκωση, έλεγχος καλής υγείας των εφαρμογών κ.α.. Όλα αυτά πραγματοποιούνται πάνω από ένα επιπλέον επίπεδο αφαιρετικότητας που καθιστά δύσκολη την δυναμική διαχείριση του μεγάλου πλήθους διεργασιών. Η αντιμετώπιση του ζητήματος αυτού είναι απαραίτητη για τους χρήστες και τους διαχειριστές των πολυεπεξεργαστικών και πολυκομβικών συστημάτων. Η διαχείριση των μοιραζόμενων πόρων μπορεί να πραγματοποιηθεί μέσω αλλαγών στο υλικό (hardware), παροχής υποστήριξης στο Λειτουργικό Σύστημα (Operating System, O.S.), ανάπτυξης μηχανισμών λογισμικού (software), ή συνδυασμού των προηγουμένων.

1.3 Εισαγωγή στο Intel RDT

Η κρυφή μνήμη (cache memory), και συγκεκριμένα το τελευταίο της επίπεδο (Last Level Cache, LLC) είναι ένας από τους πόρους για τους οποίους ανταγωνίζονται οι εκτελούμενες εφαρμογές σε ένα σύστημα, καθώς είναι διαμοιραζόμενη από τους πυρήνες ενός επεξεργαστή. Η παρακολούθηση της χρήσης καθώς και η διαχείριση της κατανομής της LLC απασχόλησε την Intel, η οποία ανέπτυξε τις αντίστοιχες τεχνολογίες Intel Cache Monitoring Technology (CMT) και Intel Cache Allocation Technology (CAT) ως τμήματα μιας ευρύτερης οικογένειας χαρακτηριστικών και δυνατοτήτων του υλικού με το όνομα RDT. [6]. Σε μία προσπάθεια εκμετάλλευσης των τεχνολογιών αυτών, ανέπτυξε επίσης διάφορα εργαλεία, μεταξύ των οποίων και το WCA, το οποίο στοχεύει στην απομόνωση συνεκτελούμενων διεργασιών και παρέχει υποστήριξη σε περιβάλλοντα kubernetes.

1.4 Υπάρχουσες λύσεις

Χάρη της βαρύτητας και της σημασίας του προβλήματος της επίδοσης συνεκτελούμενων εφαρμογών, έχουν γίνει διάφορες έρευνες και έχουν παρουσιαστεί λύσεις είτε για παρακολούθηση [11] είτε για διαχείριση και απομόνωση [4, 8, 5, 10] της πρόσβασης των εφαρμογών στην LLC, που μπορούν να εφαρμοστούν σε δημόσια νέφη. Δυστυχώς όμως, καμία από αυτές δεν δύναται να επεκταθεί συστηματικά και αποτελεσματικά σε περιβάλλοντα kubernetes, ειδικά με την δυνατότητα παροχής μίας εύχρηστης διεπαφής για ανάπτυξη μηχανισμών και αλγορίθμων ειδικά προσαρμοσμένους για τις ανάγκες του κάθε συστήματος.

1.5 Προσέγγιση της διπλωματικής

Η παρούσα διπλωματική εργασία μελετά τις τεχνολογίες του Intel RDT τις πιθανές χρήσεις τους για τη βελτίωση της επίδοσης των εφαρμογών που εκτελούνται σε ένα kubernetes cluster και αξιολογεί στην πράξη τις δύο τεχνολογίες με τη χρήση του εργαλείου WCA. Παρουσιάζει μία διερεύνηση σχετικά με τον τρόπο που συσχετίζεται η αποκλειστική ή όχι πρόσβαση σε κοινόχρηστους πόρους των εφαρμογών και αναλύει και υλοποιεί ένα μηχανισμό, ο οποίος συνδυάζοντας τις δύο προαναφερθείσες τεχνολογίες, εξασφαλίζει την επίδοση μιας εφαρμογής που χαρακτηρίζεται ως υψηλής προτεραιότητας όσο το δυνατόν εγγύτερα στη βέλτιστη κατάσταση. Η λειτουργία του μηχανισμού γίνεται με δυναμικό τρόπο, χωρίς την απαίτηση για οποιαδήποτε πληροφορία για τις εφαρμογές που εκτελούνται στο σύστημα. Με αυτόν τον τρόπο, καταφέρνει να βελτιώσει την επίδοση της κρίσιμης τάξης εφαρμογών έως και 90%, σε σχέση με την επίδοση που θα λαμβάναμε σε ένα περιβάλλον kubernetes με ανταγωνίζουσες εφαρμογές χωρίς εξωτερική επέμβαση.

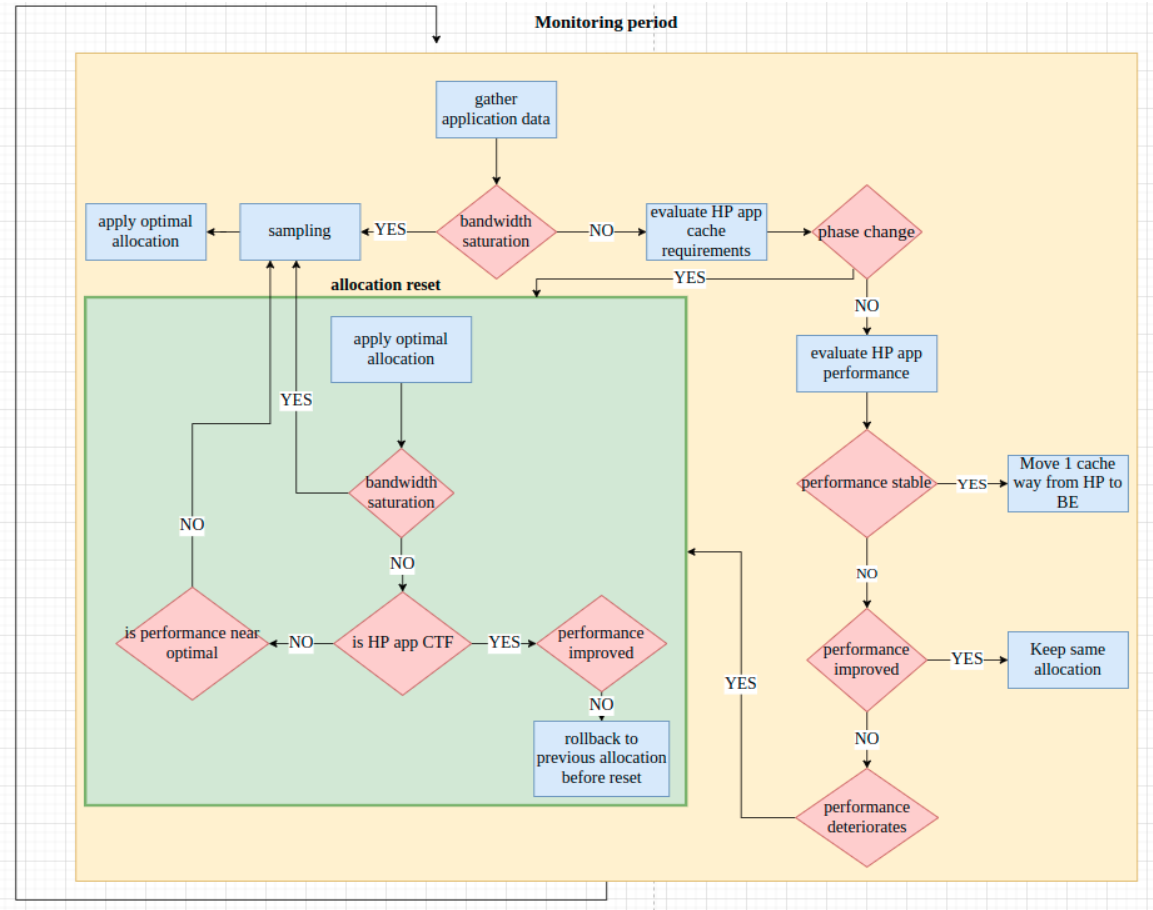
2 Βιβλιογραφικές αναφορές

2.1 Dicer

Ο Dicer είναι ένας μηχανισμός ο οποίος προσαρμόζει δυναμικά την κατανομή της κρυφής μνήμης τελευταίου επιπέδου, προσπαθώντας να ικανοποιήσει τις ανάγκες των HP εφαρμογών σε χωρητικότητα LLC σε όλη τη διάρκεια της εκτέλεσής τους. Επιχειρεί να διαφυλάξει την ποιότητα υπηρεσίας των HP εφαρμογών ενώ παράλληλα αποδίδει την μέγιστη δυνατή χωρητικότητα κρυφής μνήμης στις BE εφαρμογές για να βελτιώσει την επίδοσή τους και να αυξήσει την αξιοποίηση των πόρων του συστήματος. Στην παρούσα διπλωματική χρησιμοποιεί την τεχνολογία Intel RDT [6]

για την κατάτμηση της LLC, η οποία υλοποιείται αναθέτοντας την LLC ανά way of associativity σε κάθε τάξη υπηρεσίας.

Η λογική υψηλού επιπέδου που υλοποιεί φαίνεται στο σχήμα 1 και ακολουθεί την εξής ροή. Η εκτέλεση διαχωρίζεται σε ισομήκη διαστήματα διάρκειας T, κατά την διάρκεια των οποίων ο



Σχήμα 1: Λογική υψηλού επιπέδου του Dicer

Dicer παρακολουθεί την επίδοση της HP εφαρμογής βάσει κάποιας μετρικής, παραδοσιακά του IPC (εντολές ανά κύκλο ρολογιού) της, όπως επίσης το bandwidth που καταναλώνεται από τις εφαρμογές υπό παρακολούθηση. Στο τέλος κάθε περιόδου παρακολούθησης, ερμηνεύει την μεταβολή στην επίδοση των εφαρμογών βάσει των μετρικών που συνέλεξε και αποφασίζει πως να ανακατανέμει την LLC στις εφαρμογές που συνεκτελούνται. Πιο συγκεκριμένα, η εκτέλεση ξεκινάει με την υπόθεση πως κρίσιμη εφαρμογή ευνοείται από την Cache Takeover (CT) πολιτική. Η πολιτική αυτή εφαρμόζει την συντηρητική προσέγγιση της απόλυτης εύνοιας απέναντι στις κρίσιμες εφαρμογές, αποδίδοντας τους τη μέγιστη δυνατή χωρητικότητα κρυφής μνήμης και αφήνοντας κατά συνέπεια το ελάχιστο δυνατό για τις BE εφαρμογές. Επομένως ο Dicer εφαρμόζει τον αντίστοιχο “άδικο” διαμοιρασμό σύμφωνα με τον οποίον η HP εφαρμογή παίρνει τη μέγιστη δυνατή χωρητικότητα LLC, δήλαδή όλα τα cache ways εκτός από ένα, το οποίο ανατίθεται στην BE εφαρμογή. Στη συνέχεια, στο τέλος κάθε περιόδου ο Dicer έχει δύο επιλογές, να διαλευκάνει τον κορεσμό του bandwidth εάν υπάρχει, ή να επιχειρήσει να βελτιστοποιήσει την κατανομή της LLC.

Για τον εντοπισμό του πιθανού κορεσμού του bandwidth, ο Dicer επιβλέπει την κατανάλωση του εύρους ζώνης μνήμης των συνεκτελουμένων εφαρμογών. Εάν η κατανάλωση αυτή ξεπεράσει

ένα συγκεκριμένο κατώφλι σε κάποια περίοδο παρακολούθησης, τότε ο Dicer θεωρεί πως υπάρχει κορεσμός στον δίαυλο μνήμης και θα δώσει προτεραιότητα στην αντιμετώπιση αυτού του bottleneck. Την πρώτη φορά που θα εντοπιστεί κορεσμός του bandwidth, απορρέει το συμπέρασμα πως τα συνεκτελούμενα φορτία δεν ευνοούνται από την CT πολιτική και άρα θα πρέπει να διερευνηθούν και άλλες κατανομές που μετριάζουν την κατανάλωση του bandwidth. Αυτό επιτυγχάνεται μέσω δειγματοληψίας. Ο Dicer κατανέμει σταδιακά μειούμενη χωρητικότητα LLC στην HP εφαρμογή, αποδίδοντας την υπόλοιπη στα αντίτυπα της BE εφαρμογής. Στόχος αυτής της στρατηγικής είναι η εύρεση της κατανομής που θα βελτιστοποιήσει το IPC της HP εφαρμογής, μέσω της απονομής σε αυτή της μέγιστης δυνατής χωρητικότητας LLC για την οποία δεν υπάρχει συμφόρηση στο bandwidth. Κάθε κατανομή εφαρμόζεται για μία περίοδο παρακολούθησης, έτσι ώστε να γίνουν αισθητά τα αποτελέσματά της στην επίδοση των εφαρμογών και να μπορέσει ο μηχανισμός να πάρει συνεπείς αποφάσεις. Μόλις βρεθεί η βέλτιστη κατανομή, ο μηχανισμός την εφαρμόζει στις συνεκτελούμενες εφαρμογές και εκκινά μία νέα περίοδος παρακολούθησης.

Περνάμε τώρα στην δεύτερη δυνατότητα που υπάρχει σε κάθε περίοδο παρακολούθησης, την βελτιστοποίηση της κατανομής της κρυφής μνήμης. Αν και η δειγματοληψία που περιγράφηκε παραπάνω είναι μία αξιόπιστη μέθοδος εύρεσης της βέλτιστης κατανομής, καταλαβαίνουμε πως δεν είναι εφικτό και βιώσιμο να εφαρμόζεται συχνά, καθώς πέρα της μεγάλης της διάρκειας, ο μεγαλύτερος αριθμός κατανομών που διερευνά θα είναι υποβέλτιστος και σε αρκετές περιπτώσεις μπορεί να έχει επιβλαβείς επιπτώσεις στην επίδοση των εφαρμογών. Σε ένα πραγματικό υπολογιστικό σύστημα λοιπόν, στο οποίο η HP εφαρμογή μπορεί να κληθεί να διαχειριστεί χιλιάδες ή εκατομμύρια αιτήματα σε ένα παράθυρο μερικών λεπτών, θα ήταν καταστροφικό να πειραματιζόμαστε με τις κατανομές των διαμοιραζόμενων πόρων και να επηρεάζουμε αρνητικά την εμπειρία των χρηστών. Η στρατηγική που ακολουθεί λοιπόν ο Dicer για να επιχειρήσει να βελτιστοποιήσει τον διαμοιρασμό της LLC μεταξύ των εφαρμογών, όταν οι συνθήκες του συστήματος δεν επιβάλλουν δειγματοληψία, είναι μία προσεκτική και σταδιακή τροποποίηση της κατανομής κρυφής μνήμης τελευταίου επιπέδου της HP εφαρμογής, βάσει των μετρήσεων της προηγούμενης περιόδου παρακολούθησης, δηλαδή βάσει των αποτελεσμάτων που επέφερε η τελευταία κατανομή που ο μηχανισμός αποφάσισε να επιβάλλει στις εφαρμογές.

Η λογική υψηλού επιπέδου που ακολουθεί για να αποφασίσει τον διαμοιρασμό της κρυφής μνήμης σε κάθε περίοδο παρακολούθησης βασίζεται σε 2 κριτήρια, τις εκάστοτε ανάγκες και την επίδοση της HP εφαρμογής. Για πληθώρα σύγχρονων και μη εφαρμογών, είναι αναμενόμενο πως οι ανάγκες σε πόρους που επιδεικνύουν θα μεταβάλλονται κατά την διάρκεια της εκτέλεσής τους καθώς διέρχονται από διάφορα "στάδια" στην λειτουργία τους και υπάρχει ανομοιομορφία στο σύνολο λειτουργιών που επιτελούν. Ως παράδειγμα μπορούμε να θεωρήσουμε μία εφαρμογή η οποία κάθε 5 λεπτά διαβάζει κάποιες εγγραφές από μία βάση δεδομένων, επεξεργάζεται τα αποτελέσματα και στη συνέχεια γράφει ξανά πίσω στη βάση. Είναι προφανές πως κάθε στάδιο της εφαρμογής (ανάγνωση από την βάση, επεξεργασία δεδομένων, εγγραφή στην βάση, αναμονή) θα έχει διαφορετικές απαιτήσεις σε πόρους και άρα θα υπακούει σε διαφορετικούς κανόνες για την εύρεση της βέλτιστης κατανομής του. Στην περίπτωση λοιπόν που ο Dicer εντοπίσει μία σημαντική μεταβολή στις ανάγκες για εύρος ζώνης μνήμης της εφαρμογής, θα θεωρήσει πως έχει επέλθει μία αλλαγή φάσης στην λειτουργία της και θα επανακκινήσει την διαδικασία βελτιστοποίησης της κατανομής LLC. Μία μεταβολή κατηγοριοποιείται ως σημαντική εάν ικανοποιείται η παρακάτω σχέση σχετικά με την κατανάλωση εύρους ζώνης μνήμης της HP εφαρμογής:

$$MemBW_t^{HP} > (1 + phase\_threshold) * \sqrt[3]{\prod_{i=t-1}^{t-3} MemBW_i^{HP}}$$

Εάν οι ανάγκες τις HP εφαρμογής παραμένουν σταθερές μεταξύ δύο περιόδων παρακολούθησης,

τότε ο Dicer στρέφει την προσοχή του στις μεταβολές της επίδοσης της HP εφαρμογής και προσπαθεί να κρίνει εάν η επίδοση αυτή παρέμεινε σταθερή ή υπέστη κάποια μη αμελητέα μεταβολή. Η επίδοση θεωρείται σταθερή εάν το IPC της HP εφαρμογής παραμένει κατά ένα ποσοστό κοντά στο IPC της προηγούμενης περιόδου, σύμφωνα με την παρακάτω σχέση:

$$(1-a) * IPC_{t-1} \leq IPC_t \leq (1+a) * IPC_{t-1}$$

Εάν η μεταβολή της επίδοσης είναι αρκετά ασήμαντη ώστε να θεωρηθεί αμελητέα, ο Dicer υποθέτει πως η τρέχουσα κατανομή αποδίδει στην HP εφαρμογή περισσότερους πόρους από όσους χρειάζεται και θα μειώσει την χωρητικότητα της HP εφαρμογής κατά 1 way το οποίο θα αποδώσει στα BE αντίτυπα, επιχειρώντας να βελτιστοποιήσει όσο το δυνατόν και την δικιά τους επίδοση. Εναλλακτικά, εάν η επίδοση έχει βελτιωθεί σημαντικά σε σχέση με την προηγούμενη περίοδο παρακολούθησης, ο Dicer θα διατηρήσει την τρέχουσα κατανομή υπό την υπόθεση πως η εφαρμογή εισήλθε σε μία νέα φάση λειτουργίας με υψηλότερο IPC αλλά τις ίδιες ανάγκες σε κρυφή μνήμη. Τέλος, στην περίπτωση που η επίδοση μειώθηκε σε σύγκριση με την προηγούμενη περίοδο παρακολούθησης, ο Dicer συμπεραίνει πως αυτό οφείλεται είτε στην σταδιακή μείωση της χωρητικότητας LLC της HP εφαρμογής, είτε στο ότι η κρίσιμη εφαρμογή εισήλθε σε μία νέα φάση λειτουργίας με χαμηλότερο IPC αλλά τις ίδιες ανάγκες σε cache. Καθώς αδυνατεί να διαχωρίσει μεταξύ των δύο περιπτώσεων, ο Dicer θα επανακκινήσει την διαδικασία βελτιστοποίησης της κατανομής.

Η διαδικασία της επανεκκίνησης της κατανομής LLC είναι πολύ κρίσιμη για την λειτουργία του αλγορίθμου, καθώς καλείται να επαναφέρει την επίδοση της HP εφαρμογής εντός των ορίων ποιότητας υπηρεσίας και να επαναπροσδιορίσει τις ανάγκες σε πόρους της τρέχουσας φάσης που διανύει η εφαρμογή. Η ακριβής μεθοδολογία που θα ακολουθηθεί όμως εξαρτάται από τα χαρακτηριστικά της εφαρμογής και συγκεκριμένα από το αν είναι Cache Takeover Favoured (CTF) ή Cache Takeover Thwarted (CTT). Αρχικά, ο μηχανισμός θα εφαρμόσει την θεωρητικά βέλτιστη κατανομή σύμφωνα με τις προηγούμενες μετρήσεις. Αυτή, στην περίπτωση των CTF εφαρμογών θα είναι η "άδικη" κατανομή βάσει της οποίας η HP εφαρμογή λαμβάνει τη μέγιστη δυνατή χωρητικότητα σε LLC, ενώ για CTT εφαρμογές, θα εφαρμοστεί η βέλτιστη κατανομή σύμφωνα με την τελευταία δειγματοληψία που έλαβε χώρα. Μετά από αυτό ακολουθεί μία περίοδος αξιολόγησης κατά την οποία ο μηχανισμός θα κρίνει εάν οι υποθέσεις που έχει κάνει έχουν διαψευθεί ή όχι. Συγκεκριμένα, εάν παρατηρηθεί κορεσμός στο εύρος ζώνης μνήμης, τότε κρίνει πως απαιτείται δειγματοληψία για να βρεθεί η νέα βέλτιστη κατανομή, καθώς το φορτίο κατηγοριοποιείται πλέον ως CTT και ο μηχανισμός δεν έχει γνώση για την βέλτιστη κατανομή της τρέχουσας φάσης που διανύει η εφαρμογή. Αλλιώς, εάν η επίδοση της εφαρμογής βελτιωθεί (για CTF) ή είναι κοντά στην βέλτιστη (για CTT), τότε απορρέει το συμπέρασμα πως η απόφαση για επανεκκίνηση της εύρεσης βέλτιστης κατανομής ήταν ορθή και η διαδικασία συνεχίζει από αυτό το σημείο. Τέλος, αν η επίδοση χειροτερέψει, τότε για CTF εφαρμογές προκύπτει το συμπέρασμα πως η μείωση του IPC που πυροδότησε την διαδικασία επανεκκίνησης προκλήθηκε από μία αλλαγή φάσης με χαμηλότερο IPC και επομένως επιστρέφει στην κατανομή που είχε εφαρμόσει πρωτού ξεκινήσει η διαδικασία επανεκκίνησης της κατανομής LLC. Αντιθέτα, για CTT εφαρμογές, εάν η επίδοση δεν είναι παραπλήσια της βέλτιστης που έχει μετρηθεί έως τώρα, ο μηχανισμός υποθέτει πως τα χαρακτηριστικά της βέλτιστης παραμετροποίησης έχουν μεταβληθεί από την προηγούμενη δειγματοληψία και επομένως πρέπει να επαναληφθεί η διαδικασία. [7]

2.2 Cache slicer

Η διαχείριση της LLC σε εικονικές μηχανές (VM) σε δημόσιες πλατφόρμες cloud παρουσιάζει διάφορες προκλήσεις, οι οποίες μπορεί να οδηγήσουν σε φαινόμενα "θορυβώδους γείτονα" και

ασυνεπή επίδοση των εφαρμογών. Μία λύση σε αυτά τα προβλήματα αποτελεί το CacheSlicer, ένα σύστημα που παρέχει υποστήριξη σε επίπεδο cluster για τη διαχείριση της LLC των VM σε δημόσια νέφη. Το σύστημα περιλαμβάνει ένα API που βασίζεται σε πίνακες για τον καθορισμό πολιτικών διαχείρισης LLC, ένα μοντέλο βελτιστοποίησης για τον προσδιορισμό των καλύτερων πολιτικών και έναν προγραμματιστή VM με γνώση της LLC για την επιβολή των πολιτικών αυτών. Το CacheSlicer παρέχει υποστήριξη για ποικιλία υπηρεσιών στους χρήστες, ανάλογα με τις ανάγκες της κάθε εικονικής μηχανής, συμπεριλαμβανομένων υπηρεσιών για εικονικές μηχανές με υψηλή ευαισθησία στην κρυφή μνήμη στις οποίες παρέχει εγγυημένη απομόνωση, υποστήριξη για τυπικές εικονικές μηχανές με απομόνωση BE εφαρμογών και για εικονικές χαμηλού κόστους χωρίς εγγυήσεις απομόνωσης. Το σύστημα χρησιμοποιεί κατάτμηση υλικού σε επίπεδο cache-way και εισάγει νέες μετρήσεις για να επιτρέψει στους παρόχους να αξιολογούν τις παρεμβολές μεταξύ εφαρμογών και να διαχειρίζονται την χωρητικότητα της LLC χωρίς λεπτομερείς πληροφορίες σχετικά με τα φορτία εργασίας. [10]

2.3 Playing fetch with CAT

Οι in-memory βάσεις δεδομένων συχνά χρησιμοποιούν παράλληλες υλοποιήσεις τελεστών που μπορεί να καταναλώνουν δεδομένα με ρυθμό που υπερβαίνει το εύρος ζώνης του διαύλου μνήμης, γεγονός που μπορεί να οδηγήσει σε κορεσμό και σε επικείμενες χρονικές καθυστερήσεις. Διάφορες έξυπνες τεχνικές προσωρινής αποθήκευσης (caching) μπορούν να μετριάσουν αυτά τα σημεία συμφόρησης ως έναν βαθμό, αλλά δεν επωφελούνται εξίσου όλες οι διεργασίες σε μια μηχανή βάσης δεδομένων από την κρυφή μνήμη. Τελεστές όπως οι σαρώσεις στηλών δεν εμφανίζουν χρονική τοπικότητα πρόσβασης στη μνήμη και επομένως δεν επωφελούνται από την προσωρινή αποθήκευση. Διεργασίες χωρίς ή με χαμηλή χρονική τοπικότητα μπορούν να επωφεληθούν από προανάκτηση των δεδομένων (prefetching), όμως λαμβάνοντας υπόψιν την περιορισμένη χωρητικότητα της κρυφής μνήμης, μια διεύθυνση μνήμης θα έχει κατα πάσα πιθανότητα αντικατασταθεί μέχρι να επιχειρήσει κάποια διεργασία να την προσπελάσει ξανά.

Από την άλλη πλευρά, οι τελεστές με υψηλή χρονική τοπικότητα πρόσβασης στη μνήμη, όπως ένας τελεστής κατακερματισμού με μικρό πίνακα κατακερματισμού, επωφελούνται σημαντικά από την προσωρινή αποθήκευση. Σε σενάρια συνεκτέλεσης διεργασιών που διαμοιράζονται την κρυφή μνήμη τελευταίου επιπέδου, διεργασίες χαμηλής χρονικής τοπικότητας παρεμβαίνουν στην κρυφή μνήμη και δημιουργούν θόρυβο επιβλαβή για την επίδοση των διεργασιών υψηλής χρονικής τοπικότητας δεδομένων. Για να βελτιωθεί η αξιοποίηση της κρυφής μνήμης, τα δεδομένα ιδανικά θα πρέπει να φορτώνονται στην κρυφή μνήμη πρωτού επιχειρήσουν οι διεργασίες να τα προσπελάσουν και θα πρέπει να παραμείνουν διαθέσιμα μέχρι εκείνη την στιγμή. Για να επιτευχθεί αυτό, μπορούν να χρησιμοποιηθούν παράλληλα η προανάκτηση δεδομένων και η κατάτμηση LLC. Η τεχνολογία κατανομής προσωρινής μνήμης (CAT) της Intel κάνει τη διαμέριση LLC απλή και έχει αποδειχθεί αποτελεσματική στην ελαχιστοποίηση των διενέξεων της κρυφής μνήμης. Η δημοσίευση αυτή παρουσιάζει ένα δυναμικό πλαίσιο προανάκτησης δεδομένων και κατανομής της κρυφής μνήμης δεδομένων το οποίο προσαρμόζεται ανάλογα με τις συνεκτελούμενες διεργασίες και τον τρόπο που επικαλύπτονται τα δεδομένα που προσπελάζουν στην κρυφή μνήμη. Αυτό καθίσταται πλήρως δυνατό σε ένα περιβάλλον βάσης δεδομένων διότι τα μοτίβα πρόσβασης δεδομένων των ερωτημάτων που εκτελούνται είναι γνωστά από πριν και άρα μπορούν να παρθούν προληπτικές αποφάσεις με καλή γνώση του συστήματος. [12]

2.4 Cache inspector

Η επίδοση εφαρμογών που φιλοξενούνται σε δημόσια νέφη (public clouds) μπορεί να παρουσιάσει απρόβλεπτα χαρακτηριστικά και να παραβιάζει τους περιορισμούς της ποιότητας υπηρεσίας λόγω

παρεμβολών μεταξύ εφαρμογών, απότομες μεταβολές ή κορυφώσεις στον όγκο φορτίου που καλούνται να εξυπηρετήσουν, εσφαλμένες ρυθμίσεις σχετικά με την διαχείριση των πόρων ή και σφάλματα του συστήματος. Τα δημόσια νέφη παρέχουν κάποιες δυνατότητες παρακολούθησης της επίδοσης των εφαρμογών και των πόρων που τους αποδίδει το σύστημα, αλλά συχνά δεν επεκτείνονται σε κοινόχρηστους πόρους όπως η κρυφή μνήμη της CPU. Η επιστημονική κοινότητα έχει καταβάλει προσπάθειες με στόχο να μειώσει τις παρεμβολές σε κοινόχρηστες κρυφές μνήμες, αλλά η αξιοποίηση αυτών των τεχνικών στο cloud παραμένει πρόκληση. Για την γεφύρωση αυτού του χάσματος, μπορεί να αξιοποιηθεί το CacheInspector, ένα ελαφρύ εργαλείο παρακολούθησης της κρυφής μνήμης CPU που προσφέρει σε εφαρμογές cloud πληροφορίες σε πραγματικό χρόνο σχετικά με τους πόρους της κρυφής μνήμης που έχουν κατανεμηθεί σε ένα δημόσιο σύννεφο πολλαπλών ενοικιαστών. Το CacheInspector μπορεί να μετρήσει την κατανεμημένη χωρητικότητα της κρυφής μνήμης, την ταχύτητα και την καθυστέρηση κάθε επιπέδου κρυφής μνήμης και της κύριας μνήμης. Οι κατανομές προσωρινής μνήμης μεταβάλλονται σημαντικά με την πάροδο του χρόνου και το CacheInspector επιτρέπει στους χρήστες cloud να διαχειρίζονται καλύτερα τους διαθέσιμους πόρους τους και να βελτιστοποιούν την επίδοση των εφαρμογών τους. [11]

2.5 Parties

Το PARTIES είναι ένας διαχειριστής πόρων με επίγνωση της ποιότητας υπηρεσίας των συνεκτελουμένων εφαρμογών που έχει σχεδιαστεί για να επιτρέπει σε έναν αυθαίρετο αριθμό διαδραστικών, κρίσιμων ως προς την καθυστέρηση υπηρεσιών να μοιράζονται έναν φυσικό διακομιστή χωρίς παραβιάσεις της ποιότητας υπηρεσίας, χρησιμοποιώντας ένα σύνολο μηχανισμών κατάτμησης πόρων υλικού και λογισμικού. Προσαρμόζει δυναμικά τον διαμοιρασμό των πόρων κατά το χρόνο εκτέλεσης, με τρόπο που να ανταποκρίνεται στις απαιτήσεις QoS κάθε προγραμματισμένου φορτίου εργασίας και μεγιστοποιεί την αξιοποίηση του υποκείμενου μηχανήματος. Το σύστημα έχει αξιολογηθεί σε πλατφόρμες σύγχρονων διακομιστών με ένα σύνολο διαφορετικών διαδραστικών υπηρεσιών και δείχνει βελτίωση της επίδοσης στο πλαίσιο QoS κατά 61% κατά μέσο όρο σε σύγκριση με τους υπάρχοντες διαχειριστές πόρων. Οι εφαρμογές Cloud μετατοπίζονται σταδιακά από υπηρεσίες παρτίδας (batch) σε υπηρεσίες χαμηλής καθυστέρησης (low latency) και το PARTIES είναι σε θέση να διαχειριστεί αυτή τη νέα πραγματικότητα. Οι μηχανισμοί κατάτμησης που χρησιμοποιεί είναι διαθέσιμοι σε σύγχρονες πλατφόρμες, συμπεριλαμβανομένων κοντέινερ, thread pinning, κατάτμηση κρυφής μνήμης, κλιμάκωση συχνότητας, διαχωρισμός χωρητικότητας μνήμης και κατάτμηση εύρους ζώνης δίσκου και δικτύου για να ικανοποιήσει τις στιγμιαίες ανάγκες σε πόρους κάθε προγραμματισμένης διαδραστικής υπηρεσίας. [5]

2.6 LCA

Το LCA αποτελεί μια νέα προσέγγιση συν-προγραμματισμού για τους Πολυεπεξεργαστές Chip (CMPs) που βασίζεται σε ένα καινοτόμο μοντέλο ταξινόμησης εφαρμογών που παρακολουθεί τη χρήση πόρων σε όλη την ιεραρχία της μνήμης. Αυτό επιτρέπει την ακριβή πρόβλεψη των παρεμβολών εφαρμογών και υποστηρίζει έναν αλγόριθμο συνεκτέλεσης που υπερισχύει των υπάρχουσων πολιτικών προγραμματισμού όσον αφορά τόσο την επίδοση όσο και τη δικαιοσύνη διαμοιρασμού πόρων. Το LCA μπορεί εύκολα να εφαρμοστεί σε πραγματικά σενάρια συνεκτέλεσης εφαρμογών με ποικιλία χαρακτηριστικών ως προς τις ανάγκες στους πόρους και τις δυνατότητες του υποκείμενου διακομιστή, καθώς βασίζεται σε πληροφορίες που συλλέγονται κατά το χρόνο εκτέλεσης από υπάρχοντες μηχανισμούς παρακολούθησης σύγχρονων επεξεργαστών και δεν απαιτεί πρόσθετη υποστήριξη υλικού. Η υλοποίηση του εργαλείου εξετάζει τις προκλήσεις της διαμάχης πόρων στους CMP και τον τρόπο που ο συν-προγραμματισμός με επίγνωση της διαμάχης μπορεί να μετριάσει τις επιπτώσεις της συνεκτέλεσης

για την επίτευξη στόχων όπως η επίδοση του συστήματος, η δικαιοσύνη διαμοιρασμού πόρων και η αποφυγή λιμοκτονίας εφαρμογών, η τήρηση των ορίων QoS και η κατανάλωση ενέργειας. [4]

2.7 CoPart

Το CoPart (συντονισμένη κατάτμηση της κρυφής μνήμης τελευταίου επιπέδου και του εύρους ζώνης μνήμης) είναι μια προτεινόμενη τεχνική για τη βελτίωση της αποδοτικότητας των συνεκτελούμενων φορτίων εργασίας σε συστήματα υπολογιστικού νέφους και κέντρα δεδομένων. Το CoPart το επιτυγχάνει αυτό αναλύοντας δυναμικά τα χαρακτηριστικά των συνεκτελούμενων εφαρμογών και συντονίζοντας την κατανομή της κρυφής μνήμης τελευταίου επιπέδου και του εύρους ζώνης της μνήμης για μεγιστοποίηση της συνολικής δικαιοσύνης της κατανομής των πόρων. Το CoPart έχει σχεδιαστεί και υλοποιηθεί ως σύστημα χρόνου εκτέλεσης σε επίπεδο χρήστη σε Linux, χωρίς να απαιτεί τροποποιήσεις στο υποκείμενο λειτουργικό σύστημα. Οι πειραματικές αξιολογήσεις του CoPart χρησιμοποιώντας πλήρεις στοίβες λογισμικού και υλικού σε ένα σύστημα διακομιστή εμπορίου 16 πυρήνων δείχνουν ότι βελτιώνει σημαντικά τη δικαιοσύνη κατανομής πόρων των συνεκτελούμενων εφαρμογών (π.χ. 57,3% υψηλότερη δικαιοσύνη από μια πολιτική κατανομής πόρων που κατανέμει εξίσου πόρους σε συνεκτελούμενες εφαρμογές). [8]

3 Θεωρητικό υπόβαθρο

3.1 Εικονικοποίηση και Containers

Πριν τα τωρινά άλματα στον τομέα της εικονικοποίησης, ο παραδοσιακός τρόπος εξυπηρέτησης εφαρμογών ήταν σε φυσικούς διακομιστές (bare metal servers) [1]. Δεν υπήρχε τρόπος να καθοριστούν όρια πόρων για εφαρμογές σε έναν φυσικό διακομιστή και αυτό προκαλούσε προβλήματα κατανομής πόρων. Για παράδειγμα, εάν πολλές εφαρμογές εκτελούνται σε έναν φυσικό διακομιστή, μπορεί να υπήρχαν περιπτώσεις όπου μια εφαρμογή θα καταλάμβανε τους περισσότερους πόρους και, ως εκ τούτου, οι άλλες εφαρμογές θα είχαν χαμηλή επίδοση. Μια λύση για αυτό θα ήταν η εκτέλεση κάθε εφαρμογής σε διαφορετικό φυσικό διακομιστή. Αλλά αυτό δεν ευνοούσε την κλιμακωσιμότητα, καθώς οι πόροι θα υποαξιοποιούνταν και το κόστος αγοράς και συντήρησης πολλαπλών φυσικών διακομιστών είναι υψηλό. Ως λύση, εισήχθη η έννοια της εικονικοποίησης. Η εικονικοποίηση είναι η διαδικασία κατά την οποία ένας συγκεκριμένος πόρος όπως η RAM, η CPU, ή ο δίσκος μπορεί να «εικονικοποιηθεί» και να αναπαρασταθεί ως πολλαπλοί πόροι. Η εικονικοποίηση επιτρέπει την εκτέλεση πολλαπλών εικονικών μηχανών (Virtual Machines), με το δικό της λειτουργικό σύστημα και σύνολο πόρων η κάθε μία, σε έναν φυσικό διακομιστή. Καθιστά εφικτή λοιπόν την απομόνωση των εφαρμογών μεταξύ των VM και παρέχει ένα επίπεδο ασφάλειας καθώς οι πληροφορίες μιας εφαρμογής σε ένα VM δεν μπορούν να είναι ελεύθερα προσβάσιμες από μία εφαρμογή σε άλλο VM. Επιτρέπει επίσης την καλύτερη χρήση των πόρων σε έναν φυσικό διακομιστή, ευνοεί την κλιμακωσιμότητα και μειώνει το κόστος υλικού. Το επόμενο βήμα στην εικονικοποίηση ήταν οι περιέκτες (containers). Παρόμοια με ένα VM, ένα container έχει το δικό του σύστημα αρχείων, μερίδιο CPU, μνήμη, χώρο διεργασιών και πολλά άλλα. Η βασική διαφορά μεταξύ κοντέινερ και εικονικών μηχανών είναι ότι οι εικονικές μηχανές εικονικοποιούν μια ολόκληρη μηχανή μέχρι τα επίπεδα φυσικού υλικού ενώ τα container εικονικοποιούν μόνο επίπεδα λογισμικού πάνω από το επίπεδο του λειτουργικού συστήματος, επομένως, είναι πιο ελαφριά στους πόρους που απαιτεί η χρήση τους. Τα container έχουν γίνει δημοφιλή επειδή παρέχουν επιπλέον οφέλη, όπως φορητότητα σε πληθώρα διανομών λειτουργικών συστημάτων αλλά και στο cloud, εύκολη κλιμακωσιμότητα και αξιοπιστία της λειτουργίας τους.

3.2 Kubernetes

Για τη δημιουργία των μικροϋπηρεσιών σε containers ένα από τα πιο διάσημα εργαλεία που χρησιμοποιούνται είναι το Docker. Στην πραγματικότητα ο κώδικας, οι εξαρτήσεις και το περιβάλλον της κάθε μικροϋπηρεσίας πακετάρονται σε ένα δυαδικό αρχείο που ονομάζεται εικόνα του container (container image), η οποία είναι διαθέσιμη και αποθηκεύεται σε ένα αποθετήριο εφαρμογών (registry). Τα containers με τη σειρά τους δημιουργούνται τρέχοντας την εικόνα του container για την υπηρεσία που θα υλοποιήσουν. Η χειροκίνητη κλιμάκωση των containerized εφαρμογών είναι πολύ απαιτητική και δύσκολη. Αυτό οφείλεται στο μεγάλο πλήθος υποτμημάτων της εφαρμογής (μικροϋπηρεσίες) αλλά και σε παράγοντες που πρέπει να ληφθούν υπόψιν, όπως η εξισορρόπηση φορτίου (load balancing), η ορθή κατανομή πόρων και η ασφαλής λειτουργία των containers. Τα εργαλεία ενορχήστρωσης λύνουν το πρόβλημα αυτοματοποιώντας αυτές τις διαδικασίες και φροντίζοντας για τη σωστή εκτέλεση και τη διαθεσιμότητα των εφαρμογών. Το πιο διαδεδομένο και ευρέως χρησιμοποιούμενο εργαλείο ενορχήστρωσης είναι το Kubernetes (K8S), το οποίο είναι ένα σύστημα ανοιχτού κώδικα με στόχο να ομαδοποιεί τα κατανεμημένα τμήματα της εφαρμογής, να τα εκτελεί και να τα διαχειρίζεται αυτόματα, με δυνατότητα κλιμάκωσής τους ανάλογα με την ανάγκη τους σε πόρους. Βασίζεται σε master-slave αρχιτεκτονική, με τα κύρια τμήματα (master-components) να διαχειρίζονται την κατάσταση του cluster (συστάδας υπολογιστών - εικονικών ή πραγματικών συνδεδεμένων μέσω δικτύου), το οποίο αποτελείται από κόμβους-εργάτες (worker-nodes).

3.2.1 Αρχιτεκτονική

Pod Το pod είναι η μικρότερη υπολογιστική μονάδα που μπορεί να δημιουργηθεί και να διαχειριστεί από το Kubernetes, η οποία εκτελεί ένα ή περισσότερα containers που θα μοιράζονται τον ίδιο αποθηκευτικό χώρο και δίκτυο.

Deployment Το deployment περιγράφει τις ρυθμίσεις μιας συγκεκριμένης εφαρμογής. Καθορίζει το πλήθος των αντιγράφων των pods, το container image που θα χρησιμοποιηθεί αλλά και διάφορα άλλα πράγματα που αφορούν τα pod της εφαρμογής, όπως για παράδειγμα metadata (ετικέτες στη μορφή κλειδιού-τιμής). Φροντίζει μάλιστα για την επαναδημιουργία τους σε περίπτωση που κάποιο από αυτά τερματίσει για οποιοδήποτε λόγο. Έτσι επιτυγχάνεται μια αυτοματοποίηση στη διαχείριση των Pods και διασφάλιση της επιθυμητής κατάστασης του συστήματος.

Service Τα pods είναι πτητικά, δηλαδή το Kubernetes δεν εγγυάται ότι ένα δεδομένο φυσικό pod θα διατηρηθεί ζωντανό. Αντίθετα, μια υπηρεσία (service) αντιπροσωπεύει ένα λογικό σύνολο pods και λειτουργεί ως πύλη, επιτρέποντας στα υπόλοιπα pod του cluster να στέλνουν αιτήματα στην υπηρεσία χωρίς να χρειάζεται να παρακολουθούν ποια φυσικά pods αποτελούν πραγματικά την υπηρεσία.

Cluster Ένα σύνολο από μηχανήματα (φυσικά ή εικονικά) με εγκατεστημένο το k8s τα οποία επικοινωνούν και διαχειρίζονται ένα οικοσύστημα πακεταρισμένων εφαρμογών αποτελεί ένα k8s cluster (συστάδα). Κάθε ένα απο αυτά τα μηχανήματα μπορεί να λειτουργεί ως master node (κόμβος ελέγχου) ή ως worker node (κόμβος εργάτης), ανάλογα με τις λειτουργίες που επιτελούν οι διεργασίες που τρέχει.

Κόμβος ελέγχου Τα στοιχεία του κόμβου ελέγχου λαμβάνουν καθολικές αποφάσεις σχετικά με τη συστάδα, και αποκρίνονται σε συμβάντα που την αφορούν (για παράδειγμα, κλιμάκωση μιας εφαρμογής). Μία συστάδα μπορεί να έχει πάνω από έναν κόμβο ελέγχου για λόγους διαθεσιμότητας

και ανοχής σε σφάλματα του συστήματος. Οι βασικές οντότητες που συναποτελούν τον κόμβο ελέγχου είναι οι εξής:

etcd Χώρος αποθήκευσης κλειδιών-τιμών, υψηλής διαθεσιμότητας, που χρησιμοποιείται για την αποθήκευση πληροφοριών της συστάδας (όπως ο αριθμός των κόμβων, η κατάσταση των τρεχουσών εφαρμογών, ονοματόχωροι κλπ)

kube-apiserver Ο διακομιστής API Kubernetes είναι η κεντρική οντότητα διαχείρισης που λαμβάνει όλα τα αιτήματα REST για τροποποιήσεις της συστάδας και των λεπτομερειών που αφορούν τις τρέχουσες εφαρμογές. Επίσης, αυτό είναι το μόνο στοιχείο που επικοινωνεί με το etcd, διασφαλίζοντας ότι οι πληροφορίες που διατηρεί συμφωνούν με την τρέχουσα κατάσταση της συστάδας.

kube-scheduler Υπεύθυνο για την κατανομή των νέων pod σε κόμβους εργάτες που θα μπορούν να ικανοποιήσουν τις ανάγκες τους σε πόρους.

Kube-Controller-Manager ο οποίος εκτελεί ένα σύνολο controllers (ελεγκτών) για το τρέχον cluster. Ο controller-manager υλοποιεί τη διακυβέρνηση σε όλο το cluster.

Κόμβος εργάτης Ο worker-node είναι μια υπολογιστική μονάδα με πόρους προς κατανάλωση (CPU, RAM), η οποία συνιστά δομικό στοιχείο της συστάδας (cluster) στην οποία στήνεται μια containerized εφαρμογή. Οι κύριες διεργασίες που εκτελούνται σε έναν κόμβο εργάτη είναι οι εξής:

Kubelet Η κύρια υπηρεσία σε έναν κόμβο, που λαμβάνει τακτικά νέες ή τροποποιημένες προδιαγραφές pod (κυρίως μέσω του kube-apiserver) και διασφαλίζει ότι τα pod και τα container τους είναι υγιή και λειτουργούν στην επιθυμητή κατάσταση. Αυτό το στοιχείο δίνει αναφορά επίσης στο master node για την υγεία του διακομιστή όπου εκτελείται.

Kube-proxy Το kube-proxy διατηρεί κανόνες δικτύου στους κόμβους. Αυτοί οι κανόνες δικτύου επιτρέπουν σε συνεδρίες εντός ή εκτός του cluster να επικοινωνήσουν με τα Pods του cluster.

3.2.2 Χαρακτηριστικά

Τα σημαντικά χαρακτηριστικά του που το καθιστούν τόσο ευρέως διαδεδομένο περιλαμβάνουν:

Ασφάλεια Το k8s δίνει μεγάλη έμφαση στην ασφάλεια. Για το λόγο αυτό, περιλαμβάνει εργαλεία για την αποθήκευση και διαχείριση ευαίσθητων πληροφοριών όπως κωδικούς πρόσβασης, κλειδιά OAuth ή και SSH χωρίς να εκτίθενται μέσω του κώδικα της εφαρμογής.

Φορητότητα Το Kubernetes έχει σχεδιαστεί για να παρέχει συμβατότητα με πληθώρα επιλογών σε πάροχους cloud (AWS, Azure, Google cloud platform), container runtimes (docker, containerd) λειτουργικά συστήματα (οποιαδήποτε διανομή Linux) και αρχιτεκτονικές επεξεργαστών (virtual machines ή και bare metal).

Αυτόματη κλιμάκωση Το k8s παρέχει δυνατότητες για αυτόματη κλιμάκωση του cluster, των πακεταρισμένων εφαρμογών και των πόρων που τους αναθέτονται προς τα πάνω ή προς τα κάτω με βάση τις αναγκές που δημιουργεί το φορτίο που διαχειρίζεται το σύστημα.

Διαχείριση κύκλου ζωής εφαρμογών Μέσω του δηλωτικού μοντέλου του k8s, αρκεί να περιγράψουμε την επιθυμητή τελική κατάσταση του συστήματος και αυτό θα φροντίσει να την εφαρμόσει αυτόματα σε ελεγχόμενο ρυθμό διατηρώντας την καλή λειτουργικότητα και διαθεσιμότητα των εφαρμογών. Ως αποτέλεσμα, παρέχει δυνατότητες για αυτοματοποίηση των εξής διαδικασιών:

- Εκκίνηση των εφαρμογών και τοποθέτησης τους στον κατάλληλο κόμβο του συμπλέγματος ώστε να αξιοποιούνται οι πόροι του συστήματος
- Παύση και επανεκκίνηση μη ανταποκρινόμενων εφαρμογών
- Επαναφορά εφαρμογών σε προηγούμενες εκδόσεις ή ενημέρωσης τους σε νέες

Εξισορρόπηση φορτίου Το k8s φροντίζει ότι το εισερχόμενο φορτίο που θα κληθεί να διαχειριστεί μια εφαρμογή θα κατανεμηθεί ομοιόμορφα σε όλα τα αντίγραφά της, εξασφαλίζοντας καλή ποιότητα εξυπηρέτησης.

3.3 Κρυφή μνήμη

Η κρυφή μνήμη CPU (CPU cache) είναι μια κρυφή μνήμη υλικού που χρησιμοποιείται από την κεντρική μονάδα επεξεργασίας (CPU) ενός υπολογιστή για τη μείωση του μέσου κόστους (χρόνου ή ενέργειας) πρόσβασης σε δεδομένα από την κύρια μνήμη. Η κύρια λειτουργία που επιτελεί είναι να αποθηκεύει αντίγραφα των δεδομένων από τοποθεσίες της κύριας μνήμης που χρησιμοποιούνται συχνά, ώστε να επιταχύνει την εκτέλεση των εντολών του επεξεργαστή. Οι περισσότερες CPU έχουν μια ιεραρχία πολλαπλών επιπέδων κρυφής μνήμης (L1, L2, συχνά L3, και σπάνια ακόμη και L4), με διαφορετικές κρυφές μνήμες ειδικά για εντολές (instruction / I- cache) και ειδικά για δεδομένα (data / D- cache) στο επίπεδο 1. Τα πρώτα επίπεδα (L1, L2) είναι αποκλειστικά σε κάθε πυρήνα, ενώ τα τελευταία επίπεδα (L3, L4) διαμοιράζονται μεταξύ των πυρήνων που ανήκουν στον ίδιο κόμβο μη ομοιόμορφης πρόσβασης μνήμης (NUMA node).

Η μέτρηση της επίδοσης της κρυφής μνήμης έχει γίνει ιδιαίτερα σημαντική τα τελευταία χρόνια, όπου το χάσμα ταχύτητας μεταξύ της επίδοσης της μνήμης και της επίδοσης του επεξεργαστή αυξάνεται εκθετικά. Η κρυφή μνήμη εισήχθη για να μειώσει αυτό το χάσμα ταχύτητας. Έτσι, το να γνωρίζουμε πόσο καλά η κρυφή μνήμη μπορεί να γεφυρώσει το χάσμα στην ταχύτητα του επεξεργαστή και της μνήμης γίνεται σημαντικό, ειδικά σε συστήματα υψηλής επίδοσης. Ο πιο σημαντικός παράγοντας που επηρεάζει την επίδοση της κρυφής μνήμης είναι το ποσοστό αστοχίας (cache miss ratio).

Μία αστοχία της κρυφής μνήμης (cache miss) είναι μια αποτυχημένη προσπάθεια ανάγνωσης ή εγγραφής ενός τμήματος δεδομένων στη μνήμη cache, η οποία έχει ως αποτέλεσμα μια πρόσβαση στην κύρια μνήμη που επιφέρει πολύ μεγαλύτερη χρονική καθυστέρηση. Υπάρχουν διάφορα είδη αστοχιών, τα οποία εξαρτώνται ως ένα βαθμό και από το είδος της κρυφής μνήμης και τον τρόπο που αντιστοιχεί εκχωρήσεις στον φυσικό χώρο μνήμης της. Το πρώτο είδος είναι οι αναγκαστικές αστοχίες, ή αλλιώς αστοχίες πρώτης αναφοράς οι οποίες προκύπτουν όταν γίνεται πρόσβαση σε ένα κομμάτι της μνήμης για πρώτη φορά και είναι αναπόφευκτες την πρώτη φορά εκτέλεσης ενός προγράμματος. Το δεύτερο είδος είναι οι αστοχίες χωρητικότητας οι οποίες συμβαίνουν όταν το σύνολο δεδομένων που χρησιμοποιεί το πρόγραμμα υπό εκτέλεση είναι μεγαλύτερο από τη χωρητικότητα της κρυφής μνήμης και άρα αδυνατεί να χωρέσει εξ ολοκλήρου σε αυτή. Εφόσον η κρυφή μνήμη δεν μπορεί να συγκρατήσει όλα τα μπλοκ που απαιτούνται για την εκτέλεση του προγράμματος ταυτόχρονα, θα αναγκαστεί να απορρίψει και να αντικαταστήσει μερικά από αυτά κατά τη διάρκεια της εκτέλεσης. Το τρίτο είδος είναι οι αστοχίες σύγκρουσης ή παρέμβασης. Αυτές μπορεί να συμβούν όταν η πολιτική αντιστοίχισης που χρησιμοποιεί η κρυφή μνήμη για να αποθηκεύει τις

εκχωρήσεις, καθορίσει πως δυο μπλοκ αντιστοιχούν στην ίδια φυσική τοποθεσία της κρυφής μνήμης ακόμα και αν εξακολουθεί να υπάρχει ελεύθερος χώρος σε άλλες τοποθεσίες της.

Σε περίπτωση αστοχίας (χωρητικότητας ή σύγκρουσης), για να δημιουργηθεί χώρος για τη νέα καταχώρηση, η κρυφή μνήμη μπορεί να χρειαστεί να αντικαταστήσει μία από τις υπάρχουσες εγγραφές που διατηρεί. Η ευρετική συνάρτηση που θα χρησιμοποιηθεί για να επιλεγεί η καταχώρηση που θα αντικατασταθεί ονομάζεται πολιτική αντικατάστασης και η πιο δημοφιλής είναι η λιγότερο πρόσφατα χρησιμοποιημένη (LRU), η οποία αντικαθιστά την καταχώριση που χρησιμοποιήθηκε χρονικά πρώτη από όλες τις υπόλοιπες. Ως απόρροια αυτού, είναι απολύτως λογικό πως στα σύγχρονα πολυεπεξεργαστικά συστήματα όπου υποστηρίζεται η παράλληλη εκτέλεση πολλών εντολών, να προκαλείται μεγάλος ανταγωνισμός στην κρυφή μνήμη. Η συνεκτέλεση απαιτητικών φορτίων με αυστηρά όρια στην ποιότητα υπηρεσίας τους δημιουργούσε έναν μέχρι τώρα αναπόφευκτο κορεσμό στην κρυφή μνήμη και προκαλούσε την υπολειτουργία όλων των εφαρμογών, καθώς η συχνότητα αντικατάστασης εκχωρήσεων στην μνήμη αυξάνεται αισθητά, μαζί με το ποσοστό αστοχίας των εντολών.

Μία πολιτική αντιμετώπισης των αστοχιών συγκρούσεων σε φυσικό επίπεδο (hardware) είναι η χρήση set associative caches [2], οι οποίες μετριάζουν σε μεγάλο βαθμό το cache thrashing που μπορεί να συναντηθεί σε direct mapped κρυφές μνήμες. Cache thrashing είναι το φαινόμενο κατά το οποίο μία ή περισσότερες διεργασίες ανταγωνίζονται ενεργά κοινές διευθύνσεις στην κρυφή μνήμη με αποτέλεσμα να αντικαθιστώνται συνεχώς χρήσιμα δεδομένα και να εισάγονται σημαντικές καθυστερήσεις στην εκτέλεση των εφαρμογών, ενώ παράλληλα δεν αξιοποιείται στο μέγιστο η χωρητικότητα της κρυφής μνήμης. Σύμφωνα με την set associative αρχιτεκτονική, η κρυφή μνήμη χωρίζεται σε N ισομεγέθη τμήματα μνήμης, τα ways και οι τοποθεσίες στην μνήμη πλέον αντιστοιχούνται σε ένα way / set αντί γιά μία γραμμή, και το block μνήμης μπορεί να τοποθετηθεί σε οποιαδήποτε γραμμή του set είναι ελεύθερη. Στα σύγχρονα συστήματα οι κρυφές μνήμες δεδομένων πρώτου επιπέδου (L1D) τείνουν να έχουν 2 ή 4 ways, ενώ είναι σύνηθες για τις κρυφές μνήμες δεύτερου και τρίτου επιπέδου να έχουν τουλάχιστον 16 ways, νούμερο το οποίο τείνει να αυξάνεται παράλληλα με την σταδιακή αύξηση χωρητικότητας των νέων κρυφών μνημών [9].

3.4 RDT

Καταλαβαίνουμε λοιπόν ότι η διαμάχη σχετικά με τους κοινόχρηστους πόρους σε διαφορετικά επίπεδα της ιεραρχίας της μνήμης αποτελεί μία σημαντική αιτία χρονικής καθυστέρησης εφαρμογών στα σύγχρονα υπολογιστικά συστήματα. Οι κατασκευαστές υλικού, σε μία προσπάθεια να βελτιστοποιήσουν τις τεχνολογίες που παρέχουν και τις δυνατότητες τους να υποστηρίζουν πολύπλοκα και απαιτητικά σε πόρους συστήματα, καταβάλλουν όλο και μεγαλύτερες προσπάθειες και αρχίζουν να προτείνουν λύσεις για να μετριάσουν το πρόβλημα.

Η τεχνολογία Resource Director Technology (RDT) [6], [13] της Intel αντιπροσωπεύει μια τέτοια προσπάθεια. Δεδομένης της ευρείας υιοθέτησης των πλατφορμών Intel, οι δυνατότητες της τεχνολογίας RDT μπορούν να φέρουν νέα επίπεδα ελέγχου στον τρόπο με τον οποίο οι κοινόχρηστοι πόροι όπως η προσωρινή μνήμη τελευταίου επιπέδου (last level cache) και το εύρος ζώνης μνήμης (memory bandwidth) χρησιμοποιούνται και διαμοιράζονται σε εφαρμογές, εικονικές μηχανές (VM) και container σε μηχανήματα που χρησιμοποιούν επεξεργαστές Intel Xeon.

Συμπεριλαμβάνει τεχνολογίες που επιτρέπουν την παρακολούθηση και τον έλεγχο κοινόχρηστων πόρων, όπως το εύρος ζώνης της προσωρινής μνήμης τελευταίου επιπέδου (LLC) και της κύριας μνήμης (DRAM), που χρησιμοποιούνται από πολλές εφαρμογές, κοντέινερ ή VM που εκτελούνται ταυτόχρονα στην πλατφόρμα. Το RDT μπορεί να βοηθήσει στην ανίχνευση ”θορυβώδους γείτονα” και να συμβάλει στη μείωση των παρεμβολών στην επίδοση, εξασφαλίζοντας υψηλή επίδοση και

τήρηση των ορίων της ποιότητας της υπηρεσίας που παρέχουν οι HP διεργασίες.

Σε συστήματα Linux, το RDT εκτίθεται στον χρήστη μέσω του εικονικού συστήματος αρχείων resctrl (resource control filesystem). Η δομή αυτή επιτρέπει την οργάνωση των διεργασιών σε ιεραρχικές ομάδες των οποίων η χρήση διαφόρων τύπων πόρων μπορεί στη συνέχεια να περιοριστεί και να παρακολουθηθεί δυναμικά. Στην πράξη, η ιεραρχία αποτελείται από ομάδες πόρων (resource groups) ή αλλιώς τάξεις υπηρεσίας (classes of service) οι οποίες αναπαριστώνται ως υποφάκελοι του συστήματος αρχείων. Απο προεπιλογή, κατά την ενεργοποίηση του RDT, δημιουργείται η πρωταρχική ομάδα πόρων στην οποία ανήκουν όλες οι υπάρχοντες και μελλοντικές διεργασίες του συστήματος και η οποία έχει πρόσβαση στην καθολικότητα των πόρων του μηχανήματος. Ο χρήστης στη συνέχεια δύναται να δημιουργήσει ειδικά προσαρμοσμένες ομάδες και να τους αναθέσει την παρακολούθηση και την διαχείρηση διεργασιών είτε βάση αναγνωριστικού (pid) είτε βάση του πυρήνα της CPU στον οποίο τρέχουν. Έπειτα μπορεί να εφαρμόσει περιορισμούς σε κάθε ομάδα ξεχωριστά ως προς τους πόρους στους οποίους θα έχουν πρόσβαση δίνοντας έτσι την δυνατότητα να απομονώσουμε τις εφαρμογές του συστήματος και να ελέγξουμε τον ανταγωνισμό σε κοινόχρηστους πόρους.

Πιο συγκεκριμένα, οι τεχνολογίες που παρέχει και χρησιμοποιήθηκαν σε αυτή τη διπλωματική είναι οι εξής: Cache monitoring technology (CMT), Cache allocation technology (CAT), Memory bandwidth monitoring (MBM). Η τεχνολογία CMT επιτρέπει στο λειτουργικό σύστημα, στον επόπτη ή σε οποιονδήποτε διαχειριστή του συστήματος μνήμης να επιβλέπει την χρήση της LLC από τις εφαρμογές που τρέχουν στο σύστημα. Παρακολουθώντας τη χρήση της προσωρινής μνήμης τελευταίου επιπέδου (LLC) από μεμονωμένες διεργασίες, εφαρμογές ή VM, το CMT επιτρέπει προηγμένες αποφάσεις προγραμματισμού με επίγνωση των πόρων, βοηθά τον εντοπισμό ”θορυβώδους γείτονα” [3] και βελτιώνει τον εντοπισμό σφαλμάτων επίδοσης. Αντίστοιχα, η τεχνολογία MBM επιτρέπει την επίβλεψη του DRAM bandwidth ανά εφαρμογή ή πυρήνα ή και ομάδες αυτών στον επεξεργαστή. Αναφέρουμε ότι στην περίπτωση που υπάρχουν περισσότερες από μία υποδοχές επεξεργαστών (sockets) στην πλατφόρμα είναι δυνατόν να επιβλέπουμε και το bandwdith μεταξύ αυτών. Τα πλεονεκτήματα περιλαμβάνουν ανίχνευση θορυβωδών γειτόνων και χαρακτηρισμό και εντοπισμό σφαλμάτων επίδοσης για εφαρμογές ευαίσθητες στο εύρος ζώνης. Τέλος, η τεχνολογία CAT δίνει την δυνατότητα σε οποιονδήποτε διαχειριστή του συστήματος να καθορίσει το τμήμα της LLC που μπορεί να χρησιμοποιήσει κάθε πυρήνας, εφαρμογή ή/και ομάδες αυτών. Επιτρέπει λοιπόν σε δυναμική και κατάλληλα σχεδιασμένη ανακατανομή της κρυφής μνήμης, επιτρέποντας σε VM, container, ή εφαρμογές να επωφεληθούν από βελτιωμένη χωρητικότητα κρυφής μνήμης και αποκλειστικότητα σε αυτή.

Οι τεχνολογίες CMT-CAT υποστηρίζονται από το υλικό με την προσθήκη καταχωρητών και με τη χρήση επιπέδων αφαίρεσης (Abstraction Layers) που συσχετίζουν τους λογικούς πυρήνες (hardware threads) με τις εκτελούμενες εφαρμογές. Συγκεκριμένα, για το CMT, έχει προστεθεί ένα επίπεδο αφαίρεσης μεταξύ των εφαρμογών που τρέχουν στο σύστημα και των λογικών πυρήνων μέσω της χρήσης των RMIDs (Resource Monitoring ID). Κάθε εφαρμογή ή και συνδυασμός αυτών συσχετίζεται με ένα RMID. Επίσης, κάθε λογικός επεξεργαστής ή και ομάδα αυτών μπορεί να συσχετιστεί με ένα RMID. Για κάθε λογικό πυρήνα, μόνο ένα RMID είναι ενεργό κάθε φορά και ο συνολικός αριθμός των διαθέσιμων RMIDs είναι χαρακτηριστικό του επεξεργαστή.

Για την τεχνολογία CAT, υπάρχει ένα επίπεδο αφαίρεσης μεταξύ διεργασιών (software threads) και λογικών πυρήνων (hardware threads), το Class of Service (CLOS). Κάθε PID ή σύνολο αυτών μπορούν να αντιστοιχιστούν σε ένα CLOS. Επίσης, σε κάθε CLOS μπορούν να αντιστοιχιστούν ένας ή και περισσότεροι λογικοί πυρήνες. Αρχικά, δεν υπάρχει καμία ομαδοποίηση των πυρήνων και όλοι ανήκουν στο CLOS 0. Το RMID και το CLOS είναι ανεξάρτητα μεταξύ τους και έτσι η λειτουργία του CMT δεν επηρεάζει ή επηρεάζεται από το CAT. Η τεχνολογία CAT προσφέρει

way-based διαμοιρασμό της cache. Αυτό πραγματοποιείται με τη χρήση κατάλληλων bitmasks, οι οποίες ορίζουν το τμήμα της cache το οποίο "ανήκει" σε κάθε CLOS. Οι εφαρμογές που ανήκουν στο συγκεκριμένο CLOS μπορούν να γράψουν δεδομένα μόνο στο συγκεκριμένο τμήμα της cache όμως μπορούν να διαβάσουν ολόκληρη την cache.

3.5 WCA

Η τεχνολογία RDT δίνει αναμφισβήτητα νέες δυνατότητες στην διαχείριση των κοινόχρηστων πόρων σε ένα σύστημα. Η χειροκίνητη οργάνωση της όμως καταλήγει εικονικά αδύνατη σε ένα πολυκομβικό οικοσύστημα K8S, όπου τα pods που αντιπροσωπεύουν τις εφαρμογές είναι πτητικά με αποτέλεσμα να υπάρχει μία ασταμάτητη ροή νέων διεργασιών με την ανάγκη να υπόκεινται από την πρώτη στιγμή λειτουργίας τους σε συγκεκριμένους κανόνες ως προς τους πόρους που επιτρέπεται να αξιοποιήσουν. Κατά συνέπεια γεννήθηκε η ανάγκη για λογισμικό το οποίο θα μπορεί να διαχειρίζεται αυτή την οργάνωση σε μία συστάδα k8s και θα αποτελεί ένα επιπλέον επίπεδο αφαιρετικότητας πάνω στο RDT, επιτρέποντας στον χρήστη να διαχειρίζεται αποκλειστικά τον διαμοιρασμό και την παρακολούθηση των πόρων και όχι την δυναμική αντιστοίχιση διεργασιών σε ομάδες πόρων.

Ένα εργαλείο το οποίο παρέχει αυτές τις δυνατότητες είναι το workload collocation agent της Intel, στόχος του οποίου είναι να μειώσει τις παρεμβολές μεταξύ των συνεκτελούμενων εργασιών και να διαμοιράσει αποτελεσματικά τους πόρους του συστήματος, διασφαλίζοντας την ποιότητα της υπηρεσίας για HP διεργασίες.

Αξιοποιεί διάφορα υπάρχοντα χαρακτηριστικά και μετρικές που παρέχει ο πυρήνας (cgroups, /proc, /sys) όπως επίσης και το RDT ώστε να παρουσιάζει μία εύχρηστη και επεκτάσιμη διεπαφή που επιτρέπει στον χρήστη να έχει επίγνωση των πόρων του συστήματος και να τους διαχειρίζεται αποτελεσματικά.

Η αρχιτεκτονική του βασίζεται στην αφηρημένη Python κλάση Runner (δρομέας) η οποία ορίζει την μέθοδο run και αφήνει την υλοποίηση της ελεύθερη. Η μέθοδος αυτή νοηματικά καθορίζει τον τρόπο που θα αλληλεπιδράσει το εργαλείο με το σύστημα και εκτελείται σε έναν επαναλαμβανόμενο βρόχο ρυθμιζόμενης χρονικής διάρκειας. Η Intel παρέχει 3 διαφορετικές υλοποιήσεις της αφηρημένης κλάσης και της μεθόδου έτοιμες για χρήση, κάθε μία εκ των οποίων επιτελεί μία διαφορετική λειτουργία. Οι υλοποιήσεις αυτές μπορούν βέβαια να επεκταθούν και να ρυθμιστούν ειδικά στις ανάγκες του χρήστη και του υποκείμενου υπολογιστικού οικοσυστήματος εφαρμογών.

Η πρώτη υλοποίηση ορίζεται ως MeasurementRunner (δρομέας μετρήσεων) και αποτελεί βασικό στοιχείο του εργαλείου, υπεύθυνο για την διαχείριση και παρακολούθηση όλων των pod του cluster. Η υλοποίηση του συμπεριλαμβάνει πολλές σημαντικές εργασίες όπως:

- Αναγνώριση νέων pod και οργάνωσή τους κάτω από τις κατάλληλες ομάδες πόρων (RDT, cgroups). Αυτό επιτυγχάνεται με την εύρεση των αναγνωριστικών των διεργασιών που συνιστούν το κάθε pod και την εγγραφή τους στα κατάλληλα αρχεία των εικονικών συστημάτων αρχείων (resctrl, cgroups) των οποίων η ιεραρχία τα ομαδοποιεί κάτω από συγκεκριμένες ομάδες πόρων προς παρακολούθηση.

- Εκκάθαρση υπολειμμάτων σταματημένων η επανακκινημένων pod. Η υλοποίηση του συγκεκριμένου δρομέα έχει επίγνωση της κατάστασης υγείας του κάθε container που παρακολουθεί, με αποτέλεσμα να αναγνωρίζει πότε παύει η λειτουργία του καθενός, οπότε εφαρμόζει την καλή πρακτική να μην αφήνει αχρείαστες πληροφορίες και ίχνη στο σύστημα.

- Συγκέντρωση πληθώρας μετρικών τόσο για τον ίδιο τον διακομιστή όσο και για τα container που εξυπηρετεί. Ένας από τους κύριους λόγους που οργανώνει τις διεργασίες κάτω από τις

ιεραρχικές δομές των προαναφερθέντων ομάδων πόρων είναι για να έχει την δυνατότητα να συλλέγει τις μετρήσεις που παρέχει είτε ο ίδιος ο πυρήνας (για τα cgroups) είτε το RDT. Στις μετρήσεις αυτές συμπεριλαμβάνονται πολλές ενδιαφέροντες πληροφορίες όπως ποσοστό επιτυχίας / αστοχίας μιας διεργασίας ως προς την κρυφή μνήμη, χωρητικότητα της κρυφής μνήμης τελευταίου επιπέδου που καταλαμβάνει κάποια διεργασία ή ο χρόνος για τον οποίο κατέλαβε μία διεργασία την CPU.

- Κωδικοποίηση των μετρικών σε κατάλληλη μορφή (Prometheus format) συμβατή με τα κυρίαρχα εργαλεία αποθήκευσης και εικονικοποίησης μετρήσεων (Prometheus, Grafana) σε ένα k8s cluster, διευκολύνοντας σε μεγάλο βαθμό την συστηματική τους ανάλυση.

Η δεύτερη υλοποίηση ορίζεται ως DetectionRunner (δρομέας ανίχνευσης) και αναλαμβάνει την αλγοριθμική ανάλυση των μετρήσεων του MeasurementRunner προς εύρεση ανωμαλιών. Η υλοποίηση του αλγορίθμου ανίχνευσης είναι ελεύθερη στον χρήστη και δεν παρέχεται απο το εργαλείο, καθώς εξαρτάται από τις απαιτήσεις των εφαρμογών, τους πόρους του συστήματος και τους περιορισμούς που θα θέσει ο διαχειριστής του cluster. Στην περίπτωση που ο δρομέας ανίχνευσης εντοπίσει ανωμαλίες, έχει την ευθύνη να τις κωδικοποιήσει και αυτές ως μετρήσεις προκειμένου να μπορεί να τις αναφέρει σε εργαλεία εικονικοποίησης και ειδοποίησης των διαχειριστών του συστήματος.

Η τρίτη και τελευταία υλοποίηση ορίζεται ως AllocationRunner (δρομέας κατανομής πόρων) ο οποίος αντίστοιχα με τον δρομέα ανίχνευσης, αναλαμβάνει την αλγοριθμική ανακατανομή των πόρων του συστήματος στις διάφορες διεργασίες που παρακολουθεί το εργαλείο. Η Intel παρέχει κάποια παραδείγματα για δρομείς κατανομής πόρων, αλλά ο αλγόριθμος δυναμικής ανακατανομής πόρων είναι ξανά ανοιχτός προς σχεδιασμό και υλοποίηση του χρήστη. Το συγκεκριμένο στοιχείο είναι αυτό το οποίο έχει την δυνατότητα να επηρεάσει το σύστημα και να ενθυλακώσει την λογική μιας βελτιωμένης, δυναμικής ανακατανομής η οποία έχει επίγνωση της κατάστασης του συστήματος και επιχειρεί να απομονώσει συντρέχουσες εφαρμογές, να μειώσει τον θόρυβο στην κρυφή μνήμη τελευταίου επιπέδου ή και στην χρήση της CPU και να βελτιώσει την ποιότητα υπηρεσίας HP διεργασιών. Συγκεκριμένα, οι κατανομές που μπορεί να καθορίσει ο χρήστης εφαρμόζουν ένα σύνολο των εξής περιορισμών ανά διεργασία ή ανά ομάδα διεργασιών:

- cpu_quota: καθορίζει τον χρόνο για τον οποίο θα έχει πρόσβαση μία διεργασία στην CPU και εξαρτάται από το πλήθος λογικών πυρήνων του διακομιστή και την διάρκεια μίας περιόδου που ορίζεται από τον χρήστη

- cpu_shares: καθορίζει το ποσοστό της λογικής επεξεργαστικής ισχύος του συστήματος στην οποία θα έχει πρόσβαση η διεργασία

- cpuset_cpus: Καθορίζει τους πυρήνες του επεξεργαστή στους οποίους θα επιτρέπεται να εκτελεστεί μία διεργασία

- cpuset_mems: Καθορίζει τους κόμβους μη ομοιόμορφης πρόσβασης μνήμης (NUMA nodes) στους οποίους θα έχει πρόσβαση μία συγκεκριμένη διεργασία

- rdt: Παρέχει δυνατότητα για αξιοποίηση της τεχνολογιας CAT (Cache Allocation Technology) του RDT. Ο χρήστης μπορεί να καθορίσει τα κομμάτια της κρυφής μνήμης στα οποία θα επιτρέπεται να αντικαταστήσει υπάρχουσες εγγραφές μία διεργασία. Μπορεί κατά συνέπεια να απομονώσει τις διάφορες εφαρμογές του συστήματος και να τους δώσει θεωρητικά αποκλειστικά ένα ποσοστό χωρητικότητας της κρυφής μνήμης, περιορίζοντας έτσι τον ανταγωνισμό και βελτιώνοντας την επίδοση HP διεργασιών.

Είναι σημαντικό να αναφέρουμε πως η λειτουργία του εργαλείου είναι πολύ εύκολα παραμετροποιήσιμη, με την χρήση ενός αρχείου YAML μορφής το οποίο καθορίζει όλες τις μεταβλητές που έχουμε ορίσει στην υλοποίηση ότι θα εκτεθούν και θα τεθούν από τον χρήστη. Πέρα από παραμέτρους που θα απαιτούνται για υλοποιήσεις επεκτάσεων δρομέων του χρήστη, καποιές σημαντικές προυπάρχοντες παράμετροι είναι: η διάρκεια του βρόχου λειτουργίας του δρομέα, οι ονοματόχωροι του k8s cluster που θα παρακολουθούνται από το εργαλείο, μεταπληροφορίες που θέλουμε να αποθηκέυονται μαζί με τις μετρήσεις , οι υλοποιήσεις των δρομέων που θα χρησιμοποιηθούν ή, κάποια στατική κατανομή πόρων που θέλουμε να εφαρμόσουμε εξαρχής ή και επαναλαμβανόμενα για όλη τη διάρκεια λειτουργίας.

4 Κατάτμηση μνήμης σε K8s με χρήση WCA

4.1 Overview

Σε αυτή τη διπλωματική, θα πραγματοποιηθεί μία διερεύνηση όλων των δυνατών τρόπων κατάτμησης των κοινόχρηστων πόρων που επιτρέπει το WCA σε ένα K8S cluster. Θα εξεταστούν τα διάφορα σημεία κορεσμού της επίδοσης των εφαρμογών και θα εφαρμόσουμε στατικές και δυναμικές πολιτικές διαμοιρασμού των πόρων με γνώμωνα την βελτιστοποίηση της επίδοσης της ποιότητας υπηρεσίας τους αλλά και την αξιοποίηση των συνολικών πόρων του συστήματος.

4.2 Αρχιτεκτονική του Cluster

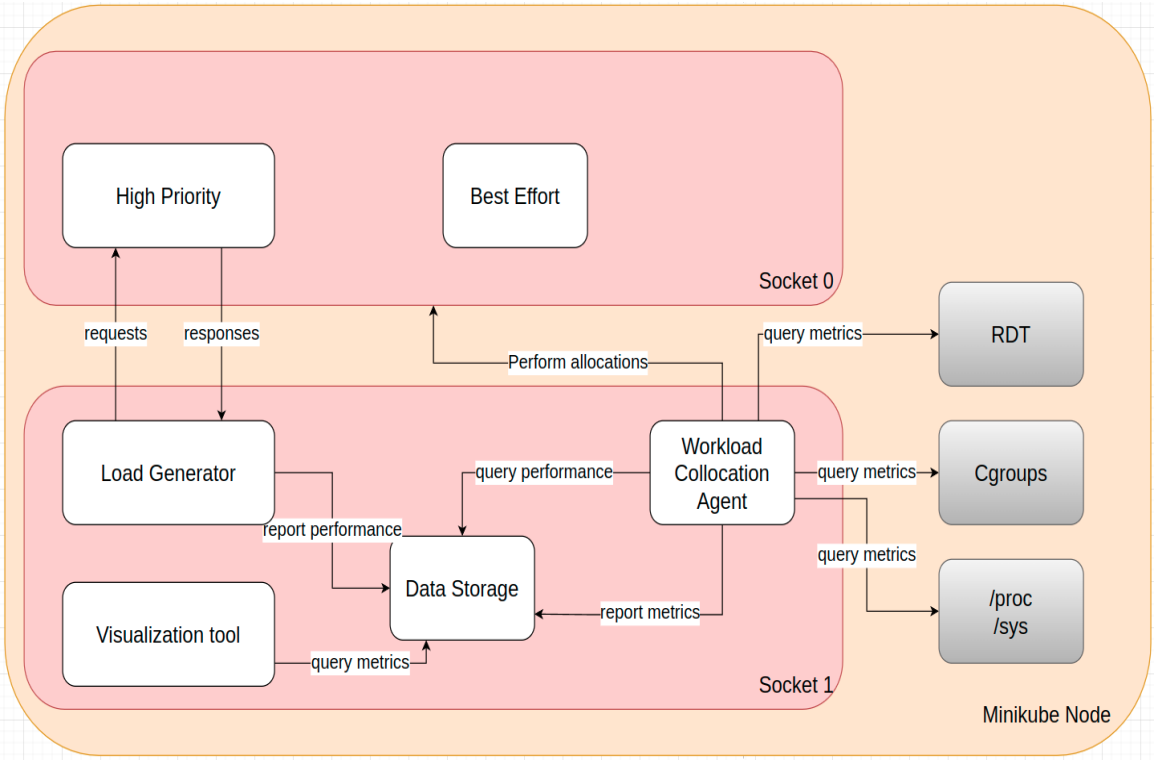
Στην παρούσα υποενότητα θα αναλυθούν όλες οι εφαρμογές του υπολογιστικού οικοσυστήματος που θα εξετάσουμε και οι κρίσιμες λειτουργίες που επιτελούν για την διεκπεραίωση των πειραμάτων. Θα σχολιαστεί η διαρρύθμισή τους σε σχέση με τον διακομιστή που τις εξυπηρετεί και ο τρόπος που επικοινωνούν μεταξύ τους αλλά και με το ίδιο το υπολογιστικό περιβάλλον ώστε να προσωμοιώνουν στο μέγιστο δυνατό πραγματικά σενάρια ανταγωνισμού κοινόχρηστων πόρων και να εκθέτουν απρόσκοπτα πληροφορίες για την επίδοση και την ποιότητα λειτουργίας τους.

4.2.1 HP Υπηρεσία

Ξεκινάμε με την τάξη υψηλής προτεραιότητας, η οποία αντιπροσωπεύει εφαρμογές των οποίων η επίδοση είναι κρίσιμης σημασίας σε ένα σύστημα και η υπέρβαση των ορίων στον χρόνο ανταπόκρισης τους μπορεί να αποβεί καταστροφική στην εμπειρία του χρήστη η την λειτουργικότητα του όλου συστήματος. Αυτή είναι η κύρια τάξη εφαρμογών της οποίας την συμπεριφορά στοχεύουμε να αναλύσουμε, να ερευνήσουμε και να συσχετίσουμε με τον ανταγωνισμό και τον διαμοιρασμό των κοινόχρηστων πόρων και ιδιαίτερα της κρυφής μνήμης τελευταίου επιπέδου σε αυτή τη διπλωματική. Σε πραγματικά συστήματα, τέτοιες εφαρμογές τείνουν να έχουν χαρακτήρα διακομιστή ο οποίος καλείται σε πραγματικό χρόνο να ανταποκριθεί σε πολλά ταυτόχρονα αιτήματα χρηστών (netflix streaming service, google search engine) και υπάρχουν αυστηρά όρια στην επιτρεπτή ποιότητα υπηρεσίας που παρέχουν.

4.2.2 Παραγωγή φορτίου και υπολογισμός απόκρισης

Για να μπορούμε να επιβλέπουμε την συμπεριφορά της HP εφαρμογής και τον τρόπο που αντιδράει στις μεταβολές των διαθέσιμων πόρων και τα διάφορα επίπεδα κορεσμού τους, είναι σημαντικό να έχουμε έλεγχο γύρω από το υπολογιστικό φορτίο που καλείται αυτή να διαχειριστεί κάθε χρονική



Σχήμα 2: Αρχιτεκτονική των εφαρμογών του Cluster

στιγμή, τόσο τον όγκο του όσο και την μορφή και την κατανομή του. Τον έλεγχο και την επίβλεψη αυτή την έχουμε εναποθέσει σε μία μικρουπηρεσία παραγωγής φορτίου (load generator microservice) η οποία προσομοιώνει την πληθώρα ταυτόχρονων αιτημάτων από τους χρήστες που καλείται να ικανοποιήσει η HP εφαρμογή. Επιπλέον ρόλος που εκτελεί η συγκεκριμένη υπηρεσία είναι η παρακολούθηση και εξαγωγή των μετρικών αποδοτικότητας της HP εφαρμογής σε μία αποθήκη δεδομένων από όπου θα μπορούν τα υπόλοιπα εργαλεία του συστήματος να έχουν πρόσβαση σε αυτή σε πραγματικό χρόνο.

4.2.3 BE Υπηρεσίες

Τον ρόλο του ανταγωνιστή τον επιτελούν οι BE εφαρμογές οι οποίες αντιπροσωπεύουν διεργασίες δευτερεύοντος σημασίας τις οποίες στοχεύουμε να βελτιστοποιήσουμε ως προς την επίδοση τους επηρεάζοντας στο λιγότερο δυνατό τις HP εφαρμογές. Στόχος λοιπόν των εφαρμογών αυτών είναι να καταναλώνουν τους κοινόχρηστους πόρους τους οποίους έχουν στην διάθεσή τους προκαλώντας έτσι κορεσμό στην LLC, το CPU time και το memory bandwidth και έχοντας επιβλαβή επίπτωση στην επίδοση των HP εφαρμογών. Όπως παρατηρούμε και στο διάγραμμα, οι εφαρμογές αυτές τοποθετήθηκαν στο ίδιο socket με τις HP εφαρμογές ώστε να ανταγωνίζονται κοινή LLC. Να αναφέρουμε σε αυτό το σημείο επίσης πως δεν έγινε χρήση του hyperthreading, καθώς διαφορετικά θα εμφανιζόταν ανταγωνισμός και στις ιδιωτικές cache των πυρήνων μεταξύ των εφαρμογών. Βασιζόμαστε μόνο στους 10 φυσικούς πυρήνες της κάθε επεξεργαστικής μονάδας.

Αυτές οι δύο τάξεις εφαρμογών (High priority, Best effort) αποτελούν τα αντικείμενα προς μελέτη, που προσομοιώνουν ένα ρεαλιστικό production υπολογιστικό οικοσύστημα πόρων και καταναλωτών. Τα στοιχεία της διπλωματικής που θα περιγράψουμε παρακάτω αποτελούν βοηθητικά εργαλεία για την εκτέλεση των πειραμάτων που μας επιτρέπουν να συνδυάσουμε τις τεχνολογίες που θα

χρησιμοποιηθούν και να αναπτύξουμε μηχανισμούς για βελτιστοποίηση της αποδοτικότητας του συστήματος. Όπως είναι λοιπόν λογικό, οι διεργασίες αυτές είναι απομονωμένες ως προς τους πόρους που καταναλώνουν, δρωντας σε ξεχωριστούς πυρήνες που αντιστοιχούν σε διαφορετικό socket από τις εφαρμογές υπό μελέτη, απαλείφοντας έτσι παρεμβάσεις στην CPU και την LLC που θα επέφεραν θόρυβο και ασυνέπεια στις μετρήσεις μας.

4.2.4 Αποθήκη δεδομένων

Για την αποθήκευση και συλλογή των επιθυμητών μετρικών των εφαρμογών μας (π.χ. CPU usage , tail latency, memory bandwidth) χρησιμοποιήσαμε το λογισμικό "Prometheus". Ο Prometheus είναι ένα εργαλείο παρακολούθησης και ειδοποιήσεων συστημάτων ανοιχτού κώδικα. Συλλέγει και αποθηκεύει μετρικές ως δεδομένα χρονοσειράς, δηλαδή οι πληροφορίες των μετρικών αποθηκεύονται μαζί με τη χρονική στιγμή κατά την οποία καταγράφηκαν, μαζί με προαιρετικά ζεύγη κλειδιών-τιμών που ονομάζονται ετικέτες (labels).

Ο Prometheus συλλέγει τις μετρικές που παράγει η εφαρμογή μας, αποθηκεύει τοπικά όλα τα δεδομένα που συλλέγονται και έπειτα εκτελεί κανόνες πάνω σε αυτά τα δεδομένα είτε για να συγκεντρώσει και να καταγράψει νέες χρονοσειρές από τα υπάρχοντα δεδομένα είτε για να δημιουργήσει ειδοποιήσεις. Το συγκεκριμένο επιτυγχάνεται μέσω του Prometheus-Operator, ο οποίος παρέχει την ανάπτυξη και τη διαχείριση του Prometheus και των σχετικών στοιχείων παρακολούθησης του Kubernetes.

4.2.5 Εργαλεία γραφικής απεικόνισης

Για την οπτικοποίηση των μετρικών που εξάγουμε από τον Prometheus χρησιμοποιούμε την "Grafana", ένα λογισμικό ανοικτού κώδικα που επιτρέπει την αναζήτηση και οπτικοποίηση των μετρικών και των αρχείων καταγραφής, οπουδήποτε και αν είναι αυτά αποθηκευμένα. Παρέχει εργαλεία για να μετατρέψουμε τα δεδομένα της βάσης δεδομένων χρονοσειρών σε διορατικά γραφήματα και οπτικοποιήσεις που διευκολύνουν στην ανάλυσή τους και την καλύτερη κατανόηση και εμβάθυνση σε αυτά.

4.2.6 WCA

Το κέντρο αποφάσεων ως προς τους κοινόχρηστους πόρους αποτελεί φυσικά το Workload Collocation Agent (WCA) το οποίο αναλύθηκε στο προηγούμενο κεφάλαιο. Όπως φαίνεται και στο διάγραμμα, όλη η διαθέσιμη πληροφορία για το σύστημα και τα επιμέρους στοιχεία του διέρχεται από το WCA. Αυτό του δίνει την δυνατότητα να λάβει εμπεριστατωμένες και ευφυείς αποφάσεις σχετικά με τον ανακαταμερισμό των πόρων και άμεση ανατροφοδότηση για τις αποφάσεις που παίρνει και τους διαμοιρασμούς που εφαρμόζει.

Ο τρόπος που το αξιοποιούμε, είναι με την δημιουργία δύο διαφορετικών Classes of Service (CloS), μία εκ των οποίων αντιστοιχεί στην HP εφαρμογή, ενώ η δεύτερη αντιστοιχεί στις BE εφαρμογές. Αυτό μας δίνει τη δυνατότητα να συγκεντρώνουμε ομαδοποιημένες μετρικές για τις δύο διαφορετικές ομάδες εφαρμογών, αλλά και το σημαντικότερο, να εφαρμόζουμε ξεχωριστούς κανόνες και περιορισμούς για την προσβασιμότητα των εφαρμογών στους κοινόχρηστους πόρους που διαθέτει το σύστημα. Από την μεριά του kubernetes, ο WCA λειτουργεί ως daemonset, με ένα τρέχον αντίγραφο της εφαρμογής ανά κόμβο εργάτη σε κάθε έναν εκ των οποίων μπορεί να εργάζεται ανεξάρτητα, με τις υπάρχουσες μετρήσεις των συνεκτελούμενων εφαρμογών και του διακομιστή που τις εξυπηρετεί. Ο λόγος που η αφαιρετικότητα που παρέχει ως προς τις διεργασίες που διαχειρίζεται είναι τόσο μείζονος σημασίας, είναι πως μπορεί και λειτουργεί ανεξαρτήτως των

φυσικών Pods και διεργασιών που μπορεί να έχουν πτητική φύση. Αντίθετα το WCA παρέχει τις δυνατότητες να ομαδοποιούμε διεργασίες και Pods σε ομάδες πόρων βάσει των kubernetes labels (ταμπέλες). Οποιοδήποτε Pod θέλουμε να ανήκει σε μία συγκεκριμένη ομάδα πόρων, αρκεί να του αποδώσουμε την αντίστοιχη ταμπέλα ως μεταπληροφορία και το WCA θα φροντίσει να ομαδοποιήσει τις διεργασίες του κατάλληλα κάτω από το σύστημα ιεραρχίας του εικονικού συστήματος αρχείων resctrl.

Στο παρακάτω κομμάτι κώδικα βλέπουμε ένα παράδειγμα παραμετροποίησης για το WCA το οποίο ορίζει πως τα Pods με τιμή "highpriority" για την ταμπέλα με κλειδί "clos" θα έχουν πρόσβαση μόνο στους πυρήνες 0-3 που αντιστοιχούν στο socket 0, όπως επίσης και μόνο στα τελευταία 4 bit, που αντιστοιχούν σε 1 way το καθένα από τα συνολικά 20 του socket 0 της LLC. Αντίστοιχα για τα Pods με τιμή "besteffort" για την ταμπέλα "clos".

```
  - name: highpriority
  allocations:
    cpuset_mems: "0"
    cpuset_cpus: "0-3"
    rdt:
      name: "highpriority"
      l3: "L3:0=0000f"
  labels:
    clos: highpriority

- name: besteffort
  allocations:
    cpuset_mems: "0"
    cpuset_cpus: "4-9"
    rdt:
      name: "besteffort"
      l3: "L3:0=000f0"
  labels:
    clos: besteffort
```

Το πρόβλημα που θα κληθούμε να λύσουμε μέσω του WCA είναι η εύρεση της βέλτιστης κατανομής της LLC, είτε δυναμικά είτε στατικά, ώστε να ικανοποιούνται στο μέγιστο δυνατό οι απαιτήσεις της ποιότητας υπηρεσίας της HP εφαρμογής, χωρίς παράλληλα να λιμοκτονούν τα αντίγραφα των BE εφαρμογών.

4.3 Dicer

Σημεία ενδιαφέροντος του μηχανισμού Σε αυτή τη παράγραφο θα σχολιάσουμε κάποια σημεία ενδιαφέροντος του μηχανισμού κατάτμησης της μνήμης που θα χρησιμοποιηθεί στην πειραματική ανάλυση του συστήματος, του Dicer, ο τρόπος υλοποίησης των οποίων στον κώδικα μπορεί να επηρεάσει σε μεγάλο βαθμό τα αποτελέσματα και τα ευρήματα που θα ανασκάψουμε. Αρχικά, ο Dicer παραδοσιακά χρησιμοποιεί το IPC ως κύρια μετρική για να κρίνει την επίδοση της HP εφαρμογής και να αποφασίσει πως θα βελτιστοποιήσει την κατανομή LLC. Στην υλοποίηση που παρουσιάζουμε, μέσω του οικοσυστήματος των επικοινωνούντων εφαρμογών μας, έχουμε πρόσβαση σε οποιαδήποτε μετρική για την επίδοση της HP εφαρμογής θέλουμε να χρησιμοποιήσουμε για

να χαρακτηρίσουμε την μεταβολή στην ποιότητα υπηρεσίας που παρέχει μεταξύ δύο περιόδων παρακολούθησης. Θα παρουσιάσουμε λοιπόν δύο διαφορετικές παραμετροποιήσεις, η πρώτη να βασίζεται στον μέσο χρόνο απόκρισης βάσει του 95ου εκατοστημόριου των εισερχομένων ομάδων αιτημάτων και η δεύτερη στο ποσοστό ομάδων αιτημάτων των οποίων το 95ο εκατοστημόριο του χρόνου απόκρισης υπερβαίνει το όριο των 10ms στην ποιότητα υπηρεσίας. Σε κάθε μια εκ των δύο περιπτώσεων, ο Dicer την στιγμή της απόφασης θα έχει διαθέσιμη την τιμή της εκάστοτε μετρικής για τις ομάδες αιτημάτων που αποστάλθηκαν την προηγούμενη περίοδο παρακολούθησης, δηλαδή τις τελευταίες 60 ομάδες ώστε οι αποφάσεις να παίρνονται αποκλειστικά βάσει της διαρρύθμισης της κρυφής μνήμης που είχε εφαρμοστεί στην προηγούμενη περίοδο παρακολούθησης.

Δεύτερο σημείο ενδιαφέροντος είναι το κριτήριο βάσει του οποίου η επίδοση της HP εφαρμογής θεωρείται σταθερή μεταξύ δύο περιόδων. Ο Dicer την απόφαση αυτή την παίρνει βάσει της σχέσης

$$(1-a) * IPC_{t-1} \leq IPC_t \leq (1+a) * IPC_{t-1}$$

την οποία θα μπορούσαμε να μεταφράσουμε ώστε εξαρτάται από την εκάστοτε μετρική που χρησιμοποιούμε για να κρίνουμε την ποιότητα υπηρεσίας. Το πρόβλημα έγκειται στο σενάριο που η μετρική μας χειροτερεύει σταθερά κάτα α% σε κάθε περίοδο. Ας πάρουμε το παράδειγμα που η μετρική μας είναι ο χρόνος απόκρισης και το α έχει την τιμή 10%. Αν θεωρήσουμε 3 διαδοχικές περιόδους οι οποίες λαμβάνουν τις μετρήσεις για τον χρόνο απόκρισης ως: 9ms, 9.9ms, 10.8ms, τότε ο Dicer στην επόμενη περίοδο θα εξακολουθεί να θεωρεί πως η επίδοση είναι σταθερή και θα περιορίζει ακόμα περισσότερο την πρόσβαση της HP εφαρμογής στην κρυφή μνήμη τελευταίου επιπέδου, διακινδυνεύοντας την επίδοσή της πολύ πέρα από το ορισμένο όριο.

Καταλαβαίνουμε λοιπόν πως το σύστημα που ερευνούμε, με τους αυστηρούς περιορισμούς που έχουμε θέσει για την ποιότητα υπηρεσίας, απαιτεί μία τροποποίηση του κανόνα αυτού. Η λύση που θα υλοποιήσουμε, για την αντιμετώπιση αυτού του σεναρίου είναι η εισαγωγή ενός επιπλέον αυστηρού άνω ορίου. Εάν η μετρική ποιότητας υπηρεσίας ξεπεράσει αυτό το όριο, η υλοποίησή μας θα θεωρήσει αμέσως πως πρέπει να λάβει δράση για να επαναφέρει την επίδοση της HP εφαρμογής, και επομένως θα εκκινήσει την διαδικασία επαναφοράς κατανομής LLC. Εξίσου σημαντικό όμως είναι να μην επαναφέρουμε άσκοπα την κατανομή LLC. Διότι σε πλήρη αντιστοιχία με το σενάριο της σταδιακής χειροτέρευσης της ποιότητας υπηρεσίας, είναι αναμενόμενες περιπτώσεις με απότομες μεταβολές η οποίες όμως είναι εντός των ορίων επίδοσης και δεν φέρουν άμεσο κίνδυνο στην ποιότητα υπηρεσίας. Αυτό διότι τα δείγματα που λαμβάνουμε φέρουν κάποια διακύμανση όπως υποδεικνύει και η διερεύνηση της υποενότητας 5.3.2. Αυτό σημαίνει πως είναι αναμενόμενο ακόμα και για κατανομές που ευνοούν πλήρως την κρίσιμη εφαρμογή και εμπίπτουν εντός του ορίου ποιότητας υπηρεσίας, να υπάρχουν κάποιες απότομες μεταβολές στα δείγματα που λαμβάνουμε. Για παράδειγμα εάν πάλι θεωρήσουμε το ίδιο παράδειγμα με το παραπάνω, αλλά αυτή τη φορά τα δείγματα που λαμβάνει ο Dicer στις διαδοχικές περιόδους είναι: 6ms, 5.5ms, 6.2ms. Σε αυτή την περίπτωση, καθώς η τελευταία μέτρηση δεν είναι εντός του επιτρεπτού εύρους μεταβολής εντός μίας περιόδου (10% στο συγκεκριμένο παράδειγμα), ο Dicer θα επανακκινήσει την διαδικασία επαναφοράς της LLC, ενώ η ποιότητα υπηρεσίας εξακολουθούσε να βρίσκεται ασφαλώς εντός του ορίου. Επομένως θα υλοποιήσουμε και ένα κάτω όριο, οποιαδήποτε μέτρηση κάτω του οποίου, θα θεωρείται ασφαλής και σταθερή, επιτρέποντας στον Dicer να μεταφέρει 1 way χωρητικότητας από την κρίσιμη εφαρμογή στις εφαρμογές χαμηλής προτεραιότητας. Η σχέση λοιπόν πλέον λαμβάνει την εξής μορφή σε ψευδοκώδικα:

```
if current_qos < lower_limit:
    performance = stable
elif current_qos > upper_limit:
```

```
    performance = deteriorated
else:
    if current_qos * (1 - a) >= previous_qos and
         current_qos * (1 + a) <= previous_qos:
        performance = stable
    elif current_qos > previous_qos:
        performance = deteriorated
    elif current_qos < previous_qos:
        performance = improved

```

Συμπεραίνουμε λοιπόν πως όσο αφορά την διαδικασία βελτιστοποίησης της κατανομής LLC, η τροποποιημένη έκδοση του Dicer που παρουσιάζουμε επιχειρεί να διατηρήσει την μετρική που αντιπροσωπεύει την ποιότητα υπηρεσίας ανάμεσα στα δύο όρια - παραμέτρους. Είναι στο χέρι μας λοιπόν να ορίσουμε ένα εύρος το οποίο βρίσκεται εντός του επιτρεπτού ορίου, αλλά ταυτόχρονα να είναι ρεαλιστικό βάσει της διερεύνησης που έχουμε πραγματοποιήσει, αλλιώς ο μηχανισμός θα οδηγηθεί σε πολιτική κατάληψης της κρυφής μνήμης και δεν θα αξιοποιεί τις δυνατότητες για δυναμική παρακολούθηση και λήψη αποφάσεων.

Επιπλέον, η εφαρμογή του Dicer μέσω του εργαλείου WCA επιφέρει κάποιους περιορισμούς στην λειτουργία του. Ο κυριότερος είναι πως οι στατικοί διαμοιρασμοί και η δυναμική λογική του Dicer που ενσωματώνουμε στο WCA δεν εκτελούνται συνεχώς στο υπόβαθρο, αλλά εκτελούνται μία φορά στην αρχή κάθε περιόδου παρακολούθησης. Αυτό έχει ως συνέπεια πως οι μεταβολές που επιφέρει ο μηχανισμός στο σύστημα, αλλά και οποιαδήποτε αξιολόγηση αυτών μέσω των μετρικών που έχουμε στην διάθεσή μας, μπορούν να εκτελούνται σε δεδομένες χρονικές στιγμές και όχι κατ'εντολή. Αξίζει επίσης να αναφέρουμε πως για χάρη του πειράματος θεωρούμε πως η HP εφαρμογή ευνοείται από την πολιτική κατάληψης της κρυφής μνήμης, χωρίς αυτό όμως να σημαίνει πως δεν θα διερευνηθούν όλες οι δυνατές κατανομές που πληρούν τις προυποθέσεις στην επίδοση των συνεκτελούμενων εφαρμογών.

Τέλος, ενδιαφέρον έχουν οι τιμές που θα αποδώσουμε στις διάφορες παραμέτρους του μηχανισμού, τις οποίες θα τις εξετάσουμε στις παρακάτω παραγράφους.

5 Αποτελέσματα

5.1 Περιγραφή συστήματος

Στον παρακάτω πίνακα παραθέτουμε τα χαρακτηριστικά του διακομιστή στον οποίον εκτελέστηκε η πειραματική μελέτη και ανάλυση τόσο των εφαρμογών όσο και των μηχανισμών που αναπτύχθηκαν. Είναι σημαντικό να αναφέρουμε πως ο επεξεργαστής Intel Xeon που χρησιμοποιήθηκε διαθέτει την τεχνολογία RDT η οποία δίνει δυνατότητες για διαχείριση και επίβλεψη των διαμοιραζόμενων πόρων που αναλύθηκε στο παραπάνω κεφάλαιο.

5.1.1 Χαρακτηριστικά Cluster

Για την δημιουργία του kubernetes cluster μέσα στο οποίο θα συνυπάρχουν οι εφαρμογές και τα εργαλεία μας, χρησιμοποιήθηκε το λογισμικό Minikube, το οποίο παρέχει δυνατότητες για δημιουργία

Όνομα μοντέλου	Intel® Xeon® E5-2630 v4
Βασική συχνότητα επεξεργαστή	2.20 GHz
CPUs	40
Αριθμός νημάτων ανά πυρήνα	2
Πυρήνες ανά socket	10
Αριθμός sockets	2
NUMA nodes	2
L1d cache	32K
L1i cache	32K
L2 cache	256K
L3 cache	25600K
Associativity	20
CMT - CAT - MBM	✓
MBA	✗

Table 1: Παράμετροι συστήματος

εικονικών μηχανών, με εγκατεστημένες όλες τις απαραίτητες εξαρτήσεις, στις οποίες θα λειτουργούν οι κόμβοι ελέγχου και οι κόμβοι εργάτες που θα φιλοξενούν τις εφαρμογές που μελετάμε. Στην παρούσα διπλωματική χρησιμοποιήθηκε μονοκομβικό cluster, με 20 cpus και 20GB μνήμης.

5.1.2 HP εφαρμογή

Για το ρόλο αυτό, επιλέχτηκε η in-memory, key-value store βάση δεδομένων MemCached από τη σουίτα Cloudsuite. Η συγκεκριμένη επιλογή δεν έγινε τυχαία καθώς πέρα από το ότι έχει χρησιμοποιηθεί ευρέως σε πλήθος εργασιών, κατέχει και σημαντική θέση στη βιομηχανία με μεγάλες εταιρίες όπως το Facebook, το Netflix να την αξιοποιούν συστηματικά. Ως μέτρο επίδοσης για την συγκεκριμένη υπηρεσία 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.

5.1.3 BE εφαρμογές

Η σουίτα Cloudsuite που προαναφέρθηκε πέρα από υπηρεσίες cloud διαθέτει και batch διεργασίες. Συγκεκριμένα διαθέτει δύο Apache Spark εφαρμογές, μία για in-memory και μία για graph analytics οι οποίες χρησιμοποιήθηκαν για τον ρόλο των BE εφαρμογών. Το in-memory benchmark χρησιμοποιεί το Apache spark και εκτελεί έναν συνεργατικό αλγόριθμο φιλτραρίσματος σε σύνολο δεδομένων με αξιολογήσεις ταινιών από χρήστες. Η graph-analytics διεργασία χρησιμοποιεί Apache GraphX και Apache spark ώστε να εκτελέσει αλγορίθμους ανάλυσης γράφων σε ένα σύνολο δεδομένων του Twitter.

5.1.4 Loader

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. 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.

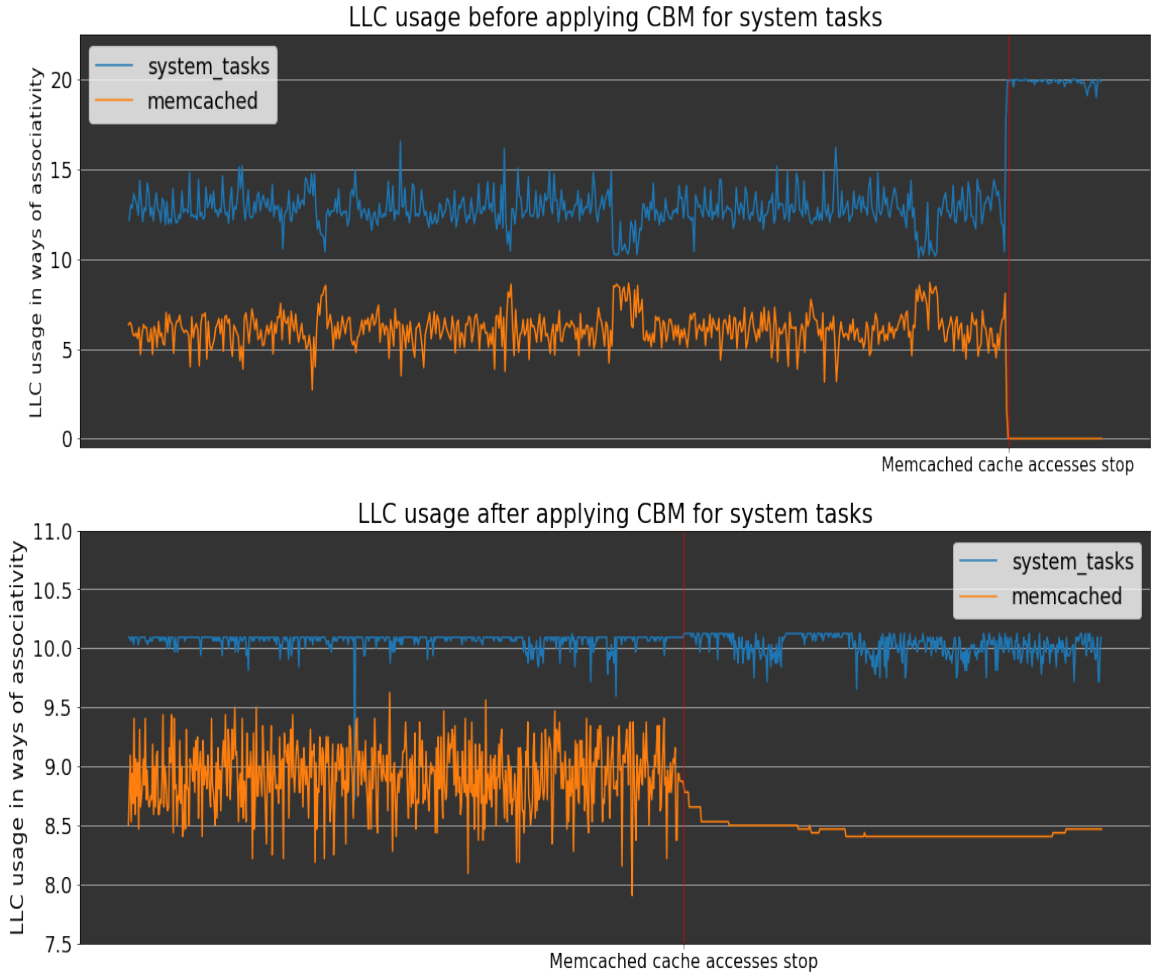
5.2 Θόρυβος από διεργασίες συστήματος

Αρχικά, για να μπορούμε να είμαστε βέβαιοι για την συνέπεια των πειραμάτων μας και ακριβείς σχετικά με το αντίκτυπο της συνεκτέλεσης και του ανταγωνισμού των πόρων στην επίδοση των εφαρμογών, πρέπει να είμαστε βέβαιοι πως δεν υπάρχουν διεργασίες του συστήματος που καταναλώνουν κοινόχρηστους πόρους χωρίς επίβλεψη και εισάγουν επιπλέον ανταγωνισμό στο σύστημά μας. Ειδικά με την αρχιτεκτονική που χρησιμοποιήσαμε, σύμφωνα με την οποία κόμβος ελέγχου και κόμβος εργάτης διενεργούν στο ίδιο μηχάνημα, αναμένουμε κάποιον θόρυβο από διαχειριστικές διεργασίες του minikube και του k8s control node που θα πρέπει να διαχειριστούμε.

Αυτό μπορεί να συμβεί διότι ενώ το WCA μας δίνει την δυνατότητα να περιορίσουμε τις προσβάσεις των εφαρμογών σε συγκεκριμένα τμήματα της LLC, δεν απαγορεύει σε άλλες διεργασίες, που δεν του έχουμε δηλώσει αποκλειστικά να διαχειρίζεται, να έχουν πρόσβαση σε οποιδήποτε τμήμα της κρυφής μνήμης. Σε ένα πιο ρεαλιστικό σύνολο μηχανημάτων στο οποίο οι κόμβοι ελέγχου θα εξυπηρετούνταν σε διαφορετικούς διακομιστές από τους κόμβους εργάτες, όλοι εκ των οποίων θα είχαν πρόσβαση στην τεχνολογία RDT, το πρόβλημα αυτό δεν θα υπήρχε και ο ανταγωνισμός θα περιοριζόταν μόνο μεταξύ εφαρμογών του cluster. Για να το πετύχουμε αυτό στο δικό μας σύστημα, θα πρέπει να περιορίσουμε την πρόσβαση που θα έχουν αυτές οι διεργασίες στην LLC.

Δυστυχώς το WCA αδυνατεί να επιδράσει πάνω σε διεργασίες πέρα από τις εφαρμογές των κόμβων εργατών, επομένως λόγω της αρχιτεκτονικής των πειραμάτων, θα πρέπει να επέμβουμε χειροκίνητα και να εξαλείψουμε τον θόρυβο αυτό. Αυτό θα το επιχειρήσουμε με μία μάσκα στα bit της cache στα οποία θα επιτρέπεται η πρόσβαση των διεργασιών που δεν ανήκουν στο οικοσύστημα των εφαρμογών που θα εξυπηρετεί το cluster μας, μέσω του resctrl interface που εκθέτει η τεχνολογία RDT. Όπως αναφέραμε σε προηγούμενο κεφάλαιο, όλες οι διεργασίες του συστήματος αρχικά συμπεριλαμβάνονται στην πρωταρχική ομάδα πόρων του RDT, η οποία έχει πρόσβαση στην καθολικότητα

των πόρων του μηχανήματος. Θα εφαρμόσουμε λοιπόν σε αυτή την ομάδα πόρων μία μάσκα η οποία θα περιορίσει την πρόσβαση όλων των διεργασιών στα 10 πάνω bit της LLC. Αυτό έχουμε



Σχήμα 3: Κατανάλωση LLC της HP εφαρμογής πριν και μετά την μάσκα στις διεργασίες του συστήματος

την δυνατότητα να το κάνουμε διότι οποιαδήποτε κατανομή και μάσκα εφαρμόσουμε σε επιμέρους ομάδες πόρων που θα αντιστοιχούν σε συγκεκριμένες εφαρμογές και διεργασίες θα έχει προτεραιότητα έναντι των περιορισμών της πρωταρχικής ομάδας πόρων. Η συμπεριφορά που παρατηρούμε πριν και μετά την εφαρμογή της μάσκας αυτής φαίνεται στο σχήμα 3. Στην πρώτη εικόνα φαίνεται η κατανάλωση της LLC από την HP εφαρμογή και από τις διεργασίες του συστήματος όταν έχουμε περιορίσει μόνο την HP εφαρμογή μέσω του WCA στα 10 κάτω bit της LLC. Παρατηρούμε πως από τα 10 ways που έχουμε αποδώσει θεωρητικά, πρακτικά η εφαρμογή μας καταναλώνει περίπου το 60%, με μία διακύμανση της τάξης του 25% που υποδηλώνει πως εξακολουθεί να υπάρχει ανταγωνισμός για την χωρητικότητα της κρυφής μνήμης μεταξύ της HP εφαρμογής μας και των διεργασιών του συστήματος.

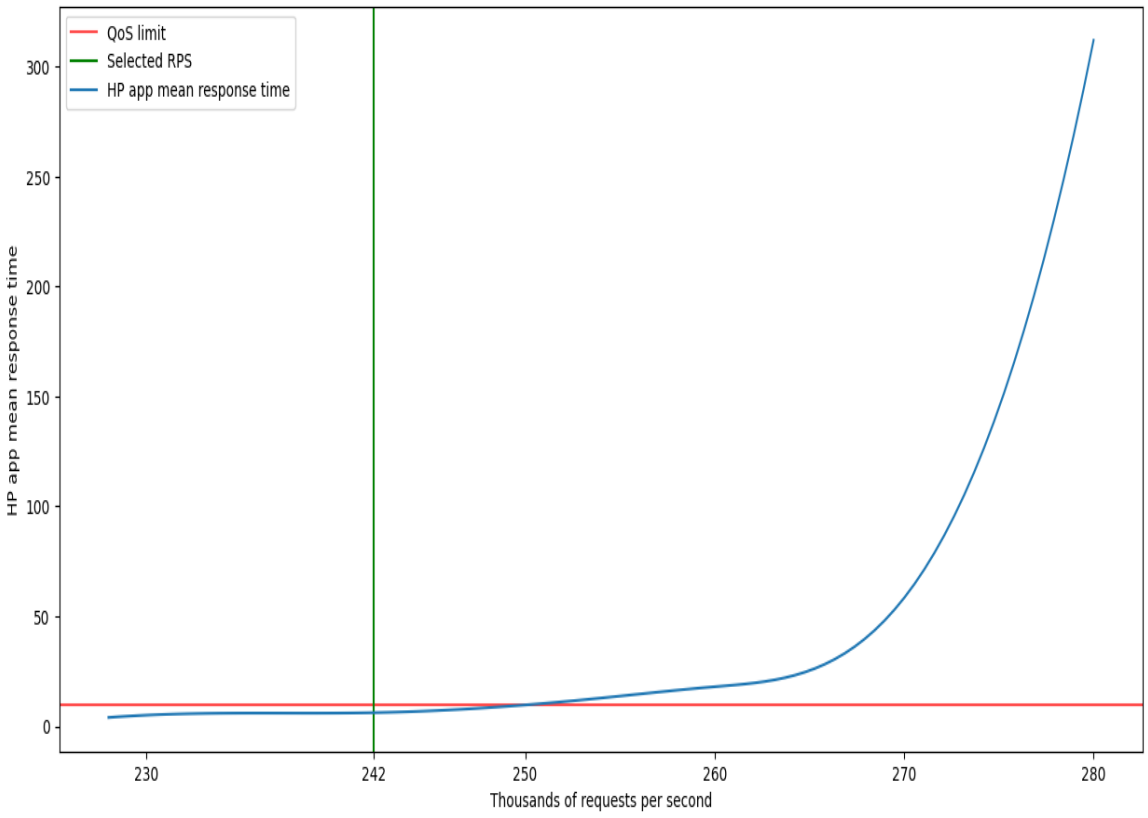
Επιπλέον βλέπουμε πως μόλις η κρίσιμη εφαρμογή σταματήσει τις προσβάσεις στην κρυφή μνήμη, οι διεργασίες του συστήματος την καταλαμβάνουν ολόκληρη, κάτι που ήταν αναμενόμενο καθώς δεν τους έχουμε εφαρμόσει κανέναν περιορισμό και επομένως ανταγωνίζονται όλη την χωρητικότητα της LLC. Στην δεύτερη εικόνα φαίνονται τα αποτελέσματα του ίδιου πειράματος όταν περιορίζουμε

χειροκίνητα και τις διεργασίες του συστήματος στα 10 πάνω bit της LLC. Η κατανάλωση πλέον αυξάνεται σημαντικά, κοντά στο 90% της χωρητικότητας που αποδώσαμε θεωρητικά στην κρίσιμη εφαρμογή, ενώ η διακύμανση μειώνεται περίπου στο 10%. Επίσης η κρίσιμη εφαρμογή διατηρεί πάνω απο το 90% της χωρητικότητας που είχε ακόμα και αφού σταματήσουν οι προσβάσεις της στην κρυφή μνήμη, κάτι που υποδηλώνει πως δεν υπάρχει άλλη εφαρμογή που να αντικαθιστά τις υπάρχουσες εκχωρήσεις στα τμήματα της LLC που έχουμε αποδώσει στην HP εφαρμογή.

5.3 Συμπεριφορά HP ως προς LLC

5.3.1 Προσδιορισμός μέγιστου επιτρεπτού φορτίου

Το επόμενο βήμα στην πειραματική προετοιμασία μας είναι να προσδιορίσουμε το φορτίο που μπορεί να εξυπηρετήσει καλώς η κρίσιμη εφαρμογή μας όταν λειτουργεί με αποκλειστικότητα στους πόρους του συστήματος. Αυτό είναι αναγκαίο διότι θέλουμε οποιαδήποτε μεταβολή στους διαθέσιμους κοινόχρηστους πόρους να επιφέρει εμφανή αποτελέσματα στην επίδοση των εφαρμογών, επομένως πρέπει να επιλέξουμε ένα φορτίο που βρίσκεται πολύ κοντά στο επιτρεπτό κατώφλι της ποιότητας υπηρεσίας. Για να το πετύχουμε αυτό, θα πραγματοποιήσουμε μία δειγματοληψία κατά την οποία θα υπολογιστεί ο μέσος χρόνος απόκρισης της HP εφαρμογής για διάφορους όγκους φορτίου ώστε να καταλήξουμε στον όγκο φορτίου που μπορεί να εξυπηρετηθεί εντός των ορίων ποιότητας υπηρεσίας. Ο loader που χρησιμοποιήσαμε έχει την δυνατότητα να εντοπίσει το μέγιστο



Σχήμα 4: Επίδοση της HP εφαρμογής ανά όγκο φορτίου

φορτίο που μπορεί να εξυπηρετηθεί για κάθε συγκεκριμένη παραμετροποίηση και κατανομή των

αιτημάτων αν και είναι πολύ πιθανό πως το φορτίο αυτό θα παραβιάζει τα όρια της ποιότητας υπηρεσίας που έχουμε θέσει. Μπορούμε όμως να χρησιμοποιήσουμε το μέγεθος του φορτίου αυτού ως αφετηρία και να το μειώνουμε βηματικά έως ότου τα αιτήματα να ικανοποιούνται εντός του ορίου των 10ms για το 95στο εκατοστημόριο της κάθε ομάδας. Με αυτή τη μεθοδολογία, λήφθηκαν συνολικά 6 δείγματα, ξεκινώντας από τα 280,000 αιτήματα το δευτερόλεπτο και εφαρμόζοντας μείωση στον όγκο του φορτίου σε κάθε βήμα της δειγματοληψίας κατά 10,000 αιτήματα το δευτερόλεπτο. Ως τελικό μέγεθος φορτίου, θέλουμε να επιλέξουμε μία τιμή κοντά στο όριο της ποιότητας υπηρεσίας και κάτω απο αυτό, έτσι ώστε η συμπεριφορά της HP εφαρμογής μας να είναι όσο το δυνατόν ευαίσθητη στις μεταβολές των πόρων που της διατίθενται κατά τα πειράματά μας. Καταλήγουμε λοιπόν στην επιλογή των 242,000 αιτημάτων το δευτερόλεπτο ως το ιδανικό φορτίο που μπορεί να εξυπηρετήσει καλώς η κρίσιμη εφαρμογή μας και θα αποτελεί πλέον σταθερά για την διεξαγωγή των περαιτέρω πειραμάτων μας.

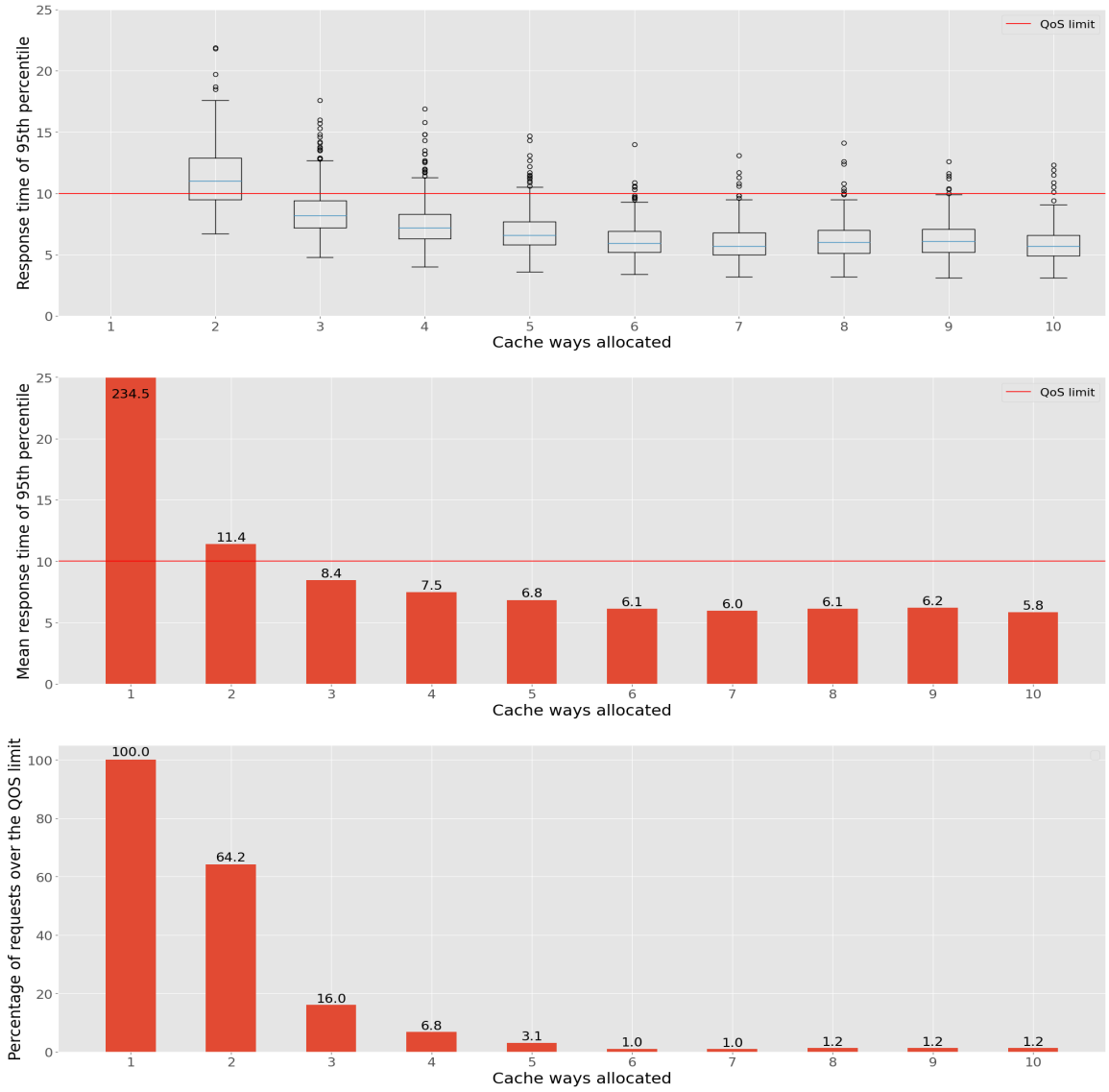
5.3.2 Ανάλυση επίδοσης ως προς χωρητικότητα LLC

Αφού προσδιορίστηκε το ανώτατο φορτίο που μπορεί να εξυπηρετηθέι, θα πραγματοποιήσουμε μία στατική ανάλυση ως προς την συμπεριφορά του χρόνου απόκρισης της HP εφαρμογής όταν έχει πρόσβαση σε διαφορετικό ποσοστό χωρητικότητας της LLC για να εξάγουμε κάποια διορατικά συμπεράσματα σχετικά με τις ανάγκες της HP εφαρμογής και τον τρόπο που αντιδράει στις μεταβολές της χωρητικότητας κρυφής μνήμης. Τα δείγματα που λήφθηκαν είναι 500 για κάθε κατανομή χωρητικότητας της LLC. Αυτό σημαίνει πως για κάθε κατανομή χωρητικότητας LLC η υπηρεσία παραγωγής φορτίου ρυθμίστηκε για να αποστείλει συνολικά 500 ομάδες των 242,000 αιτημάτων. Κάθε δευτερόλεπτο εκκινάται η αποστολή μίας νέας ομάδας ανεξαρτήτως από το αποτέλεσμα της προηγούμενης και κάθε ομάδα αναλύεται ξεχωριστά ως προς τις μετρικές του χρόνου απόκρισης.

Η τελική ανάλυση φαίνεται στο σχήμα 5 όπου παρουσιάζονται τρεις γραφικές παραστάσεις, κάθε μία εκ των οποίων απεικονίζει με διαφορετικό τρόπο την ίδια μετρική, το 95στό εκατοστημόριο του χρόνου απόκρισης της HP εφαρμογής για κάθε ομάδα αιτημάτων που κλήθηκε να διαχειριστεί. Η πρώτη γραφική αναπαριστά την μετρική αυτή σε μορφή διαγράμματος πλαισίου (boxplot), για να έχουμε καλύτερη συνολική κατανόηση της συμπεριφοράς της εφαρμογής και της κατανομής των δειγμάτων.

Το διάγραμμα πλαισίου είναι ένας τρόπος γραφικής αναπαράστασης στατιστικών δεδομένων που βασίζεται στη διάμεσο και σε διάφορα τεταρτημόρια του συνόλου δεδομένων. Στο συγκεκριμένο διάγραμμα, η γαλάζια γραμμή στη μέση κάθε πλαισίου αναπαριστά την διάμεσο του συγκεκριμένου συνόλου δεδομένων. Η πάνω πλευρά του ορθογωνίου αναπαριστά το 3ο τεταρτημόριο (Q3), ενώ η κάτω πλευρά το 1ο τεταρτημόριο (Q1). Οι κάθετες γραμμές που εκτείνονται πέρα από το ορθογώνιο εκτείνονται έως τις τιμές [Q1 – 1.5 * (Q3 – Q1), Q3 + 1.5 * (Q3 – Q1)] και τα κυκλικά σημεία πέρα από αυτές τις τιμές είναι οι ακραίες τιμές (outliers). Το δεύτερο διάγραμμα αναπαριστά τις μέσες τιμές της μετρικής μας ανά κατανομή της LLC, ενώ το τρίτο διάγραμμα αναπαριστά το ποσοστό των δειγμάτων που ήταν πάνω από το όριο της ποιότητας υπηρεσίας και άρα θεωρούνται αποτυχημένα.

Το πρώτο πράγμα που αξίζει να παρατηρήσουμε είναι πως η τιμή της μετρικής όταν απονέμουμε 1 way στην κρίσιμη εφαρμογή εικοσαπλασιάζεται και ξεφεύγει από το ύψος του διαγράμματος στις πρώτες δύο γραφικές απεικονίσεις. Επιβεβαιώνεται με αυτό τον τρόπο λοιπόν η ανάγκη για βελτιστοποίηση της αξιοποίησης της LLC σε ένα υπολογιστικό σύστημα, καθώς αποδεικνύεται πως υποβέλτιστοι διαμερισμοί μπορούν να οδηγήσουν σε τεράστιες καθυστερήσεις, ενώ μικρές μεταβολές στην διαθέσιμη χωρητικότητα της LLC μπορούν να επιφέρουν μεγάλες μεταβολές στην επίδοση. Κάτα δεύτερον, βλέπουμε πως ποιοτικά όλες οι γραφικές παραστάσεις είναι πανομοιότυπες, με την μετρική μας να συγκλίνει από τα 6 ways και μετά, με μία διακύμανση της τάξης του 5%. Οπότε μπορούμε να κάνουμε την υπόθεση πως η κρίσιμη εφαρμογή μας αποκτάει bottleneck στην



Σχήμα 5: Επίδοση της HP εφαρμογής ανά αποκλειστική χωρητικότητα LLC

επίδοσή της λόγω LLC από τα 5 ways αποκλειστικής χωρητικότητας και κάτω. Αξίζει να σημειωθεί πως οι ανάγκες σε LLC που έχει η κρίσιμη εφαρμογή και τα στατιστικά για τις μετρικές που συγκεντρώθηκαν παραπάνω δεν περιμένουμε να ανταποκρίνονται με ακρίβεια σε ενα περιβάλλον συνεκτέλεσης, όπου μπορεί να εισάγονται επιπλέον καθυστερήσεις και bottlenecks λόγω όλων των πόρων που διαμοιράζονται εφαρμογές ενός μηχανήματος. Επομένως δεν θα ήταν απολύτως ρεαλιστικό να θέτουμε τελικό στόχο ένα ποσοστό αποτυχίας 2% των αιτημάτων σε ένα πείραμα συνεκτέλεσης με BE εφαρμογές αν και βλέπουμε πως αυτό θα ήταν εφικτό στην περίπτωση της απομονωμένης εκτέλεσης ακόμα και χωρίς όλη τη διαθέσιμη χωρητικότητα σε LLC.

5.4 Συνεκτελέσεις

Στη συνέχεια περνάμε στην πειραματική ανάλυση της συμπεριφοράς των εφαρμογών σε σενάρια συνεκτέλεσης της HP εφαρμογής με πολλαπλά αντίγραφα BE εφαρμογών.

5.4.1 Επιπρόσθετα bottlenecks της επίδοσης

Όπως προαναφέραμε, στα πειράματα συνεκτέλεσης είναι αναπόφευκτη η εισαγωγή επιπλέον bottlenecks στην επίδοση της HP εφαρμογής λόγω διαμοιραζόμενων πόρων του συστήματος πέρα από την LLC ή την CPU, τους οποίους δεν έχουμε την δυνατότητα να διαχειριστούμε και να διαμοιράσουμε σε διεργασίες και εφαρμογές μέσω του WCA και του RDT. Παραδείγματα τέτοιων πόρων είναι το εύρος ζώνης κύριας μνήμης (memory bandwidth) και το εύρος ζώνης του δικτύου (network bandwidth). Με το bandwidth κύριας μνήμης αναφερόμαστε στον ρυθμό με τον οποίο ο επεξεργαστής δύναται να διαβάσει ή να αποθηκεύσει δεδομένα στην κύρια μνήμη, ο οποίος είναι άνω φραγμένος και διαμοιράζεται μεταξύ εφαρμογών καθώς χρησιμοποιούν τον ίδιο δίαυλο μνήμης (memory bus). Ειδικά στο δικό μας σύστημα που αναμένουμε υψηλό ποσοστό αστοχιών της κρυφής μνήμης κυρίως σε μη αποδοτικούς διαμοιρασμούς και κατά συνέπεια πληθώρα προσβάσεων στην κύρια μνήμη, μπορεί να προκληθεί κορεσμός του διάυλου αυτού που να οδηγήσει σε υπολειτουργία όλων των συνεκτελουμένων εφαρμογών. Ομοίως, το εύρος ζώνης δικτύου αναφέρεται στον ρυθμό με τον οποίο μπορούν να μεταφερθούν δεδομένα εντός ενός υπολογιστικού δικτύου ή μέσω κάποιας διαδικτυακής σύνδεσης. Εάν μία εφαρμογή μονοπωλεί τον συγκεκριμένο πόρο και προκαλεί κορεσμό στο δίκτυο, είναι λογικό να εισάγει καθυστερήσεις σε εφαρμογές του ίδιου δικτύου. Συγκεκριμένα η ίδια η Cloudsuite επισημαίνει πως οι ανάγκες του memcached σε εύρος ζώνης δικτύου είναι υψηλές καθώς όλα τα αιτήματα μεταξύ server και loader ικανοποιούνται μέσω δικτύου. Επομένως η επίδοση της εφαρμογής θα είναι ευαίσθητη σε επιπλέον κίνηση στο δίκτυο που μπορεί να προκληθεί από BE εφαρμογές. Έχοντας τις παραπάνω παρατηρήσεις υπόψιν, περνάμε στην πειραματική μεθοδολογία της συνεκτέλεσης των εφαρμογών.

5.4.2 Πειραματική μεθοδολογία

Κατά τη διαδικασία της συνεκτέλεσης, εφόσων επιβεβαιώσουμε πως στο περιβάλλον τρέχει η κρίσιμη εφαρμογή μας, μαζί με το WCA κατάλληλα παραμετροποιημένο για το σενάριο που εξετάζουμε, εκκινούμε την υπηρεσία παραγωγής φορτίου. Αυτή θα παράξει για συγκεκριμένη χρονική διάρκεια ένα αρχικό φορτίο στην HP εφαρμογή, προκειμένου να επιφέρουμε το σύστημα σε μια φυσιολογική κατάσταση λειτουργίας, αποκόβοντας ανωμαλίες λόγω της επανεκίννησης του πειράματος από τις μετρήσεις. Επιπλέον λόγος είναι για να διεκδικήσουν οι εφαρμογές μας ολοκληρωτικά όσους πόρους τους έχουμε αποδώσει, δηλαδή να καταλάβουν την πλήρη χωρητικότητα της κρυφής μνήμης τελευταίου επιπέδου που τους αναλογεί και να αντικαταστήσουν προυπάρχουσες εγγραφές που δεν προκάλεσαν οι ίδιες. Στην συνέχεια εκκινούμε μία BE εφαρμογή και μετράμε την επίδοση των δύο τάξεων εφαρμογών για συγκεκριμένη χρονική διάρκεια.

Όπως προαναφέραμε, καθώς δεν κάνουμε χρήση του hyperthreading, πρέπει να είμαστε βέβαιοι πως ο συνολικός αριθμός νημάτων της HP και της BE εφαρμογής δεν υπερβαίνουν τους φυσικούς πυρήνες του socket 0, που αποτελεί το socket των εφαρμογών υπό μελέτη. Αυτό, ειδικά σε ένα περιβάλλον kubernetes, όπου κάθε εφαρμογή μπορεί να έχει μεγάλο αριθμό ενεργών αντίγραφων (replicas) σημαίνει πως πρέπει να τηρήσουμε την σχέση $(replicas^{HP} * threads^{HP}) + (replicas^{BE} * threads^{BE}) = 10$. Αξίζει να αναφέρουμε επίσης πως εάν ο περιορισμός αυτός και τα χαρακτηριστικά των εφαρμογών το επιτρέπουν, είναι πιο συμφέρον να έχουμε μεγάλο αριθμό από replicas των BE εφαρμογών, ώστε να μεγιστοποιήσουμε τον ανταγωνισμό στην κρυφή μνήμη καθώς κάθε replica θα διεκδικεί εκ νέου, έναντι των υπολοίπων, την κρυφή μνήμη που έχουμε αποδώσει στο BE CloS, προσωμοιώνοντας έτσι καλύτερα ένα ρεαλιστικό περιβάλλον όπου συνεκτελούνται πολλές διεργασίες χαμηλής προτεραιότητας ταυτόχρονα, ακόμα και αν αυτές μοιράζονται σε μεγάλο βαθμό τα δεδομένα και τις εντολές που θα φέρνουν στην LLC.

Η κύρια διεργασία που εκτελεί το κάθε BE replica έχει προγραμματιστεί να επανακιννείται

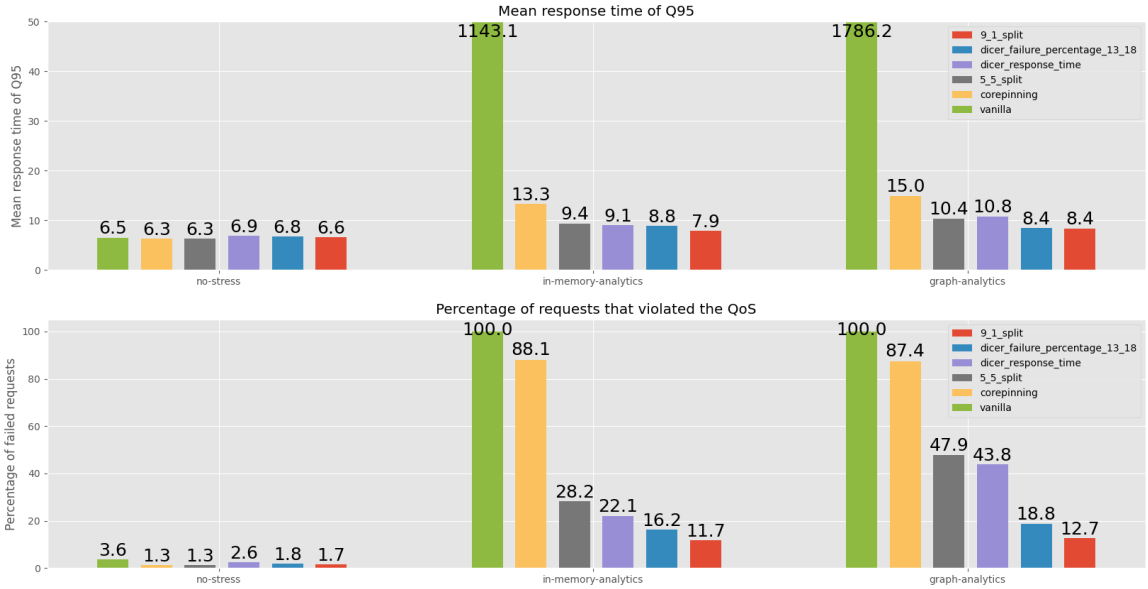
αμέσως μετά την ολοκλήρωσή της, ώστε να έχουμε μία συνεχή και επαναλαμβανόμενη ροή εντολών που πραγματοποιούν προσβάσεις στην κρυφή μνήμη και να μπορούμε να πάρουμε όσα δείγματα θέλουμε για την επίδοση των BE εφαρμογών κατά την διάρκεια του πειράματος. Οι παράμετροι των πειραμάτων που θα παραμείνουν σταθερές και αφορούν διάφορες πτυχές του οικοσυστήματός μας παρατίθενται στον παρακάτω πίνακα.

Παράμετροι περιβάλλοντος / εφαρμογών	Τιμή
memcached server threads	4
in-memory-analytics threads	1
graph-analytics threads	6
memcached server replicas	1
in-memory-analytics replicas	6
graph-analytics replicas	1
in-memory-analytics duration	25min
graph-analytics duration	25min
HP core ids	0-3
BE core ids	4-9
Utility apps core ids	10-19
Service requests per second	242000
Target response time	10ms
Quantile	q95
WCA monitoring period duration	1min

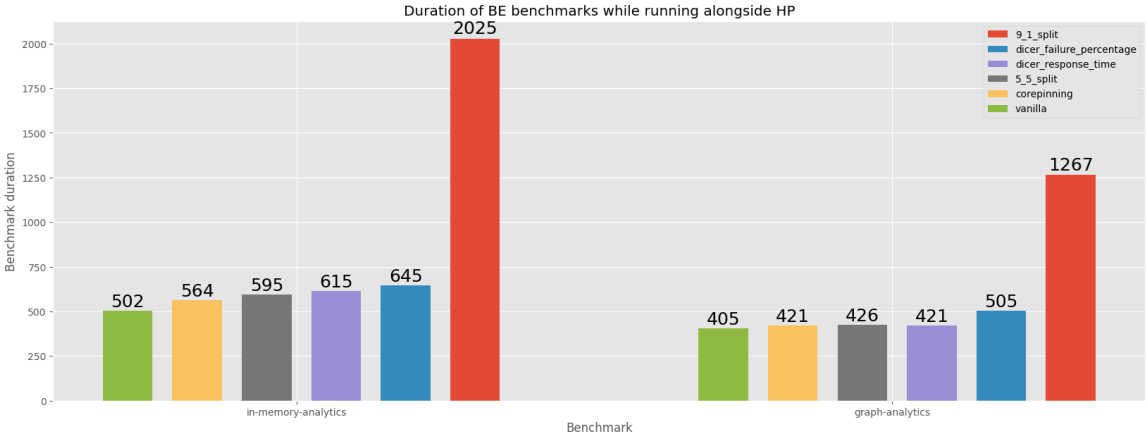
Table 2: Σταθερές παράμετροι πειραμάτων

Τα συνολικά αποτελέσματα σχετικά με την επίδοση της HP και εκάστοτε BE εφαρμογής φαίνονται στα σχήματα 6 και 7 αντίστοιχα. Συγκεκριμένα, το σχήμα 6 χωρίζεται σε δύο διαγράμματα. Το πρώτο αναπαριστά τον μέσο χρόνο απόκρισης της HP εφαρμογής σύμφωνα με το 95στο εκατοστημόριο των ομάδων αιτημάτων, για τις εξής συνεκτελέσεις: Απομονωμένη εκτέλεση (no-stress) κατά την οποία η HP εφαρμογή εκτελείται χωρίς ανταγωνισμό στους κοινόχρηστους πόρους που της έχουν αποδωθεί, συνεκτέλεση με το in-memory-analytics benchmark και συνεκτέλεση με το graph-analytics benchmark. Για κάθε μία από αυτές τις συνεκτελέσεις διερευνούμε 6 διαφορετικές πολιτικές διαμοιρασμού των πόρων και τις επιπτώσεις που επιφέρουν στην μετρική ενδιαφέροντος. Ακριβώς αντίστοιχα, στο δεύτερο διάγραμμα του σχήματος φαίνεται το ποσοστό των ομάδων αιτημάτων οι οποίες δεν ικανοποιήθηκαν εντός του ορίου ποιότητας υπηρεσίας που έχουμε θέσει, στα 10ms για το 95στο εκατοστημόριο. Ο λόγος που επιλέχτηκε και αυτή η μετρική είναι διότι αναπαριστά με μεγαλύτερη ακρίβεια το ποσοστό των χρηστών που σε ένα πραγματικό σύστημα θα έμεναν δυσαρεστημένοι απο την υπηρεσία που παρέχουν οι εφαρμογές μας, σύμφωνα με τα κριτήρια που έχουμε θέσει.

Αντίθετα, ο μέσος χρόνος απόκρισης της εφαρμογής, ενώ μπορεί να σκιαγραφήσει ποιοτικά τα συνολικά επίπεδα υπηρεσίας που παρέχουν οι εφαρμογές, δεν δύναται να αποφασίσει αν ενας χρήστης ή μια ομάδα αιτημάτων εξυπηρετήθηκε καλώς ή όχι και δεν παρέχει πολύ χρήσιμη γνώση



Σχήμα 6: Συνολική επίδοση της HP εφαρμογής ανά πολιτική διαμοιρασμού των πόρων



Σχήμα 7: Συνολική επίδοση της BE εφαρμογής ανά πολιτική διαμοιρασμού των πόρων

για τα αποτελέσματα του πειράματος χωρίς να γνωρίζουμε πόσο πραγματικά απείχαν τα ξεχωριστά δείγματα από τον μέσο όρο. Αντίστοιχα, στο σχήμα 7 φαίνεται η επίδοση των BE εφαρμογών, ως ο συνολικός χρόνος εκτέλεσης της κύριας διεργασίας τους, με βάση την κάθε πολιτική διαμοιρασμού που εφαρμόζουμε. Καταλαβαίνουμε λοιπόν πως στοχεύουμε η διερεύνηση να μας οδηγήσει σε μία πολιτική η οποία να ελαχιστοποιεί τόσο τις μετρικές της HP εφαρμογής που απεικονίζονται στο σχήμα 6, όσο και τον συνολικό χρόνο εκτέλεσης των BE εφαρμογών του σχήματος 7. Το κάθε πειραματικό σενάριο / πολιτική διαμοιρασμού πόρων θα αναλυθεί στις παρακάτω υποενότητες.

5.4.3 Unmanaged / Vanilla

Το πρώτο σενάριο (vanilla / unmanaged) ανταποκρίνεται στην προυπάρχουσα κατάσταση σε ένα kubernetes cluster, χωρίς εξωτερική παρέμβαση για αποδοτική αξιοποίηση των κοινόχρηστων πόρων. Κρίσιμη εφαρμογή και BE εφαρμογές μοιράζονται χρόνο CPU και χωρητικότητα στην

κρυφή μνήμη τελευταίου επιπέδου, συνθήκες που μπορεί να οδηγήσουν στην υπολειτουργία και των δύο τάξεων εφαρμογών και στην ασυνέπεια της επίδοσής τους καθώς ο διαμοιρασμός των πόρων είναι απρόβλεπτος και μη ελεγχόμενος. Παρ'όλα αυτά, η πολιτική αυτή εξασφαλίζει πως οι πόροι του διακομιστή δεν σπαταλώνται όπως θα γινόταν εάν αποδίδαμε σε μία εφαρμογή αποκλειστικά παραπάνω CPU ή cache από όση είχε ανάγκη για να λειτουργεί εντός των ορίων ποιότητας υπηρεσίας της. Αξίζει να αναφέρουμε πως το k8s παρέχει περιορισμένες δυνατότητες διαχείρισης πόρων όπως η CPU ή η κύρια μνήμη, μέσω του δηλωτικού του μοντέλου, με την δήλωση άνω και κάτω ορίων στους πόρους που θα διαθέτει το cluster σε συγκεκριμένες εφαρμογές. Σε καμία περίπτωση όμως δεν ανταγωνίζεται τις δυνατότητες του WCA για επίβλεψη και δυναμική αναδιανομή των κοινόχρηστων πόρων με μεγάλη έμφαση στην LLC.

Σχήμα 8: Επίδοση της HP εφαρμογής και κατανομή της κρυφής μνήμης για το vanilla σενάριο

Τα αποτελέσματα του πειράματος φαίνονται στο σχήμα 8. Στο πρώτο εκ των δύο διαγραμμάτων φαίνεται ο χρόνος απόκρισης της HP εφαρμογής μας για κάθε ομάδα αιτημάτων που λαμβάνει ανά ένα δευτερόλεπτο, τόσο για την συνεκτέλεση με το in-memory-analytics, όσο και το graph-analytics. Από την ποιοτική μορφή της συμπεριφοράς της HP εφαρμογής και για τις δύο συνεκτελέσεις είναι εμφανές πως της είναι αδύνατο να εξυπηρετήσει τον ορισμένο ρυθμό και όγκο της κίνησης με το υψηλό και ανεξέλεγκτο επίπεδο ανταγωνισμού που παρατηρείται στους κοινόχρηστους πόρους. Αξίζει επίσης να σημειώσουμε πως καθώς το memcached λειτουργεί με ένα μοντέλο ουράς ως προς την εξυπηρέτηση των αιτημάτων, ο χρόνος απόκρισης αυξάνεται με γραμμικό χαρακτήρα ως προς τον χρόνο. Αυτό συμβαίνει διότι η εφαρμογή δεν δύναται να ανταποκριθεί στον ρυθμό με τον οποίο λαμβάνει τα αιτήματα, με αποτέλεσμα να αφήνει κατάλοιπα αιτημάτων με κάθε εισερχόμενη ομάδα τα οποία επεκτείνουν την τρέχουσα ουρά και αυξάνουν την αναμονή των επόμενων αιτημάτων που ορίζουν και τον χρόνο απόκρισης της HP εφαρμογής. Αυτό φαίνεται και από το μήκος του άξονα X

του διαγράμματος 8, το οποίο ενώ θα αναμέναμε να είναι 3000 αντιστοιχώντας 1 ομάδα αιτημάτων σε κάθε δευτερόλεπτο διάρκειας του πειράματος, είναι κοντά στις 1800 ομάδες αιτημάτων ενώ οι υπόλοιπες δεν κατάφεραν να εξυπηρετηθούν.

Όπως είναι αναμενόμενο λοιπόν, κοιτώντας τα συμπυκνωμένα αποτελέσματα στα σχήματα 6 και 7, βλέπουμε πως η κρίσιμη εφαρμογή μας εξυπηρετεί καλώς το 0% των ομάδων αιτημάτων που λαμβάνει, με τον μέσο χρόνο απόκρισης να εκτοξεύεται στις τάξεις δευτερολέπτων και τα ποσοστά αποτυχημένων ομάδων αιτημάτων να βρίσκονται στο 100% και για τις δύο συνεκτελέσεις. Συνδυάζοντας αυτά τα νούμερα με το γεγονός πως ο χρόνος εκτέλεσης των BE εφαρμογών για το δεδομένο σενάριο διαμοιρασμού πόρων είναι ο ελάχιστος σε σχέση με τα υπόλοιπα σενάρια, μπορούμε να εξάγουμε ασφαλώς το συμπέρασμα πως οι BE εφαρμογές μας διεκδικούν πιο επιθετικά τους διαθέσιμους κοινόχρηστους πόρους με αποτέλεσμα η κρίσιμη εφαρμογή μας να υπολειτουργεί σε πολύ έντονο βαθμό.

Τον ανταγωνισμό ως προς την χωρητικότητα κρυφής μνήμης τελευταίου επιπέδου μπορούμε να τον διερευνήσουμε στο δεύτερο διάγραμμα του σχήματος 8 όπου απεικονίζεται η κατανάλωση LLC από την HP εφαρμογή και τα BE αντίτυπα. Τόσο κατα την συνεκτέλεση με το in-memory-analytics όσο και με το graph-analytics παρατηρούμε πως η HP εφαρμογή καταλαμβάνει 10-20% της προσβάσιμης χωρητικότητας κρυφής μνήμης, ενώ την υπόλοιπη την καταλαμβάνει η BE εφαρμογή. Βλέπουμε λοιπόν πως η πολιτική αυτή, η οποία θα ακολουθηθεί αν δεν υπάρχει εξωτερική παρέμβαση για διαχείριση των πόρων σε ενα k8s cluster, ενώ αξιοποιεί τους πόρους του συστήματος, δεν μπορεί να εξασφαλίσει συγκεκριμένα επίπεδα ποιότητας υπηρεσίας για καμία από τις συνεκτελούμενες εφαρμογές. Αυτό γιατί αφήνει τις αποφάσεις για την κατανομή των πόρων στο λειτουργικό σύστημα και τις ίδιες τις ανάγκες των εφαρμογών, μη λαμβάνοντας υπόψιν τις διαφορετικές κλάσεις προτεραιότητας που μπορεί να ανήκουν οι εφαρμογές.

5.4.4 Corepinning

Τα σύγχρονα λειτουργικά συστήματα επαναπρογραμματίζουν συνεχώς τις διεργασίες προς εκτέλεση, εκμεταλλευόμενα την πολυπύρηνη αρχιτεκτονική των επεξεργαστών για να βελτιστοποιήσουν την επίδοση του συστήματος. Κάθε φορά που μια διεργασία ή ένα νήμα προγραμματίζεται προς εκτέλεση, το λειτουργικό σύστημα μπορεί να την αναθέσει σε οποιονδήποτε πυρήνα είναι ελεύθερος. Για αυτόν τον λόγο, ο πυρήνας στον οποίο εκτελείται ένα νήμα ή μια διεργασία μπορεί να είναι διαφορετικός κάθε φορά, κάτι που σημαίνει πως ο επεξεργαστής θα πρέπει να αντιγράψει τα σχετικά δεδομένα και τις εντολές της στην κρυφή μνήμη του επεξεργαστή, εάν τα δεδομένα δεν βρίσκονται ήδη εκεί. Το cpu affinity ή corepinning επιτρέπει στις εφαρμογές να δεσμεύουν τις διεργασίες ή νήματά τους σε έναν συγκεκριμένο πυρήνα ή σε ένα σύνολο πυρήνων. Η πρακτική αυτή εκμεταλλεύεται το γεγονός ότι τα υπολείμματα μιας εκτέλεσης διεργασίας παραμένουν έγκυρα όταν η ίδια διεργασία ή νήμα προγραμματίζεται για δεύτερη φορά στον ίδιο επεξεργαστή. Έτσι, η κρυφή μνήμη πρώτων επιπέδων του κάθε επεξεργαστή μπορεί να εξακολουθεί να είναι έγκυρη όταν μια εκτέλεση νήματος προγραμματίζεται εκ νέου. Αυτό βοηθά στην κλιμάκωση της επίδοσης σε αρχιτεκτονικές πολλαπλών πυρήνων επεξεργαστών που μοιράζονται την ίδια παγκόσμια μνήμη και έχουν τοπικές κρυφές μνήμες (UMA Architecture). Επιπλέον, εφαρμόζοντάς την σε όλες τις απαιτητικές σε CPU διεργασίες του συστήματος, μπορούμε να αποτρέψουμε τον διαμοιρασμό του χρόνου CPU σε επίπεδο πυρήνα, μεταξύ νημάτων τα οποία μπορεί να προγραμματίζονταν να εκτελεστούν στον ίδιο πυρήνα και άρα θα έπρεπε να μοιραστούν χρόνο στην CPU. Στο επόμενο σενάριο (corepinning / cpu affinity), εφαρμόζουμε αυτή την πρακτική και δεσμεύουμε τις διεργασίες κάθε ομάδας πόρων σε ανεξάρτητα σύνολα επεξεργαστικών πυρήνων, εξασφαλίζοντας την επεξεργαστική ισχύ που θα έχει διαθέσιμη η κάθε εφαρμογή και περιορίζοντας τον ανταγωνισμό πλέον μόνο στην LLC.

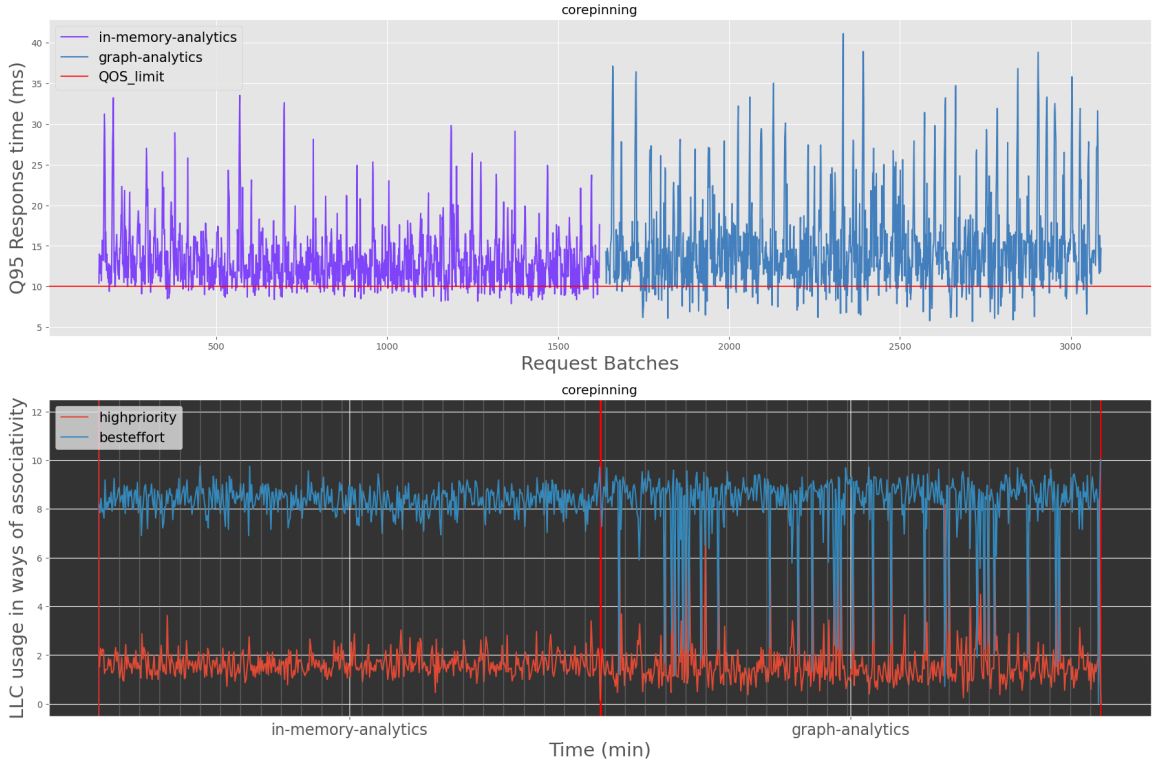
Η παραμετροποίηση του WCA η οποία καθορίζει τους κανόνες σύμφωνα με τους οποίους θα

κατανέμει τους πόρους φαίνονται στο παρακάτω απόσπασμα, σύμφωνα με το οποίο περιορίζουμε τα 4 νήματα της HP τάξης υπηρεσίας να έχουν πρόσβαση στους πυρήνες 0-3, ενώ τα 6 συνολικά νήματα της BE τάξης υπηρεσίας στους πυρήνες 4-9. Αξίζει να σημειωθεί πως ο ανταγωνισμός στα 10 τελευταία ways της LLC θα εξακολουθεί να υπάρχει σε αυτό το πειραματικό σενάριο. Τα αποτελέσματα του πειράματος αυτού φαίνονται στο σχήμα 9. Παρατηρούμε λοιπόν πως εξαλείφοντας τον ανταγωνισμό σε επίπεδο CPU, η κρίσιμη εφαρμογή πλέον δύναται να ανταποκριθεί στον ρυθμό εισερχομένων αιτημάτων καθώς ο χρόνος απόκρισης φαίνεται να είναι ανεξάρτητος σε σχέση με την μεταβολή του χρόνου. Παρά την σημαντική βελτίωση που επέφερε η απλή αυτή αλλαγή στην πολιτική κατανομής των υπολογιστικών πυρήνων, βλέπουμε πως το μεγαλύτερο ποσοστό των ομάδων αιτημάτων εξακολουθούν να υπερβαίνουν το όριο που έχουμε ορίσει ως προς την ποιότητα υπηρεσίας.

```
  - name: highpriority
  allocations:
    cpuset_mems: "0"
    cpuset_cpus: "0-3"
    rdt:
      name: "highpriority"
      l3: "L3:0=003ff"
  labels:
    clos: highpriority

- name: besteffort
  allocations:
    cpuset_mems: "0"
    cpuset_cpus: "4-9"
    rdt:
      name: "besteffort"
      l3: "L3:0=003ff"
  labels:
    clos: besteffort
```

Συγκεκριμένα, από το σχήμα 6 βλέπουμε πως το 88% των ομάδων αιτημάτων ξεπερνά το όριο ποιότητας υπηρεσίας, ενώ ο μέσος χρόνος απόκρισης είναι κατά 33 και 50% μεγαλύτερος από το επιτρεπτό για την συνεκτέλεση με το in-memory και το graph-analytics αντίστοιχα. Παράλληλα, ενδιαφέρον έχει πως από το σχήμα 7 φαίνεται πως οι χρόνοι εκτέλεσης των εφαρμογών χαμηλής προτεραιότητας δεν υπέστησαν το ίδιο δραστική μεταβολή με την κρίσιμη εφαρμογή, με τον χρόνο εκτέλεσης του in-memory-analytics να αυξάνεται κατά 12%, ενώ τον χρόνο εκτέλεσης του graph-analytics να αυξάνεται μόνο 4%. Ήδη από το πρώτο σενάριο παρέμβασης στον τρόπο με τον οποίο αναθέτει πόρους στις διεργασίες του συστήματος το λειτουργικό, βλέπουμε πως βελτιώνεται κατά πολύ η ποιότητα υπηρεσίας που παρέχει η κρίσιμη εφαρμογή, με χαμηλό κόστος στην επίδοση των εφαρμογών χαμηλής προτεραιότητας. Παρ'όλα αυτά, το γεγονός πως ο ανταγωνισμός στην LLC εξακολουθεί, όπως ήταν αναμενόμενο, να ευνοεί σε μεγάλο βαθμό τις BE εφαρμογές μας υποδεικνύει ότι στο συγκεκριμένο πρόβλημα θα πρέπει να διεισδύσουμε και στον τρόπο κατανομής της κρυφής μνήμης και να απομονώσουμε τις προσβάσεις σε αυτήν ώστε να ικανοποιήσουμε τις ανάγκες της HP εφαρμογής, όπως μας είχε υποδείξει και η διερεύνηση της υποενότητας 5.3.2



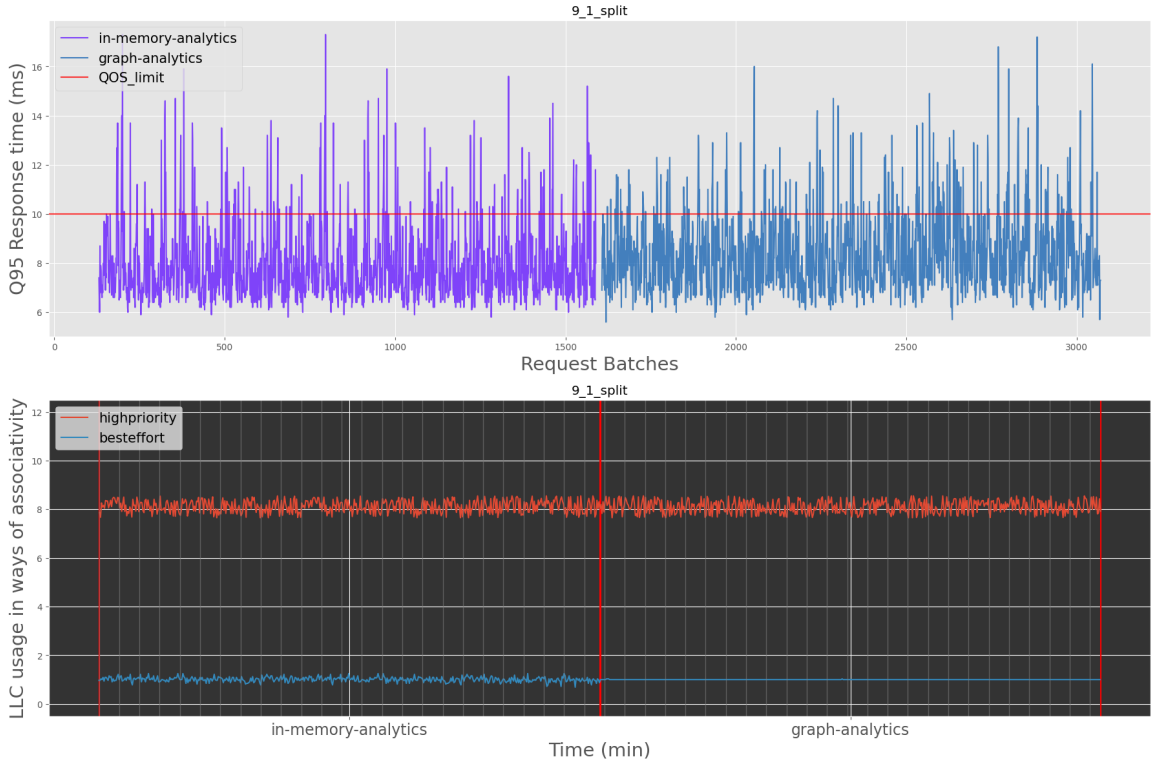
Σχήμα 9: Επίδοση της HP εφαρμογής και κατανομή της κρυφής μνήμης για το corepinning σενάριο

5.4.5 Static cache splits

Στο τρίτο σενάριο (static cache splits) απομονώνουμε με στατικούς διαμοιρασμούς και την LLC με χρήση της τεχνολογίας CAT μέσω του WCA και ερευνούμε πως επηρεάζεται τόσο η κρίσιμη εφαρμογή, όσο και τα αντίγραφα της BE εφαρμογής. Μεταξύ άλλων, ένας στατικός διαμοιρασμός που θα ερευνηθεί είναι και η κατάληψη της κρυφής μνήμης (cache takeover) απο την κρίσιμη εφαρμογή. Η πολιτική αυτή επιχειρεί να λύσει το πρόβλημα του κορεσμού της κρυφής μνήμης με την συντηρητική λύση της απόλυτης εύνοιας απέναντι στις κρίσιμες εφαρμογές, αποδίδοντας τους τη μέγιστη δυνατή χωρητικότητα κρυφής μνήμης και αφήνοντας κατά συνέπεια το ελάχιστο δυνατό για τις BE εφαρμογές. Αν και διαισθητικά θα περιμέναμε αυτός ο στατικός διαμοιρασμός να δίνει με απόλυτη συνέπεια τα καλύτερα αποτελέσματα ως προς την κρίσιμη εφαρμογή, αυτό δεν είναι εγγυημένο, καθώς ο περιορισμός της πρόσβασης των BE εφαρμογών σε πολύ μικρό ποσοστό της κρυφής μνήμης μπορεί να μεγενθύνει τον κορεσμό του bandwidth στον δίαυλο μνήμης, κατά συνέπεια καθυστερώντας την εκτέλεση των εντολών της HP εφαρμογής. Επιπλέον, αυτός ο διαμοιρασμός επιφέρει τον κίνδυνο της σπατάλης πόρων, καθώς η HP εφαρμογή πιθανώς να μπορεί να ικανοποιήσει τους περιορισμους στο QoS της με αποκλειστική πρόσβαση σε μικρότερο ποσοστό της LLC, που σημαίνει πως περιορίζεται άσκοπα η χωρητικότητα των BE εφαρμογών. Αυτό όπως είδαμε στην

στατική ανάλυση της HP εφαρμογής μπορεί να εισάγει πολύ μεγάλη καθυστέρηση στην επίδοση μίας εφαρμογής (της τάξης του 2000% στο 1 way) επομένως είναι πολύ σημαντικό να είμαστε βέβαιοι πως αξιοποιούνται στο μέγιστο όλοι οι διαθέσιμοι πόροι. Ως αποτέλεσμα, θα ερευνήσουμε και πιο δίκαιους στατικούς διαμοιρασμούς και τον τρόπο με τον οποίο η αύξηση της LLC που αποδίδουμε στα BE αντίγραφα αποσυμφορεί τις καθυστερήσεις λόγω bandwidth εξακολουθώντας να διατηρεί την επίδοση της HP και βελτιώνοντας την επίδοση των BE.

Κατάληψη κρυφής μνήμης (9-1 split) Ας ξεκινήσουμε με την ανάλυση των αποτελεσμάτων της πολιτικής κατάληψης κρυφής μνήμης στην επίδοση της HP εφαρμογής, τα οποία φαίνονται στο σχήμα 10. Καταρχάς, από το διάγραμμα κατανομής της κρυφής μνήμης είναι προφανές πως τα αντίτυπα της BE εφαρμογής έχουν περιοριστεί σε 1 way της LLC, επιτρέποντας στην κρίσιμη εφαρμογή να καταλάβει το θεωρητικό μέγιστο δυνατό από τα 10 συνολικά ways που διαθέτουμε στις εφαρμογές του συστήματος. Η επίδραση της εξάλειψης του ανταγωνισμού στον κοινόχρηστο πόρο της χωρητικότητας LLC είναι εμφανής και στον χρόνο απόκρισης της HP εφαρμογής, καθώς το γράφημα έχει μετατοπιστεί κάθετα κατά σημαντικό ποσοστό και πλέον η πλειοψηφία των ομάδων αιτημάτων ικανοποιούνται εντός του ορίου που έχουμε θέσει στην ποιότητα υπηρεσίας. Διασταυρώνοντας

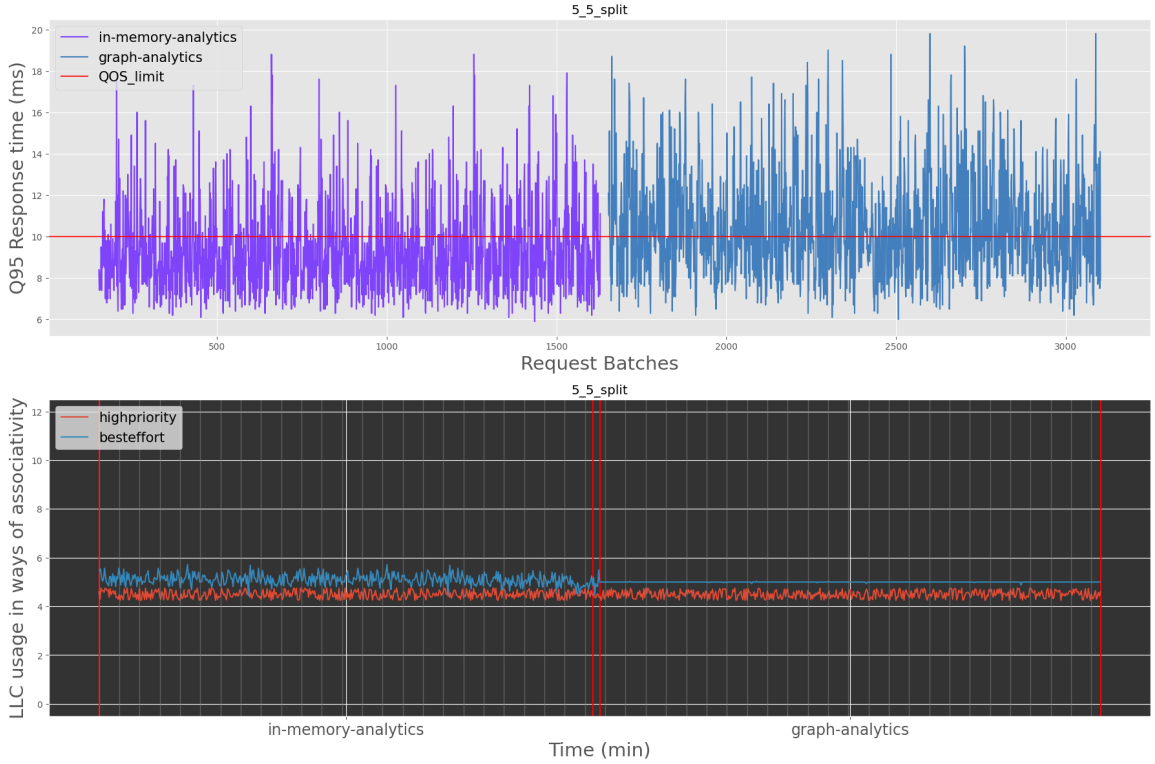


Σχήμα 10: Επίδοση της HP εφαρμογής και κατανομή της κρυφής μνήμης για τον 9-1 διαμοιρασμό

με τα νούμερα του σχήματος 6, βλέπουμε πως ο μέσος χρόνος απόκρισης πλέον κυμαίνεται 8 και 8.4ms για την συνεκτέλεση με το in-memory-analytics και το graph-analytics, με το ποσοστό αποτυχημένων αιτημάτων να πέφτει στο 11.2% και το 13.2% αντίστοιχα, τιμές χαμηλότερες από όλες διαφορετικές στρατηγικές που θα διερευνήσουμε όπως μαρτυράει το γράφημα. Μείζονος σημασίας όμως αποτελεί και το αντίκτυπο του διαμοιρασμού στις BE εφαρμογές. Βάσει του σχήματος 7, βλέπουμε δραστική αύξηση στους χρόνους ολοκλήρωσης των διεργασιών των εφαρμογών χαμηλής

προτεραιότητας, της τάξης του 400 και 300% σε σχέση με το vanilla σενάριο. Συμπεραίνουμε λοιπόν πως ο διαμοιρασμός αυτός, αν και ευνοεί την επίδοση της HP εφαρμογής, οδηγεί σε μία κατάσταση λιμοκτονίας για τις BE εφαρμογές με αποτέλεσμα αυτές να υπολειτουργούν σε πολύ έντονο βαθμό.

Ισομερής διαμοιρασμός (5-5 split) Παρά την ικανοποιητική επίδοση της πολιτικής κατάληψης της κρυφής μνήμης, εξακολουθούν να ισχύουν τα ερωτήματα που έχουμε θέσει σχετικά με τον κορεσμό του διαύλου μνήμης και την υποαξιοποίηση των πόρων του συστήματος που μπορεί αυτή να προκαλεί. Επομένως ενδιαφέρον έχει η διερεύνηση μιας πιο ισομερούς κατανομής, όπως η απολύτως δίκαια κατανομή που επιλέχθηκε για το επόμενο πείραμα, σύμφωνα με την οποία αποδίδουμε 5 ways στην κρίσιμη εφαρμογή και τα υπόλοιπα 5 ways στην BE εφαρμογή. Εξετάζοντας τα



Σχήμα 11: Επίδοση της HP εφαρμογής και κατανομή της κρυφής μνήμης για τον 5-5 διαμοιρασμό

αποτελέσματα αυτού του διαμοιρασμού από τα σχήματα 6 και 11 απορρέουν τα εξής συμπεράσματα. Καταρχάς η απομόνωση των προσβάσεων των δύο εφαρμογών στην LLC βελτιώνει σε μεγάλο βαθμό την επίδοση της HP εφαρμογής σε σχέση με το corepinning σεναρίο, αλλά δυστυχώς απέχει αρκετά από το πειραματικό βέλτιστο της κατάληψης της κρυφής μνήμης. Ενώ φαινομενικά οι μέσοι χρόνοι απόκρισης κυμαίνονται γύρω από το όριο ποιότητας υπηρεσίας, τα ποσοστά αποτυχημένων ομάδων αιτημάτων, όντας στο 29 και 48% για τις δύο περιπτώσεις συνεκτέλεσης αντίστοιχα, δεν θα μπορούσαν να είναι αποδεκτά σε ένα πραγματικό σύστημα εξυπηρέτησης χρηστών, ειδικά έχοντας την γνώση πως είναι εφικτή μία εξυπηρέτηση της τάξης του 90% για τα εισερχόμενα αιτήματα. Από την μεριά των BE εφαρμογών, το σχήμα 7 υποδεικνύει πως δεν έχουν μεγάλη ευαισθησία στην χωρητικότητα κρυφής μνήμης πάνω από κάποιο κατώφλι, καθώς συγκριτικά με το σενάριο

του corepinning κατά το οποίο οι BE εφαρμογές καταλάμβαναν περίπου τα 8 από τα 10 διαθέσιμα LLC ways, η ρίψη στα 5 ways επέφερε μία ανεπαίσθητη αύξηση στους χρόνους εκτέλεσής τους.

5.4.6 Dicer

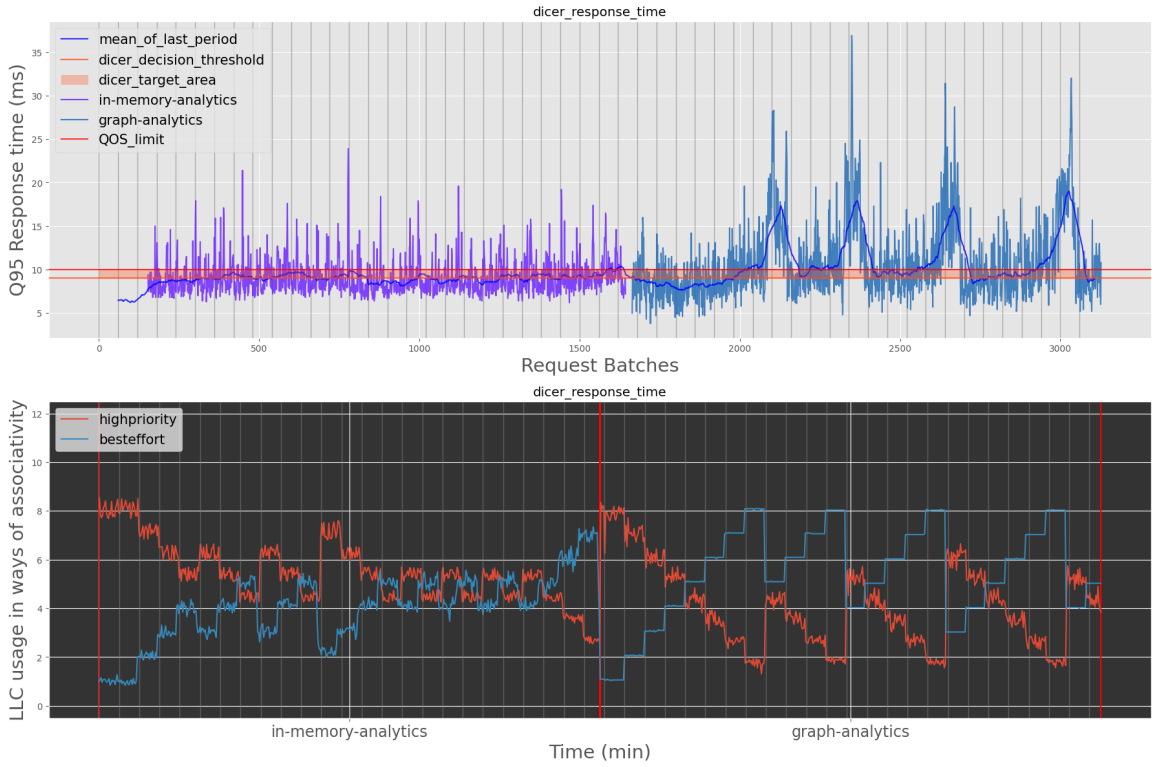
Στο τέταρτο και τελευταίο σενάριο (dynamic cache splits) θα εφαρμόσουμε έναν μηχανισμό δυναμικής κατανομής της κρυφής μνήμης, τον Dicer ο οποίος αναλύθηκε σε προηγούμενες ενότητες και θα ερευνήσουμε τις διάφορες παραμετροποιήσεις του και τις δυνατότητες που παρέχει για να λύσουμε το πρόβλημα της βέλτιστης κατανομής της κρυφής μνήμης.

Dicer με χρόνο απόκρισης Πρώτα λοιπόν θα ερευνήσουμε την παραμετροποίηση σύμφωνα με την οποία η μετρική που αντιπροσωπεύει την ποιότητα υπηρεσίας θα είναι ο μέσος χρόνος απόκρισης της HP εφαρμογής. Οι τιμές των διάφορων παραμέτρων που χρησιμοποιήθηκαν φαίνονται στον παρακάτω πίνακα. Ας αναλύσουμε λοιπόν αρχικά την συμπεριφορά της συγκεκριμένης

Παράμετροι Dicer	Τιμή
QoS metric	Mean(response time q95)
QoS lower limit	7ms
QoS upper limit	10ms
QoS drop tolerance percentage (α)	20%
phase threshold	15%
average memory bandwidth threshold	50000000000 bps

Table 3: Παράμετροι Dicer με χρόνο απόκρισης

παραμετροποίησης του μηχανισμού που απεικονίζεται στο σχήμα 12. Στο πρώτο διάγραμμα απεικονίζεται ξανά ο μέσος χρόνος απόκρισης της HP εφαρμογής. Παραθέτουμε επιπλέον τα άνω και κάτω όρια με κόκκινο και πορτοκαλί χρώμα αντίστοιχα. Επίσης με μπλέ χρώμα φαίνεται ο μέσος χρόνος απόκρισης βάσει των τελευταίων 60 δειγμάτων, δηλαδή ο μέσος χρόνος απόκρισης της τελευταίας περιόδου παρακολούθησης, μετρική βάσει της οποίας ο Dicer αποφασίζει πως θα βελτιστοποιήσει την κατανομή LLC. Όπως προαναφέρθηκε, ποιοτικά ο μηχανισμός επιχειρεί να διατηρήσει την μπλέ γραμμή εντός των δύο ορίων, κάτι το οποίο καταφέρνει αρκετά ικανοποιητικά για την συνεκτέλεση με το in-memory-analytics. Και από το σχήμα 6 φαίνεται πως πράγματι, ο μέσος χρόνος απόκρισης έχει περιοριστεί στα 9ms. Αξίζει να σχολιάσουμε βέβαια πως ακόμα και με τον μέσο χρόνο απόκρισης εντός του επιτρεπτού ορίου, βλεπουμε πως το ποσοστό αποτυχημένων αιτημάτων βρίσκεται στο 22% λόγω της διακύμανσης που αναμένεται να υπάρχει στο σύνολο των δειγμάτων. Επομένως ενώ φαινομενικά η παραμετροποίηση πέτυχε τον στόχο της, η ποιότητα υπηρεσίας που παρέχει η εφαρμογή εξακολουθεί να είναι υποβέλτιστη. Αντίθετα όμως, στην συνεκτέλεση με το graph-analytics παρατηρούμε διαφορετική συμπεριφορά. Αυτό που διαπιστώνουμε είναι πως ο μηχανισμός δεν καταφέρνει να ανιχνεύσει εγκαίρως ανησυχητικές ενδείξεις στα δείγματα που λαμβάνει για την επίδοση της HP εφαρμογής, με αποτέλεσμα να δοκιμάζει κατανομές που έχουν απότομα και επιβλαβή αποτελέσματα στην ποιότητα υπηρεσίας της. Πιο συγκεκριμένα, συγκρίνοντας τα δύο διαγράμματα του σχήματος 12, βλέπουμε πως εφαρμόζεται 4 φορές η κατανομή 2-8 (2 ways στην HP, 8 ways στην BE) κάθε μία εκ των οποίων προκαλεί μία υψηλή κορυφή στον χρόνο απόκρισης που ακολουθείται από μία ανακατανομή της κρυφής μνήμης για να επαναφέρει ο μηχανισμός την ποιότητα υπηρεσίας εντός των επιτρεπτών ορίων.



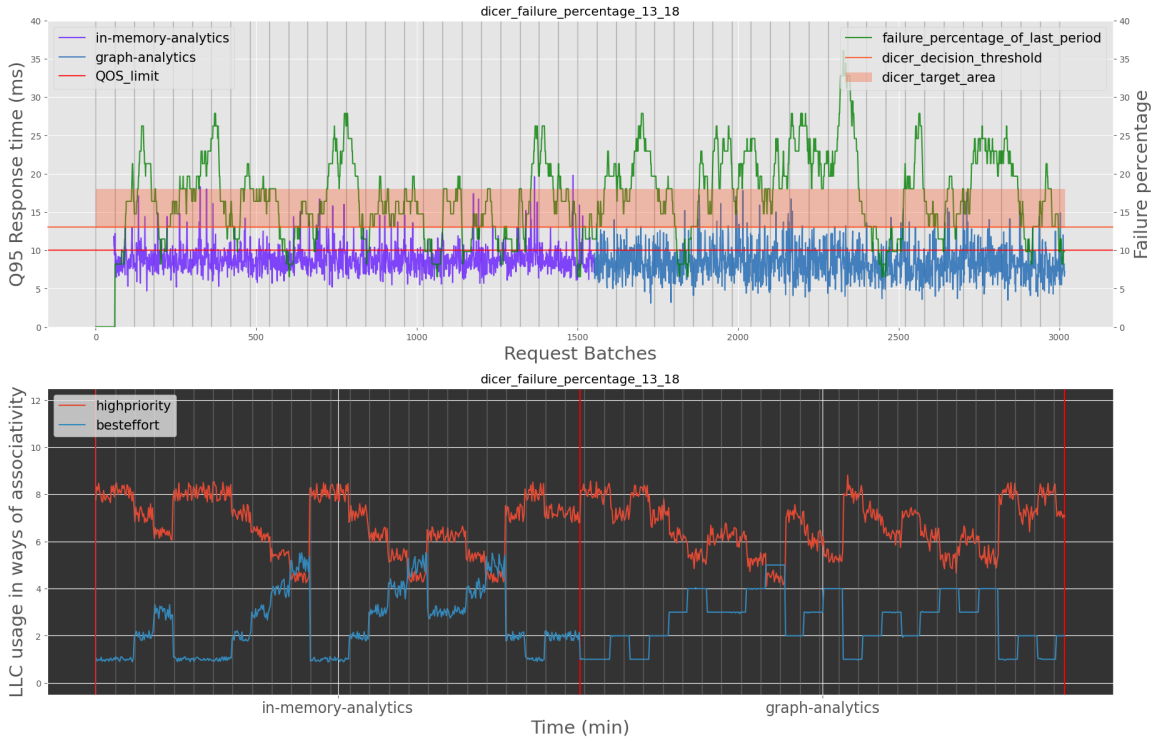
Σχήμα 12: Επίδοση της HP εφαρμογής και κατανομή της κρυφής μνήμης για τον dicer με χρόνο απόκρισης

Dicer με ποσοστό αποτυχημένων ομάδων αιτημάτων Στη συνέχεια θα εξετάσουμε την παραμετροποίηση σύμφωνα με την οποία η μετρική που αντιπροσωπεύει την ποιότητα υπηρεσίας θα είναι το ποσοστό των ομάδων αιτημάτων που δεν εξυπηρετήθηκαν εντός του ορίου ποιότητας υπηρεσίας. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν φαίνονται στον παρακάτω πίνακα.

Παράμετροι Dicer	Τιμή
QoS metric	Percentage of failed request batches
QoS lower limit	13%
QoS upper limit	18%
QoS drop tolerance percentage (α)	20%
phase threshold	15%
average memory bandwidth threshold	50000000000 bps

Table 4: Παράμετροι Dicer με ποσοστό αποτυχημένων ομάδων αιτημάτων

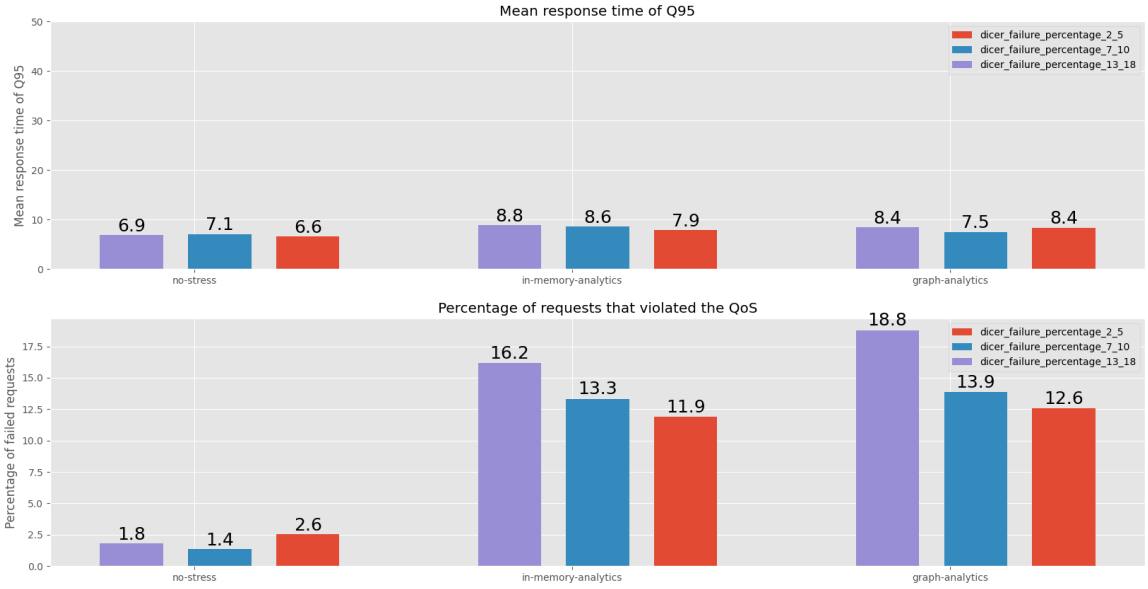
Η μορφή της συμπεριφοράς του μηχανισμού είναι παρόμοια, όπως φαίνεται στο σχήμα 13. Πλέον όμως η μετρική απόφασης είναι το ποσοστό των τελευταίων 60 ομάδων αιτημάτων που δεν ικανοποιήθηκαν εντός του ορίου για την ποιότητα της υπηρεσίας, η οποία φαίνεται με την πράσινη γραμμή στο πρώτο διάγραμμα του σχήματος. Παρατηρώντας το διάγραμμα με την κατανομή της LLC, μπορούμε να δούμε πως η παραμετροποίηση αυτή τροποποιεί τα κριτήρια απόφασης, ειδικά για την συνεκτέλεση



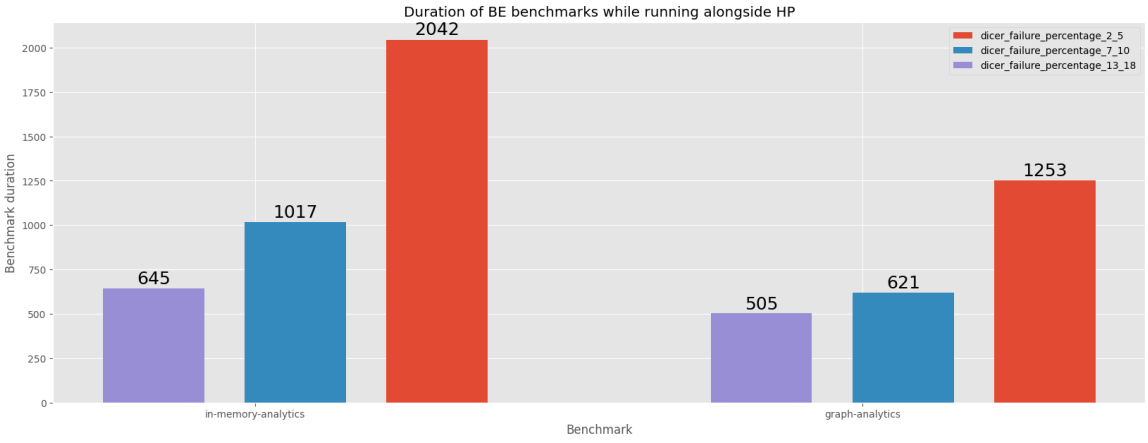
Σχήμα 13: Επίδοση της HP εφαρμογής και κατανομή της κρυφής μνήμης για τον dicer με ποσοστό αποτυχημένων ομάδων αιτημάτων

με το graph-analytics κατά την οποία πλέον αναγνωρίζει με μεγαλύτερη επιτυχία τάσεις της εφαρμογής να υποπέσει σε κατάσταση υπολειτουργίας και φροντίζει να της αποδίδει μεγαλύτερο ποσοστό χωρητικότητας κρυφής μνήμης. Κρίνοντας και από τα αποτελέσματα των σχημάτων 6 και 7, η στρατηγική αυτή είναι πιο επιτυχής, καθώς πετυχαίνει ποσοστά αποτυχίας κοντά στην κατάληψη κρυφής μνήμης, χωρίς όμως να πληγώνει σημαντικά την επίδοση της BE εφαρμογής.

Διερεύνηση παραμέτρων Στηριζόμενοι στα ευρήματα των παραπάνω ενοτήτων, είναι φανερό πως τα βέλτιστα αποτελέσματα συμψηφίζοντας την επίδοση και των δύο τάξεων εφαρμογών, τα επιφέρει η προσέγγιση του Dicer με γνώμωνα αποφάσεων το ποσοστό αποτυχημένων αιτημάτων. Ενδιαφέρον παρουσιάζει λοιπόν η πραγματοποίηση μίας περαιτέρω διερεύνησης σχετικά με τον τρόπο που θα αντιδράσει ο μηχανισμός και η επίδοση των εφαρμογών εάν μετατοπίσουμε το εύρος των ορίων ποιότητας υπηρεσίας στο οποίο προσπαθεί ο Dicer να συγκλίνει. Τα συνολικά αποτελέσματα της εν λόγω διερεύνησης φαίνονται στο σχήμα 14 για την HP εφαρμογή και στο σχήμα 15 για την BE εφαρμογή.



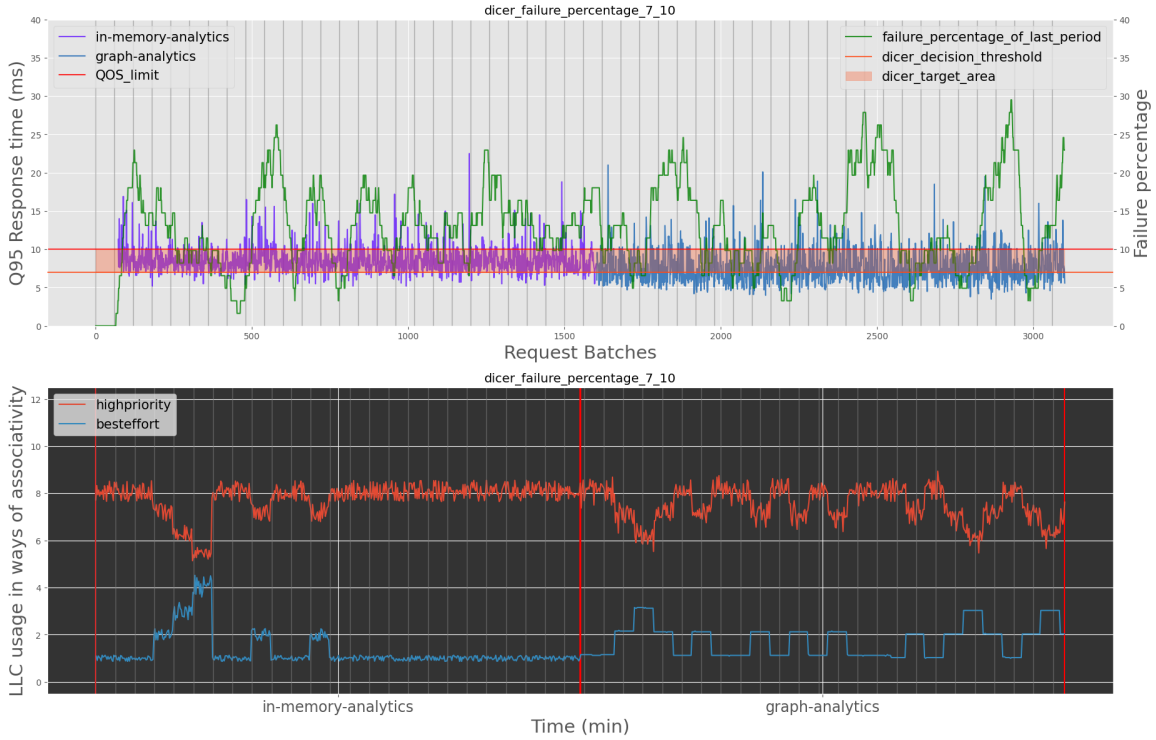
Σχήμα 14: Επίδοση της HP εφαρμογής για τις παραμετροποιήσεις του dicer με ποσοστό αποτυχημένων ομάδων αιτημάτων



Σχήμα 15: Επίδοση της BE εφαρμογής για τις παραμετροποιήσεις του dicer με ποσοστό αποτυχημένων ομάδων αιτημάτων

Ας ξεκινήσουμε λοιπόν από την πρώτη νέα παραμετροποίηση, αυτή με όρια 7-10%. Όντας πιο συντηρητική ως προς την απόδοση LLC στην BE εφαρμογή σε σχέση με την παραμετροποίηση 13-

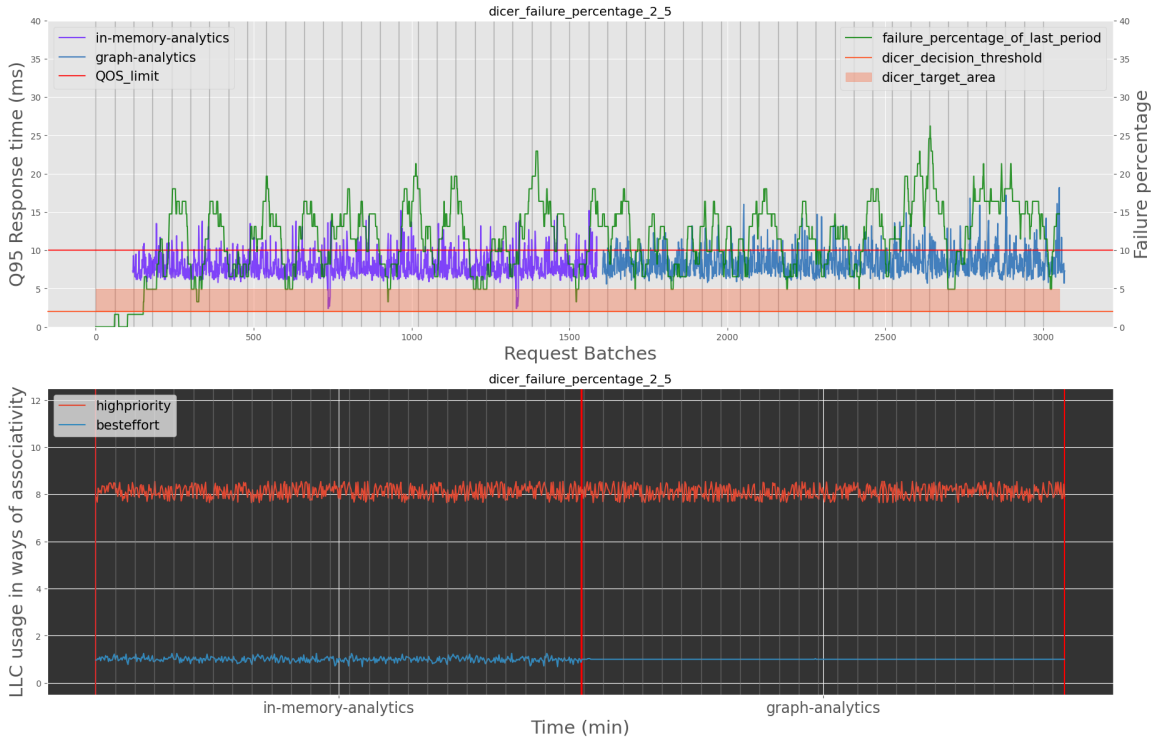
18%, παρατηρούμε στο σχήμα 16 πως είναι ελάχιστες οι περιπτώσεις που ικανοποιούν τις προυποθέσεις του μηχανισμού για να δώσει LLC στην BE εφαρμογή, ενώ πάντα επαναφέρει γρήγορα την κατανομή σε πολιτική κατάληψης της κρυφής μνήμης καθώς αδυνατεί να διατηρήσει για μεγάλη διάρκεια την ποιότητα υπηρεσίας της HP εφαρμογής στο επιθυμητό εύρος. Παρ'όλα αυτά, παρατηρούμε πως η χρονική διάρκεια για την οποία η BE εφαρμογή λειτουργεί με περισσότερα από 1 way, έχει σημαντικά αποτελέσματα στην επίδοσή της, καθώς η διάρκεια εκτέλεσής της δέχεται μείωση 50% και για τις δύο συνεκτελέσεις. Συνολικά λοιπόν, βλέπουμε πως ενώ το όριο 7-10% δεν είναι εφικτό για το δεδομένο πείραμα και σύστημα, η παραμετροποίηση παρουσιάζει μία ελαστικότητα σε σχέση με την στατική κατάληψη της κρυφής μνήμης η οποία μπορεί να ανακουφίσει τις BE εφαρμογές από καταστάσεις λιμοκτονίας και να βελτιώσει την δικαιότητα των διαμοιρασμών.



Σχήμα 16: Επίδοση της HP εφαρμογής και κατανομή της κρυφής μνήμης για τον την παραμετροποίηση με όρια 7-10

Από την άλλη μεριά, η παραμετροποίηση με όρια 2-5% εισάγει συνθήκες για την ποιότητα υπηρεσίας που είναι ανέφικτο να ικανοποιηθούν, επομένως καταλήγει να συμπεριφέρεται σαν πολιτική κατάληψης της κρυφής μνήμης.

Μπορούμε να βγάλουμε το συμπέρασμα λοιπόν, πως ενώ ο μηχανισμός που υλοποιήσαμε είναι δυναμικός και δεν έχει ανάγκη προηγούμενη γνώση του συστήματος για να προσεγγίσει μία ικανοποιητική κατανομή, είναι σημαντικό να θέτουμε ουσιώδεις μετρικές και τιμές ή εύρη τιμών ως κατώφλια αποφάσεων αλλιώς μπορεί να καταλήξουμε να υποαξιοποιούμε τους υπολογιστικούς πόρους του συστήματος ή να θυσιάζουμε επίδοση κυνηγώντας μία άπιαστη τιμή.



Σχήμα 17: Επίδοση της HP εφαρμογής και κατανομή της κρυφής μνήμης για τον την παραμετροποίηση με όρια 2-5

6 Επίλογος και μελλοντικές προεκτάσεις

Συνοψίζοντας λοιπόν, σε αυτή τη διπλωματική επιχειρήσαμε να αντιμετωπίσουμε το πρόβλημα του ανταγωνισμού για κοινόχρηστους πόρους και ειδικά για χωρητικότητα LLC σε kubernetes cluster που παρουσιάζεται όταν συνεκτελούνται εφαρμογές που ανήκουν σε διαφορετικές τάξεις ποιότητας υπηρεσίας. Μέσω του WCA που βασίζεται στις τεχνολογίες RDT της Intel, υλοποιήθηκε μία πληθώρα προσεγγίσεων, είτε στατικών είτε δυναμικών, όλες εκ των οποίων στόχευαν στην απομόνωση των προσβάσεων των δύο τάξεων εφαρμογών στην LLC προς εξάλειψη του θορύβου στην κρυφή μνήμη και βελτίωση της επίδοσης των εφαρμογών. Ως δυναμικός μηχανισμός διαμοιρασμού της LLC διερευνήθηκε εκτενώς ο Dicer και οι πιθανές παραμετροποιήσεις του σε συσχετισμό με τα όρια που έχουμε θέσει για την ποιότητα υπηρεσίας των εφαρμογών και τα αποτελέσματα που αποφέρουν στην επίδοση του συστήματος.

Η συγκεκριμένη εργασία θα μπορούσε να επεκταθεί σε πολλές κατευθύνσεις. Μια πρώτη επέκταση του μηχανισμού, αφορά την ενσωμάτωση της νέας τεχνολογίας του MBA, ούτως ώστε, να αντιμετωπίζεται επιτυχώς πιο άμεσα ο κορεσμός στο bandwidth. Μάλιστα, αντίστοιχα με τη λογική της δυναμικής εκχώρηση μνήμης LLC θα μπορούσε να επεκταθεί ο μηχανισμός ώστε επίσης δυναμικά να ορίζει και τον περιορισμό του bandwidth των πυρήνων με χαμηλής προτεραιότητας εφαρμογές. Στόχος θα ήταν η πλήρης απομόνωση των κοινόχρηστων πόρων on-chip στις εφαρμογές και η βέλτιστη εκχώρηση πόρων που καλύπτουν τις υπολογιστικές απαιτήσεις της HP εφαρμογής ενώ παράλληλα αυξάνουν την δικαιοσύνη του συστήματος.

Επιπλέον, ενδιαφέρον θα είχε η ενσωμάτωση μηχανικής μάθησης στον μηχανισμό κατανομής πόρων η οποία θα μπορούσε να λειτουργεί συμπληρωματικά με το σύνολο κανόνων που εφαρμόστηκαν στην παρούσα διπλωματική για να εντοπίζονται πιο αποδοτικά μοτίβα συμπεριφοράς και επίδοσης

των εφαρμογών και να προλαμβάνονται έκτροπες καταστάσεις.

7 Βιβλιογραφία

References

[1] Fernando Camargos, Gabriel Girard, and B Ligneris. “Virtualization of Linux servers”. In: *Proceedings of the Linux Symposium*. Vol. 2008. Citeseer. 2008.

[2] David Harris and Sarah L Harris. *Digital design and computer architecture*. Morgan Kaufmann, 2010.

[3] Lingjia Tang, Jason Mars, and Mary Lou Soffa. “Contentiousness vs. sensitivity: improving contention aware runtime systems on multicore architectures”. In: *Proceedings of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era*. 2011.

[4] Alexandros-Herodotos Haritatos et al. “LCA: A memory link and cache-aware co-scheduling approach for CMPs”. In: *Proceedings of the 23rd international conference on Parallel architectures and compilation*. 2014, pp. 469–470.

[5] Shuang Chen, Christina Delimitrou, and José F Martínez. “Parties: Qos-aware resource partitioning for multiple interactive services”. In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2019, pp. 107–120.

[6] *Intel resource director technology (Intel RDT) on 2nd generation Intel Xeon scalable processors reference manual*. 2019. URL: `https://www.intel.com/content/dam/develop/external/us/en/documents/180115-intel-rdtcascadelake-serverreferencemanual-806717.pdf`.

[7] Konstantinos Nikas et al. “DICER: Diligent cache partitioning for efficient workload consolidation”. In: *Proceedings of the 48th International Conference on Parallel Processing*. 2019.

[8] Jinsu Park, Seongbeom Park, and Woongki Baek. “CoPart: Coordinated partitioning of last-level cache and memory bandwidth for fairness-aware workload consolidation on commodity servers”. In: *Proceedings of the Fourteenth EuroSys Conference 2019*. 2019, pp. 1–16.

[9] Dhammpal Ramtake et al. “Cache Associativity Analysis of Multicore Systems”. In: *2020 International Conference on Computer Science, Engineering and Applications (ICCSEA)*. IEEE. 2020.

[10] Mohammad Shahrad, Sameh Elnikety, and Ricardo Bianchini. “Provisioning differentiated last-level cache allocations to vms in public clouds”. In: *Proceedings of the ACM Symposium on Cloud Computing*. 2021, pp. 319–334.

[11] Weijia Song et al. “CacheInspector: Reverse Engineering Cache Resources in Public Clouds”. In: *ACM Transactions on Architecture and Code Optimization (TACO)* 18.3 (2021), pp. 1–25.

[12] Qitian Zeng, Kyle C Hale, and Boris Glavic. “Playing Fetch with CAT: Composing Cache Partitioning and Prefetching for Task-based Query Processing”. In: *Proceedings of the 17th International Workshop on Data Management on New Hardware (DaMoN 2021)*. 2021, pp. 1–5.

[13] Parul Sohal et al. “A Closer Look at Intel Resource Director Technology (RDT)”. In: *Proceedings of the 30th International Conference on Real-Time Networks and Systems*. 2022.