Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Εργαστηριο Υπολογιστικων Συστηματων

Μελέτη και βελτιστοποίηση επίδοσης μικροϋπηρεσιών σε σύγχρονα συστήματα

Διπλωματικη Εργασια

του

ΚΩΝΣΤΑΝΤΙΝΟΥ ΣΦΑΚΙΩΤΑΚΗ

Επιβλέπων: Γεώργιος Γκούμας
Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβρης 2023

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Υπολογιστικών Συστημάτων

Μελέτη και βελτιστοποίηση επίδοσης μικροϋπηρεσιών σε σύγχρονα συστήματα

Διπλωματικη Εργασια

του

Κωνσταντίνου Σφακιωτάκη

Επιβλέπων: Γεώργιος Γκούμας
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 28η Σεπτεμβρίου 2023.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Γεώργιος Γκούμας	Νεκτάριος Κοζύρης	Διονύσιος Πνευματικάτος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβρης 2023

(Υπογραφή)

.....

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

Περίληψη

Αντικείμενο της διπλωματικής εργασίας είναι η μελέτη και βελτιστοποίση της απόδοσης μικροϋπηρεσιών σε σύγχρονα συστήματα. Οι μικροϋπηρεσίες αποτελούν μία αρχιτεκτονική σχεδιάσης εφαρμογών ιδιαίτερα διαδεδομένη τα τελευταία χρόνια εξαιτίας των πολλών πλεονεκτημάτων που παρέχει στο χώρο του νέφους. Η απότομη διάδοσης της την έχει κάνει στόχο έρευνας, με σκοπό να βρεθούν βελτιστοποιήσεις τόσο στις εφαρμογές που την αξιοποιούν όσο και στα μηχανήματα που τη φιλοξενούν.

Ένα τομέας με ιδιαίτερες προκλήσεις είναι η μελέτη της απόδοσης των μικροϋπηρεσιών υπό το πρίσμα της μικροαρχιτεκτονικής του συστήματος που θα φιλοξενηθεί, τόσο στη συλλογή δεδομένων όσο και στο χαρακτηρισμό της απόδοσης της. Δεδομένα που αποτελούν πρόκληση να συλλεχθουν μπορούν να θεωρηθούν μετρικές της σωλήνωσης του επεξεργαστή, όπως αστοχίες στις κρυφές μνήμες ή καθυστερήσεις σε συγκεκριμένα μέρη της σωλήνωσης.

Η εργασία επικεντρώνεται κυρίως στην υλοποίηση ενός συστήματος για τη συλλογή δεδομένων για την απόδοση των μικροϋπηρεσιών και ερμηνεία αυτών ώστε μέσω καλύτερης τοποθέτησης των μικροϋπηρεσιών στους διαθέσιμους επεξεργαστές να επιτευχθεί βελτίωση της απόδοσης. Η αξιολόγηση γίνεται με βάση τις εντολές ανά κύκλους και δευτερευώντος με τη χρονική απόκριση ουράς. Γίνονται μετρήσεις των προτεινόμενων τοποθετήσεων ελέγχεται αν οι προτάσεις τελικά δίνουν καλύτερα αποτελέσματα αναφορικά με τη χρονική απόκριση της εφαρμογής στα αιτήματα και το IPC.

Λέξεις Κλειδιά

Μικροϋπηρεσία, Νέφος, Μαχανική Απόδοσης, Μέτρο Σύγκρισης, Μικροαρχιτεκτονική, Καταχωρητές απόδοσης

Abstract

The aim of this thesis is to study and optimize the performance of microservices in modern systems. Microservices are an application design architecture and the advantages it provides in the cloud space has made it a very popular choice and a target of research to optimize its performance.

Studying the performance of microservices in the light of microarchitecture is a challenge both to collect data and to characterize it. Data related to processor pipelining, cache failures or delays in part of the pipelining can provide important information about the operation of the system hosting the microservices. Additional data can be collected while stressing various resources used by the application to determine which of them show resistance and which show sensitivity.

The paper mainly focuses on implementing a system to collect data on the performance of microservices and interpreting it to arrive at a better placement of microservices on the available processors. The evaluation is based on instructions per cycles and in a second phase on the tail latency to compare the two methods. Measurements of the proposed placements and various random measurements are made to see if the proposals finally give better results regarding the application's temporal response to requests and IPC.

Keywords

Microservice, Cloud, Performance, Benchmark, Microarchitecture, Performance counters

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή κ. Γεώργιο Γκούμα.

Επίσης ευχαριστώ ιδιαίτερα την κ. Νικέλα Παπαδοπούλου για

την καθοδήγηση και για τη βοήθεια στην εκπόνηση της διπλωματικής εργασίας μου.

Περιεχόμενα

Περίληψη **1**

Abstract **3**

Ευχαριστίες **5**

Περιεχόμενα **10**

Κατάλογος Σχημάτων **13**

Κατάλογος Πινάκων **15**

1 Εισαγωγή **17**

1.1 Αντικείμενο της διπλωματικής 18

1.2 Συνεισφορά 18

1.3 Οργάνωση του τόμου 19

2 Συγγενικές εργασίες **21**

2.1 Εισαγωγή 21

2.2 Πίεση που ασκούν οι μικροϋπηρεσίες στους πόρους ενος συστήματος . 21

2.3 Απομόνωση container για μελέτη της συμπεριφοράς του 22

2.4 Διαφοροποίηση εργασίας 22

3 Θεωρητικό υπόβαθρο 25

3.1 Εισαγωγή 25

3.2 Cpu pipeline, performance counters 25

3.3 Perf tool 30

3.4 Latency / Tail latency 32

3.5 DeathStarBench, microservices applications 33

3.6 Load Genarator 36

3.7 Stress testing, IBench suite 37

3.8 Δυσκολία μέτρησης απόδοσης 39

3.8.1 Εικονικοποίηση και διαμοιρασμός πόρων 39

3.8.2 Θορυβώδεις γείτονες και μεταβλητότητα επιδόσεων 39

3.8.3 Περιορισμένη ορατότητα και έλεγχος 40

3.9 Προσεγγίσεις λύσεων των προκλήσεων μέτρησης 40

3.10 Microservices architecture communication 41

4 Απόδοση μικροϋπηρεσιών σε σύγχρονα συστήματα 43

4.1 Εισαγωγή 43

4.2 Διαδικασία ανάλυσης απόδοσης εφαρμογών 43

4.3 Contention Interference 45

4.3.1 Κατανόηση του ανταγωνισμού και της παρεμβολής 46

4.3.2 Επιπτώσεις του ανταγωνισμού και της παρεμβολής 46

4.3.3 Σημασία της επίλυσης της διαμάχης και της παρεμβολής 47

5 Μεθοδολογία **49**

5.1 Εισαγωγή 49

5.2 Απομώνωση μικροϋπηρεσιών 49

5.3 Topdown analysis 51

5.4 Εξευρένηση διαφορετικών placements 51

5.5 Τεχνικές δυσκολίες 52

5.5.1 Perf events 52

5.5.2 Δυσκολίες στην εγκατάσταση 52

5.5.3 Reservation Mongodb 54

5.5.4 Jaegar tracing 54

6 Πειραματική Αξιολόγηση **57**

6.1 Παράμετροι αξιολόγησης 57

6.2 Οργάνωση πειραμάτων 57

6.2.1 Φιλοξενία εφαρμογών 57

6.2.2 Εκτέλεση των μέτρησεων 58

6.2.3 Εισαγωγή δοκιμών καταπόνησης 60

6.3 Αποτελέσματα της μελέτης 60

6.3.1 IPC απομονωμένης μικροϋπηρεσίας 60

6.3.2 TopDown ανάλυση 66

6.3.3 Εξέταση τιμών των διαφόρων cache 71

6.3.4 Tail Latency σύγκριση με IPC 74

6.3.5 Έλεγχος για βελτίωση του latency 74

7 Επίλογος **83**

7.1 Σύνοψη και συμπεράσματα 83

7.2 Μελλοντικές επεκτάσεις 84

Βιβλιογραφία **86**

Γλωσσάριο **89**

Κατάλογος Σχημάτων

3.1 TopDown Breakdown 26

3.2 Topdown Hierarchy 27

3.3 Topdown hotspot 29

3.4 Tail Latency 33

3.5 Social network 35

3.6 Media Microservices 35

3.7 Hotel reservation 36

3.8 Microservive communication patterns 42

3.9 Microservice architecture 42

4.1 Επίπεδα εξερεύνησης απόδοσης 45

6.1 Geolocation microservice IPC 62

6.2 Geolocation microservice IPC 62

6.3 Frontend microservice IPC 63

6.4 Frontend microservice IPC 63

6.5 Profile microservice IPC 64

6.6 Profile microservice IPC 64

6.7 Rate microservice IPC 65

6.8 Rate microservice IPC 65
6.9 Recommendation microservice IPC 66
6.10 Recommendation microservice IPC 66
6.11 Reservation microservice IPC 67
6.12 Reservation microservice IPC 67
6.13 Search microservice IPC 68
6.14 Search microservice IPC 68
6.15 User microservice IPC 69
6.16 User microservice IPC 69
6.17 Cycle breakdown 70
6.18 L1D MPKI 71
6.19 L1I MPKI 72
6.20 L2 MPKI 72
6.21 L3 MPKI 73
6.22 DTLB MPKI 73
6.23 Tail Latency Frontend 75
6.24 Tail Latency Geo 75
6.25 Tail Latency Rate 76
6.26 Tail Latency Recommendation 76
6.27 Tail Latency Reservation 77
6.28 Tail Latency Search 77
6.29 Tail Latency User 78
6.30 Tail Latency Profile 78
6.31 Latency 80

6.32 Tail Latency 80

Κατάλογος Πινάκων

3.1 Hardware events για μέτρηση κύκλων (Πηγή [13]) 28

5.1 Perfmon events 52

6.1 Microbenchmarks placement 61

6.2 Toplev results 71

6.3 Different scenarios placements 81

Κεφάλαιο 1

Εισαγωγή

Τα datacenter έχουν αρχίσει και φιλοξενούν όλο και μεγαλύτερο αριθμό εφαρμογών. Εφαρμογές που στην πλειοψηφία τους παρουσιάζουν μεγάλο βαθμό ετερογένειας, με χρήση όλο και περισσότερων τεχνολογίων, με τις περισσότερες από αυτές να έχουν αυστηρά performance constraints (latency, tail-latency, throughput, availability), και πολλές ομάδες να δουλεύουν σε διαφορετικά μέρη της εφαρμογής ώστε να γίνονται ενημερώσεις και αναβαθμίσεις.

Από την πλευρά των παρόχων ένα κύριο πρόβλημα αποτελεί η διάθεση και αποδοτική χρήση των διαθέσιμων πόρων. Πεδίο έρευνας αποτελεί για αυτό το λόγο η βέλτιστη αξιοποίηση των πόρων και ο περιορισμός του χρόνου που μένουν αδρανείς. Για να επιτευχθούν όλα τα παραπάνω και ταυτόχρονα να διευκολυνθεί η επίβλεψη των εφαρμογών έχει αρχίσει και υιοθετείται ένα νέο μοντέλο, αυτό των μικροϋπηρεσιων (microservices). Μικροϋπηρεσίες είναι ενα αρχιτεκτονικό στυλ ανάπτυξης λογισμικού. Χωρίζει την εφαρμογή σε ένα γράφο υπηρεσιών, με κάθε υπηρεσία να αναλαμβάνει ένα κομμάτι της λειτουργικότητας της όλης εφαρμογής.

Βασικά χαρακτηριστικά των υπηρεσιών αυτών αποτελούν η ελαφριά σύζευξη μεταξύ τους και η ανεξαρτησία στην κλιμάκωση και στις ενημερώσεις. Ένα από τα μεγαλύτερα πλεονεκτήματα είναι ότι οι μικρουπηρεσίες μπορούν να αναπτυχθούν ανεξάρτητα. Μια ομάδα μπορεί να ενημερώσει μια υπάρχουσα υπηρεσία χωρίς να ανακατασκευάσει και να επανεκκινήσει ολόκληρη την εφαρμογή. Είναι εφικτό να αναπτυχθούν και να συντηρηθούν από μικρές ομάδες, δίνοντας τους τη δυνατότητα να χρησιμοποιήσουν οποιαδήποτε

γλώσσα προγραμματισμού ή προγραμματιστικό μοντέλο.

Ενώ η αρχιτεκτονική μικρουπηρεσιών προσφέρει αρκετά πλεονεκτήματα , έχει επίσης ορισμένα μειονεκτήματα που πρέπει να ληφθούν υπόψη πριν από την εφαρμογή της (Πηγή [16]):

Η αρχιτεκτονική μικρουπηρεσιών είναι πιο πολύπλοκη από την παραδοσιακή μονολιθική αρχιτεκτονική, καθώς περιλαμβάνει πολλαπλές υπηρεσίες που επικοινωνούν μεταξύ τους. Αυτό μπορεί να καταστήσει την ανάπτυξη και τη συντήρηση πιο πολύπλοκες και να απαιτήσει πρόσθετους πόρους και τεχνογνωσία.

Με την αρχιτεκτονική μικρουπηρεσιών, οι υπηρεσίες κατανέμονται σε πολλαπλούς διακομιστές, γεγονός που μπορεί να δημιουργήσει προκλήσεις σχετικά με τη συνέπεια των δεδομένων, την καθυστέρηση του δικτύου και την ανακάλυψη υπηρεσιών. Η δοκιμή και η αποσφαλμάτωση μπορεί να είναι πιο πολύπλοκες σε μια αρχιτεκτονική μικρουπηρεσιών, καθώς υπάρχουν πολλαπλές υπηρεσίες που πρέπει να δοκιμαστούν και να αποσφαλματωθούν ανεξάρτητα, καθώς και η δοκιμή ολοκλήρωσης μεταξύ των υπηρεσιών. Επίσης υπάρχει πρόσθετη επιβάρυνση που σχετίζεται με τη διαχείριση πολλαπλών υπηρεσιών, συμπεριλαμβανομένης της ανάπτυξης, της κλιμάκωσης και της παρακολούθησης.

1.1 Αντικείμενο της διπλωματικής

Στόχος της παρούσας διπλωματικής είναι να εξερευνήσει και να μελετήσει την αποδόση στην αρχιτεκτονική των microservices. Συγκεκριμένα να ερευνήσει πως η μικροαρχιτεκτονική επηρεάζει την συνολική απόδοση του συστήματος και αν τα συμπεράσματα από αυτή τη μελέτη μπορούν να οδηγήσουν σε καλύτερη τοποθέτηση της εφαρμογής στους υπάρχοντες επεξεργαστές με στόχο της βελτίωση του latency και του tail latency ενώ το utilization κρατιέται σε υψηλό επίπεδο. Τελικά αν μπορούμε χαρακτηρίζοντας την απόδοση των μικρουπηρεσιών ατομικά να φτάσουμε σε μία συνολικά καλύτερη εμπειρία για το χρήστη.

1.2 Συνεισφορά

Η συνεισφορά της διπλωματικής συνοψίζεται ως εξής:

1. Μελετήθηκαν οι performance counters του επεξεργαστή

2. Υλοποιήθηκαν προτάσεις για τοποθέτηση των microservices, με γνώμονα το IPC και το tail latency
3. Αξιολογήθηκε η επίδοση των προταθέντων τοποθετήσεων
4. Μελετήθηκε που σπαταλόνται οι περισσότεροι κύκλοι στη σωλήνωση του επεξεργαστή
5. Γίνανε προτάσεις για επέκταση της διπλωματικής

1.3 Οργάνωση του τόμου

Η παρούσα διπλωματική εργασία χωρίστηκε σε 7 ενότητες. Η πρώτη ενότητα αποτελεί την εισαγωγή στο πρόβλημα που αντιμετωπίζεται και θα μελετηθεί. Εργασίες σχετικές με το αντικείμενο της διπλωματικής παρουσιάζονται στο Κεφάλαιο 2. Το Κεφάλαιο 3 δίνει ένα θεωρητικό υπόβαθρο και παρουσιάζει έννοεις και εργαλεία απαραίτητα για την κατανοηση της διπλωματικής. Στο Κεφάλαιο 4 αναφερόμαστε στα προβλήματα που αντιμετωπίζει και σε μεθόδους που θα χρησιμοποιηθούν χωρίς να αναπτυχθούν τεχνικές λεπτομέρειες. Στο Κεφάλαιο 5 αναπτύσσεται πλέον με τεχνικούς όρους η μεθοδολόγια που ακολοθούθηκε καθώς και τεχνικά προβλήματα που αντιμετωπίστηκαν. Στο Κεφάλαιο 6 έχουμε τα αποτελέσματα με μορφή γραφικών και ανάλυση τους. Στο Κεφάλαιο 7 έχουμε τον επίλογο της διπλωματικής με σύνοψη των συμπερασμάτων και πρόταση μελλοντικών επεκτάσεων.

Κεφάλαιο 2

Συγγενικές εργασίες

2.1 Εισαγωγή

Παρακάτω θα παρατεθούν εργασίες με σχετικό περιεχόμενο με την παρούσα διπλωματική εργασία. Κάθε υποενότητα έχει τίτλο την θεματική που η παρούσα εργασία και η συγγενική έχουν κοινό πρίσμα. Και τελός παρουσιάζεται η προσφόρα της εργασίας και το ερευνητικό κενό που καλύπτει.

2.2 Πίεση που ασκούν οι μικροϋπηρεσίες στους πόρους ενος συστήματος

Η εργασία με τίτλο An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems που δημοσιεύθηκε από τους Gan, Yu and Zhang, Yanqi and Cheng, Dailun and Shetty, Ankitha and Rathi, Priyal and Katarki, Nayan and Bruno, Ariana and Hu, Justin and Ritchken, Brian and Jackson, Brendon and others (Πηγή [7]), αναλύει τις επιπτώσεις των μικρουπηρεσιών που έχουν σε ολόκληρη τη στοίβα των συστημάτων νέφους. Για αυτό το λόγο και δημιούργησαν τη σουίτα deathstarbench που είναι και η σουίτα που θα εξεταστεί σε αυτή τη διπλωματική. Οι επιπτώσεις που επικεντρώνεται η συγγενική εργασία αφορούν επιπτώσεις στην αρχιτεκτονική, στη δικτύωση και τα λειτουργικά συστήματα, καθώς και ερευνόνται οι προκλήσεις τους σε σχέση με τη διαχείριση συστάδων, και οι συμβιβασμοί τους από την άποψη του σχεδιασμού εφαρμογών και των frameworks προγραμματισμού.

2.3 Απομόνωση container για μελέτη της συμπεριφοράς του

Η εργασία με τίτλο PARTIES: QoS-Aware Resource Partitioning for Multiple Interactive Services που έχει δημοσιευτεί από τους Chen, Shuang and Delimitrou, Christina and Mart ınez, Jos e F. (Πηγή [2]) αναλύει την ανάγκη για μείωση των καθυστερήσεων από τον ανταγωνισμό πόρων που συμβαίνει στις σύγχορες multitenant υποδομές νέφους. Για τη λύση αυτού του προβλήματος αναπτύσσουν το σύστημα PARTIES, έναν διαχειριστή πόρων με γνώμονα την ποιότητα υπηρεσιών.

Η παραπάνω εργασία χρησιμοποιεί παρόμοια μεθοδολογία με την παρούσα εργασία για να μετρήσει την ευαισθησία των εφαρμογών υπό το πρίσμα του κατά πόσο επηρεάζεται η χρονική απόκριση ουράς από τον ανταγωνισμό για κοινούς πόρους. Γίνεται χρήση τεχνικών απομόνωσης και εισαγωγή stressing microbenchmarks για να μετρηθεί η αντοχή του απομωνομένου service στη πίεση των πόρων.

Η παρούσα διπλωματική ακολουθεί παρόμοια προσσέγγιση απλώς η ευαισθησία εξετάζεται κυρίως υπό το πρίσμα των εντολών ανά κύκλους, η οποία θα αποτελέσει και τη βασική μετρική αξιολόγησης ενώ δευτερευόντως θα εξετατεί και η χρονική απόκριση ουράς.

2.4 Διαφοροποίηση εργασίας

Οι προηγούμενες εργασίες αξιολογούν την απόδοση των συστημάτων που φιλοξενούν τα microservices με βάση τη χρονική απόκριση ουράς και χωρίς να παρουσιάζουν σε βάθος τα απολέσματα που αφορούν τα συγκεκριμένα microservices, ιδιαίτερα σε μικροαρχιτεκτονικό επίπεδο. Η συγκεκριμένη εργασία ακολουθεί μία διαφορετική προσέγγιση που είναι να αναζητήσει και να παρουσιάσει όλα τα αποτελέσματα για διάφορες σημαντικές μετρικές χαμηλού επιπέδου που αφορούν τη σωλήνωση του επεξεργαστή (Πηγές [12], [21], [6]). Η μετρική αξιολόγησης που χρησιμοποιείται είναι η IPC σε αντίθεση με το tail latency των προηγούμενων εργασιών και αναζητούνται στατικές τοποθετήσεις της εφαρμογής στους διαθέσιμους πυρήνες του συστήματος και προτείνει μία τέτοια τοποθέτηση της εφαρμογής για βελτίωση του latency και του tail latency. Τέλος η μέτρηση γίνεται με τους μετρητές απόδοσης του επεξεργαστή αρχικά με το perf tool (Πηγή [15]) και το toplev (Πηγή [14]), ενώ οι προηγούμενες εργασίες χρησιμοποιούν

το intel v-tune (Πηγή [11]).

Κεφάλαιο 3

Θεωρητικό υπόβαθρο

3.1 Εισαγωγή

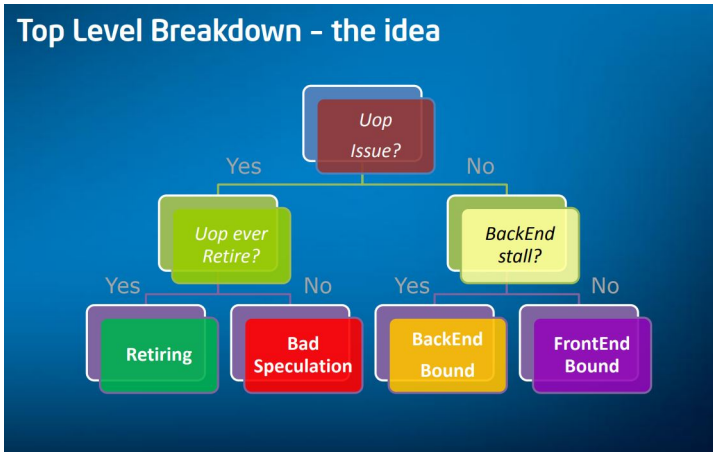
Παρακάτω θα αναλυθούν έννοεις και εργαλεία που η παράθεση τους είναι βασική για την κατανόηση της παρούσας διπλωματικής εργασίας. Συγκεκριμένα θα αναλυθούν τα εργαλεία που έγιναν οι μετρήσεις καθώς και οι μετρικές, η παραγωγή αιτημάτων, η σουίτα που στρέσαρε τα συστήμα για να έχουμε οριακές μετρήσεις.

3.2 Cpu pipeline, performance counters

Για την ανάλυση της απόδοσης σε αρχιτεκτονικό επίπεδο είναι σημαντικό να μετρηθούν και να αναλυθούν βασικές μετρικές του pipeline των επεξεργαστών. Άρα κρίνεται σημαντικό να κατανοήσουμε από τι αποτελείται το pipeline και σε ποιο σημείο του μπορούν συμβούν καθυστερήσεις.

Το pipeline των σύγχρονων επεξεργαστών χωρίζεται κυρίως σε δύο μέρη το frontend και το backend.

Το frontend είναι υπεύθυνο για να φέρει τις εντολές από τη μνήμη και να τις μεταφράσει σε μίκρο-διαδικασίες (uops). Συγκεκριμένα προβλέπεται από τον branch predictor η επόμενη διεύθυνση που θα γίνει fetch, γίνεται fetch των cache lines, αναλύονται σε εντολές και γίνονται decode σε uops τα οποία αργότερα θα εκτελεστούν. Τέλος τα uops τροφοδοτούνται σε μία ουρά-έτοιμων uops και από εκεί στο backend. Με τη σειρά του



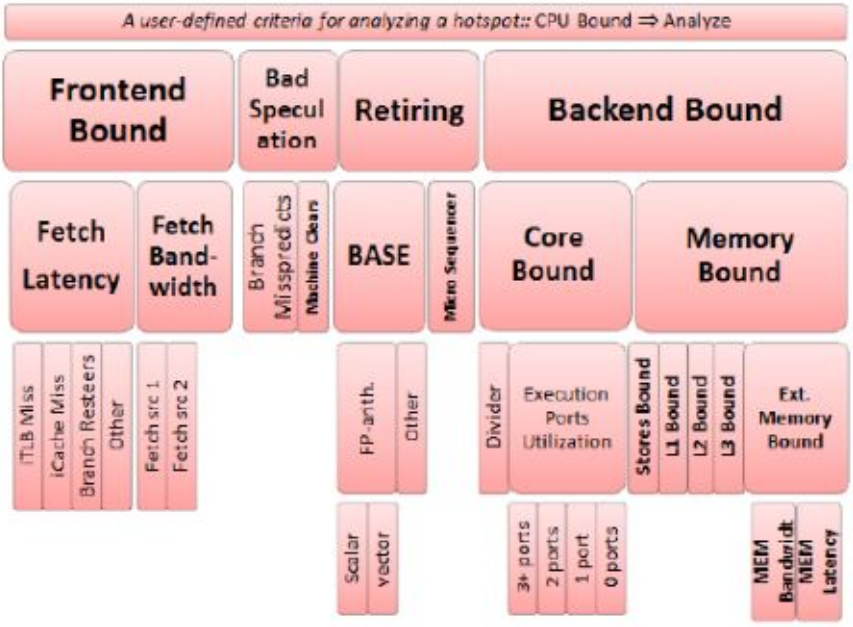
Σχήμα 3.1: TopDown Breakdown

το backend είναι υπεύθυνο για τη δρομολόγηση, την εκτέλεση και το commit αυτών των uops.

Αυτό που μας ενδιαφέρει για να πετύχουμε υψηλές επιδόσεις στον επεξεργαστή είναι να μην έχουμε καθυστερήσεις στο pipeline. Άρα εντοπίζοντας σε ποιο σημείο του pipeline έχουν συμβεί καθυστερήσεις (bottleneck) μπορεί να μας δώσει μία πολύ καλή οπτική του προβλήματος ώστε να βελτιστοποιήσουμε ότι πραγματικά έχει σημασία.

Έτσι για να μπορέσουμε να εντοπίσουμε τα bottlenecks χωρίζουμε τις καθυστερήσεις κύκλων του επεξεργαστή σε 4 κατηγορίες: frontend stalls, backend stalls, instructions retired και bad speculation όπως φαίνεται στο Σχ. 3.1. Οι 4 βασικές κατηγορίες χωρίζονται περαιτέρω σε υποκατηγορίες για ακριβέστερο προσδιορισμό των bottlenecks, όπως φαίνεται στο Σχ. 3.2 (Πηγή [21]).

Τα frontend stalls αφορούν stalls που συμβαίνουν στο κομμάτι frontend του pipeline του επεξεργαστή, με αποτέλεσμα το backend μέρος να υποτροφοδοτείται από εντολές. Μία υψηλή τιμή καθυστερήσεων σε αυτή την κατηγορία μπορεί να προέρχεται είτε από latency στο fetch των εντολών είτε από το bandwidth του fetch. Στην 1η κατηγορία μπορούμε να θεωρήσουμε ότι ανήκουν καθυστερήσεις που προέρχονται από την I-cache, την Itlb cache, καθώς και από flushes εξαιτίας λανθασμένης πρόβλεψης του branch predictor. Ενώ στη δεύτερη ανήκουν προβλήματα στο bandwidth που αφορούν τις



Σχήμα 3.2: Topdown Hierarchy

μονάδες που τοποθετούν εντολές στην ουρά έτοιμων uops, για παράδειγμα οι instruction decoders.

Bad Speculation Stalls αναφέρονται σε καθυστερήσεις που κυρίως συμβαίνουν εξαιτίας λανθασμένων προβλέψεων του branch predictor, όπου τα εκτελεσμένα uops πρέπει να πεταχθούν, είτε uops καθυστερούν την εκτέλεση τους επειδή το pipeline βρίσκεται σε κατάσταση ανάκαμψης από λανθασμένη πρόβλεψη.

Retiring αυτή η κατηγορία αναφέρεται σε uops που γίνονται retire, ιδανικά εδώ θέλουμε όσο περισσότερα uops retired γίνεται.

Τέλος έχουμε τα Backend Stalls τα οποία συμβαίνουν όταν δεν μπορούν να έρθουν εντολές στο backend μέρος του pipeline εξαιτίας έλλειψης πόρων για να τις υποδεχτούν. Οι καθυστερήσεις αυτού του τύπου διακρίνονται σε δύο υποκατηγορίες: μνήμης και πυρήνα. Οι καθυστερήσεις μνήμης αφορούν καθυστερήσεις που συμβαίνουν στο υποσύστημα μνήμης ενώ οι καθυστερήσεις πυρήνα αφορούν κίνηση στις μονάδες εκτέλεσης ή μη βέλτιστη χρήση αυτών.

Για την μέτρηση των διάφορων καθυστερήσεων του pipeline του επεξεργαστή αλλά και

για άλλες μετρικές οι επεξεργαστές έχουν ειδικού σκοπού καταχωρητές (registers) που μπορούν να χρησιμοποιηθούν για να μετρήσουμε hardware events, όπως εντολές που έχουν εκτελεστεί, cache misses και πολλά άλλα.

Η μέτρηση των 4 κατηγοριών που αναφέρθηκαν προηγουμένως μπορεί να γίνει μετρώντας τα events που φαίνονται στον Πίνακα 3.1 με βάση τις παρακάτων εξισώσεις:

• Front End Bound = IDQ UOPS NOT DELIVERED.CORE / (4 * Clockticks)

• Bad Speculation = (UOPS ISSUED.ANY - UOPS RETIRED.RETIRE SLOTS + 4* INT MISC.RECOVERY CYCLES) /(4* Clockticks)

• Retiring = UOPS RETIRED.RETIRE SLOTS / (4 * Clockticks)

• Back End Bound = 1 – (FrontEnd Bound + Bad Speculation + Retiring)

Ένα pipeline slot αντιπροσωπεύει τους πόρους υλικού που απαιτούνται για την επεξεργασία ενός uOp. Ο χαρακτηρισμός Top-Down υποθέτει ότι για κάθε πυρήνα, σε κάθε κύκλο ρολογιού, υπάρχουν αρκετές διαθέσιμες υποδοχές αγωγού. Αυτός ο αριθμός ονομάζεται πλάτος αγωγού (Πηγή [21]).

IDQ_UOPS_NOT_DELIVERED.CORE	Αυτό το event μετράει τον αριθμό των uops που δεν έφτασαν στο backend per cycle όταν αυτό δεν είχε καθυστερήσεις.
UOPS_ISSUED.ANY	Αυτό το event μετράει τον αριθμό των uops που στάλθηκαν από το frontend στο backend
UOPS_RETIRED.RETIRE_SLOTS	Αυτό το event μετράει τον αριθμό των χρησιμοποιούμενων retirement slots σε κάθε κύκλο.
INT_MISC.RECOVERY_CYCLES	Αυτό το event μετράει τον αριθμό των μη χρησιμοποιούμενων issue slots εξαιτίας λανθασμένων προβλέψεων.

Πίνακας 3.1: Hardware events για μέτρηση κύκλων (Πηγή [13])

Σύμφωνα με την Intel ο προσδιορισμός του κατά πόσον μία κατηγορία αποτελεί ση-

Category	Expected Range of Pipeline Slots in This Category, for a Hotspot in a Well-Tuned:		
	Client/Desktop Application	Server/Database/Distributed application	High Performance Computing (HPC) application
Retiring	20-50%	10-30%	30-70%
Back-End Bound	20-40%	20-60%	20-40%
Front-End Bound	5-10%	10-25%	5-10%
Bad Speculation	5-10%	5-10%	1-5%

Σχήμα 3.3: Topdown hotspot

μείο συμφόρησης μπορεί να εξαρτάται από το φόρτο εργασίας, αλλά ορισμένες γενικές κατευθυντήριες γραμμές παρέχονται στο Σχ. 3.3 (Πηγή [10]).

3.3 Perf tool

Το perf tool (Πηγή [15]) είναι ένα εργαλείο των linux που μας επιτρέπει να έχουμε πρόσβαση στους μετρητές απόδοσης του επεξεργαστή. Βέβαια η χρήση του δεν είναι μόνο αυτή καθώς αποτελεί ένα πολύ ισχυρό εργαλείο που μπορεί να χρησιμοποιηθεί για πλήρη ανάλυση της απόδοσης του κώδικα, να βρεθούν τα σημεία που προκαλούν bottlenecks και είναι τα πιο δαπανηρά.

Είναι ένα ισχυρό εργαλείο ανάλυσης επιδόσεων και δημιουργίας προφίλ που χρησιμοποιεί τους μετρητές επιδόσεων που είναι διαθέσιμοι στις σύγχρονες CPU. Αξιοποιώντας τους μετρητές επιδόσεων, το εργαλείο Perf επιτρέπει στους προγραμματιστές και τους διαχειριστές συστημάτων να μετρούν και να βελτιστοποιούν την απόδοση των εφαρμογών τους.

Οι μετρητές επιδόσεων είναι εξειδικευμένοι καταχωρητές που παρακολουθούν και καταγράφουν συμβάντα που σχετίζονται με μετρήσεις επιδόσεων. Αυτές οι μετρικές μπορεί να περιλαμβάνουν εκτελεσμένες εντολές, χτυπήματα και αστοχίες στη μνήμη cache, προβλέψεις διακλαδώσεων και πολλές άλλες σχετικές παραμέτρους. Το εργαλείο Linux Perf αλληλεπιδρά με αυτούς τους μετρητές επιδόσεων για να συλλέγει δεδομένα επιδόσεων χαμηλού επιπέδου κατά τη διάρκεια της εκτέλεσης εφαρμογών.

Οι χρήστες του εργαλείου δύνατε να διαμορφώνουν και να επιλέγουν συγκεκριμένα συμβάντα απόδοσης που τους ενδιαφέρουν, είτε καθορίζοντας συμβάντα χρησιμοποιώντας προκαθορισμένα ονόματα ή να τα προσαρμόσουν με βάση συγκεκριμένα συμβάντα της CPU και μετρητές επιδόσεων που είναι διαθέσιμοι στο σύστημα, το οποίο όμως και απαιτεί μεγαλύτερη προσπάθεια και γνώση καθώς τα events αυτά αλλάζουν από επεξεργαστή σε επεξεργαστή και η ανάγνωση τους πρέπει να γίνεται με ιδιαίτερη προσοχή.

Το εργαλείο Perf υποστηρίζει την παρακολούθηση σε όλο το σύστημα, επιτρέποντας στους χρήστες να καταγράφουν δεδομένα απόδοσης σε όλες τις διεργασίες και τα νήματα του συστήματος. Αυτή η λειτουργία είναι ιδιαίτερα χρήσιμη για την ανάλυση της συνολικής συμπεριφοράς του συστήματος και τον εντοπισμό προβλημάτων απόδοσης σε επίπεδο συστήματος.

Με το πλούσιο σύνολο χαρακτηριστικών του, το εργαλείο Perf διευκολύνει τις δρα-

στηριότητες ρύθμισης επιδόσεων και δημιουργίας προφίλ. Βοηθά στην κατανόηση του τρόπου με τον οποίο διάφορα στοιχεία λογισμικού, αλγόριθμοι ή διαμορφώσεις επηρεάζουν την απόδοση τηςCPU. Συγκρίνοντας διαφορετικά σενάρια και διαμορφώσεις, οι προγραμματιστές μπορούν να λαμβάνουν τεκμηριωμένες αποφάσεις για τη βελτιστοποίηση των εφαρμογών τους για μέγιστη απόδοση.

Έχει κυρίως δύο τρόπους για μέτρηση, ο ένας είναι ο counting και ο δεύτερος είναι ο sampling.

Η λειτουργία καταμέτρησης στο Perf επιτρέπει στους προγραμματιστές και τους διαχειριστές συστημάτων να μετρήσουν με ακρίβεια την εμφάνιση συγκεκριμένων γεγονότων που τους ενδιαφέρουν. Επιτρέπει τη συλλογή μετρήσεων συμβάντων, όπως οι κύκλοι CPU, οι αστοχίες στη μνήμη ςαςηε ή οι εντολές που αποσύρονται. Αυτή η λειτουργία είναι ιδιαίτερα χρήσιμη όταν εστιάζετε σε συγκεκριμένες μετρήσεις επιδόσεων ή όταν προσπαθείτε να προσδιορίσετε την ακριβή συχνότητα συγκεκριμένων συμβάντων. Χρησιμοποιώντας τη λειτουργία καταμέτρησης, οι χρήστες μπορούν να εντοπίσουν τα σημεία συμφόρησης της απόδοσης και να κατανοήσουν βαθύτερα τη συμπεριφορά των εφαρμογών τους.

Η λειτουργία δειγματοληψίας, από την άλλη πλευρά, υιοθετεί μια στατιστική προσέγγιση στην παρακολούθηση των επιδόσεων. Αντί να καταγράφει κάθε συμβάν, λαμβάνει περιοδικά δείγματα της κατάστασης της εφαρμογής σε προκαθορισμένα χρονικά διαστήματα. Κάθε δείγμα παρέχει ένα στιγμιότυπο των χαρακτηριστικών απόδοσης κατά τη διάρκεια της συγκεκριμένης στιγμής, προσφέροντας μια στατιστική αναπαράσταση της συμπεριφοράς της εφαρμογής. Η λειτουργία δειγματοληψίας επιτρέπει στους χρήστες να αναλύουν τις συνολικές τάσεις απόδοσης, να εντοπίζουν τα hotspots και να αποκτούν πληροφορίες σχετικά με τα γενικά πρότυπα και την κατανομή της χρήσης των πόρων εντός της εφαρμογής. Με τη συλλογή δειγμάτων με καθορισμένο ρυθμό, οι προγραμματιστές μπορούν να αποκτήσουν μια αντιπροσωπευτική εικόνα της συμπεριφοράς της εφαρμογής χωρίς την ανάγκη συνεχούς παρακολούθησης.

Όταν γίνεται χρήση της λειτουργίας παρακολούθησης του Perf, είναι σημαντικό να εξετάζεται ο συμβιβασμός μεταξύ ακρίβειας και επιβάρυνσης. Η λειτουργία καταμέτρησης, με τις ακριβείς μετρήσεις συμβάντων, συνήθως συνεπάγεται μικρότερη επιβάρυνση, καθώς εστιάζει μόνο σε συγκεκριμένα συμβάντα που παρουσιάζουν ενδιαφέρον. Η λειτουργία δειγματοληψίας, από την άλλη πλευρά, εισάγει κάποια επιβάρυνση λόγω των

περιοδικών διακοπών που απαιτούνται για τη συλλογή δεδομένων. Ο ρυθμός δειγματοληψίας παίζει καθοριστικό ρόλο στην εξισορρόπηση του επιπέδου ακρίβειας και των επιπτώσεων στην απόδοση της εφαρμογής. Η επιλογή του κατάλληλου ρυθμού δειγματοληψίας εξασφαλίζει ότι τα δείγματα που συλλέγονται είναι αντιπροσωπευτικά της συμπεριφοράς της εφαρμογής, ελαχιστοποιώντας παράλληλα τις επιπτώσεις στην απόδοση.

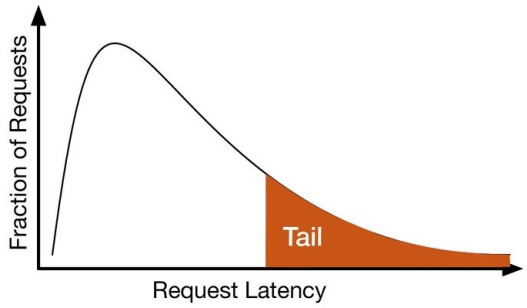
3.4 Latency / Tail latency

Η χρονική απόκριση (latency) αποτελεί βασικό υπαίτιο για προβλήματα επίδοσης και είναι ένας από τους κύριους λόγους χαμηλής ποιότητας εμπειρίας των χρηστών μίας εφαρμογη. Ως χρονική απόκριση θεωρούμε τον χρόνο που μεσολαβεί από την αποστολή ενός αιτήματος μέχρι τη λήψη απάντησης για το συγκεκριμένο αίτημα, για παράδειγμα ο χρόνος που παίρνει για να μας εμφανιστούν τα αποτελέσματα μίας αναζήτησης στο διαδίκτυο.

Η χρονική απόκριση ουράς (tail latency) αποτελεί τα υψηλά εκατοστημμόρια της χρονικής απόκρισης, δηλαδή τις χειρότερες περιπτώσεις και παρότι συμβαίνει σε μικρή συχνότητα αποτελεί μία σημαντική μετρική, καθώς μπορεί να βλάψει σημαντικά την απόδοση των εφαρμογών εισαγάγωντας χρόνους ανάκαμψης της εφαρμογής. Η μετρική αυτή έχει αποκτήσει ιδιαίτερη σημασία και δυσκολία στην μέτρηση και βελτιστοποίηση ειδικά σε συστήματα με μεγάλο μέγεθος και πολυπλοκότητα. Προφανώς σε συστήματα με λίγα αιτήματα το δευτερόλεπτο το να υπάρχουν καθυστερήσεις σε λίγους χρήστες δεν αποτελεί μεγάλο πρόβλημα αλλά όταν εξετάζουμε συστήματα που δέχονται πόλλα αίτηματα το δευτερόλεπτο το μέγεθος των χρηστών που θα έχουν καθυστερήσεις στο να λάβουν απάντηση αυξάνεται σημαντικά και δεν μπορεί να αγνοηθεί.

Στις εφαρμογές μικρουπηρεσιών, οι πολλαπλές κλήσεις υπηρεσιών και τα άλματα δικτύου συμβάλλουν στη συνολική καθυστέρηση. Οι βασικές πτυχές της καθυστέρησης περιλαμβάνουν:

1. Εμπειρία χρήστη: Η χαμηλή καθυστέρηση είναι απαραίτητη για την παροχή μιας ευέλικτης εμπειρίας χρήστη. Οι χρήστες αναμένουν γρήγορους χρόνους απόκρισης και οι



Σχήμα 3.4: Tail Latency

υψηλές καθυστερήσεις μπορεί να οδηγήσουν σε απογοήτευση και μειωμένη ικανοποίηση των χρηστών. Η καθυστέρηση επηρεάζει άμεσα τη χρηστικότητα της εφαρμογής και τη δέσμευση των χρηστών.

2. Αλληλεπιδράσεις υπηρεσιών: Στις αρχιτεκτονικές μικρουπηρεσιών, οι υπηρεσίες συχνά επικοινωνούν μεταξύ τους για την εκπλήρωση των αιτημάτων των πελατών. Κάθε κλήση υπηρεσίας εισάγει πρόσθετη καθυστέρηση και το σωρευτικό αποτέλεσμα των πολλαπλών αλληλεπιδράσεων μπορεί να επηρεάσει σημαντικά τον συνολικό χρόνο απόκρισης της εφαρμογής.

3. Επεκτασιμότητα: Η καθυστέρηση μπορεί να επηρεάσει την επεκτασιμότητα των εφαρμογών μικρουπηρεσιών. Οι υψηλές καθυστερήσεις μπορούν να περιορίσουν την απόδοση και τη χωρητικότητα του συστήματος, οδηγώντας σε υποβάθμιση των επιδόσεων και σε αδυναμία αποτελεσματικής διαχείρισης του αυξανόμενου φορτίου.

4. Σταθερότητα του συστήματος: Η καθυστέρηση ουράς μπορεί να επηρεάσει τη σταθερότητα και την αξιοπιστία των εφαρμογών μικρουπηρεσιών. Οι καθυστερήσεις μεγάλης ουράς μπορεί να προκαλέσουν αλυσιδωτές αποτυχίες και εξάντληση των πόρων σε μεταγενέστερες υπηρεσίες, οδηγώντας σε υποβάθμιση της απόδοσης και διακοπή των υπηρεσιών.

3.5 DeathStarBench, microservices applications

Η σουίτα DeathStarBench (Πηγή [7]) παρέχει τρεις εφαρμογές σχεδιασμένες με αρχιτεκτονική μικροϋπηρεσίων. Όλες οι εφαρμογές είναι σχεδιασμένες με κάποιες από τις βασικές αρχές των μικροϋπηρεσιών και προσπαθούν να εκμεταλλευτούν τα πλεονεκτήμα-

τα της αρχιτεκτονικής. Συγκεκριμένα είναι βασισμένο σε γνωστές εφαρμογές ανοιχτού κώδικα, όπως Nginx, memcached, mongodb, Mysql και πολλά άλλα, οπότε ο νέος κώδικας που γράφτηκε αφορούσε κυριώς επικοινωνία μεταξύ των αυτών των υπηρεσιών, με χρήση πρωτοκόλλων όπως gRPC, http, Apache Thrift. Έχουν χρησιμοποιηθεί διαφορετικές γλώσσες προγραμματισμού σε διάφορα σημεία ώστε να υπάρχει ετερογένεια και η απομόνωση τον διάφορων μικροϋπηρεσιών γίνεται με τα docker containers. Και τέλος υπάρχει ευκολία στην προσαρμογή και αλλαγή των διαφορετικών υπηρεσιών, το οποίο αποτελεί και ένα από τα βασικά πλεονεκτήματα των μικροϋπηρεσιων. Οι τρεις εφαρμογές είναι

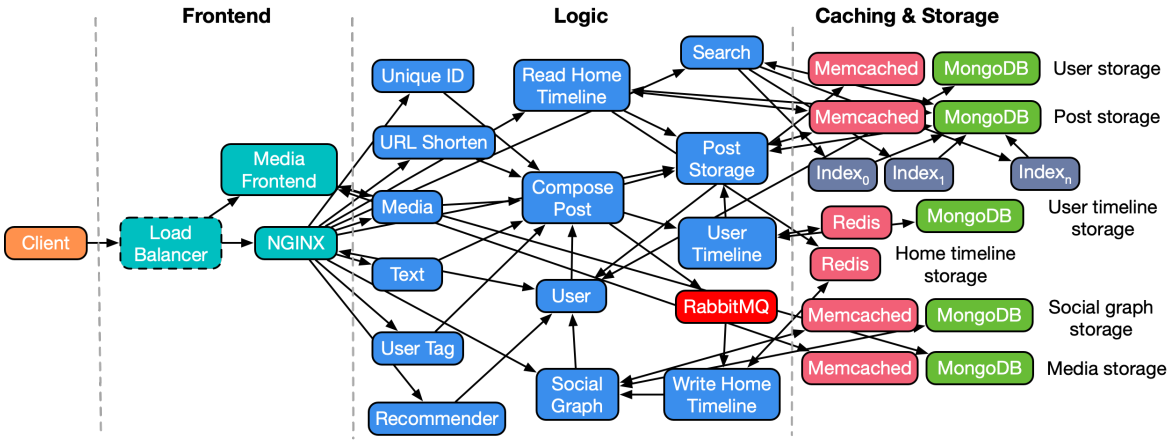
1. social network
2. mediamicroservices
3. hotelreservation

3.5.1 Social Network

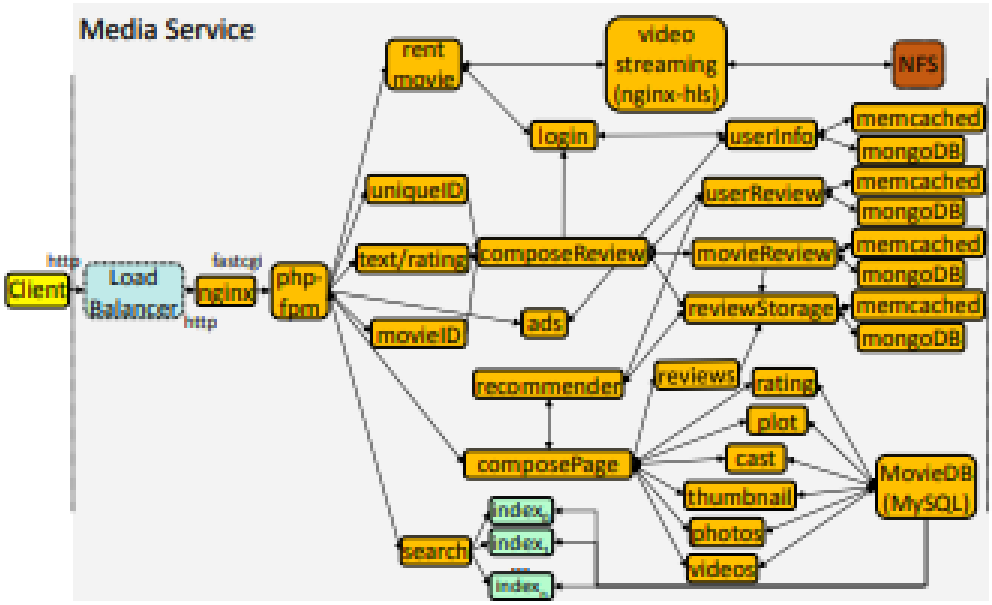
Η εφαρμογή αυτή αποτελεί ένα κοινωνικό δίκτυο με αμφίδρομες σχέσεις μεταξύ των χρηστών. Οι χρήστες τις εφαρμογής έχουν τη δυνατότητα να συνθέσουν αναρτήσεις με κείμενο, συνδέσμους, πολυμέσα και ετικέτες σε άλλους χρήστες. Μπορούν να διαβάσουν και να αλληλεπιδράσουν με αναρτήσεις χρηστών που ακολουθούν και ακόμα να δουν τις αναρτήσεις κάποιου συγκεκριμένου χρήστη. Το αίτημα των χρηστών μεταφέρεται μέσω http σε έναν ισορροπιστή φόρτου (load balancer) και αυτός με τη σειρά του το προωθεί στην υπεύθυνη υπηρεσία για την εξυπηρέτηση του αιτήματος και η τελική απάντηση στο αίτημα συνθέτεται μέσω μίας σειράς από εξαρτημένες υπηρεσίες που κάθε μία επιτελεί το σκοπό της και προωθεί το μήνυμα στην επόμενη υπηρεσία, φτιάχοντας ένα δίκτο εξαρτήσεων μεταξύ των μικροϋπηρεσιών όπως φαίνεται στο σχήμα 3.5. Η εφαρμογή διαθέτει memchached (Πηγή [17]) για caching και mongodb (Πηγή [5]) για να κρατάει τα δεδομένα.

3.5.2 Media microservices

Η συγκεκριμένη εφαρμογή αποτελεί υπηρεσία για περιήγηση πληροφοριών ταινιών καθώς και αξιολόγηση και κριτική. Συγκεκριμέμα οι χρήστες μπορούν να δουν πλοκή, φωτογραφίες, βίντεο, συντελεστές και κριτικές για κάποια συγκεκριμένη ταινία. Ακόμα συνδεόμενοι μπορούν να προσθέσουν νέες αξιολογησείς και κριτικές.

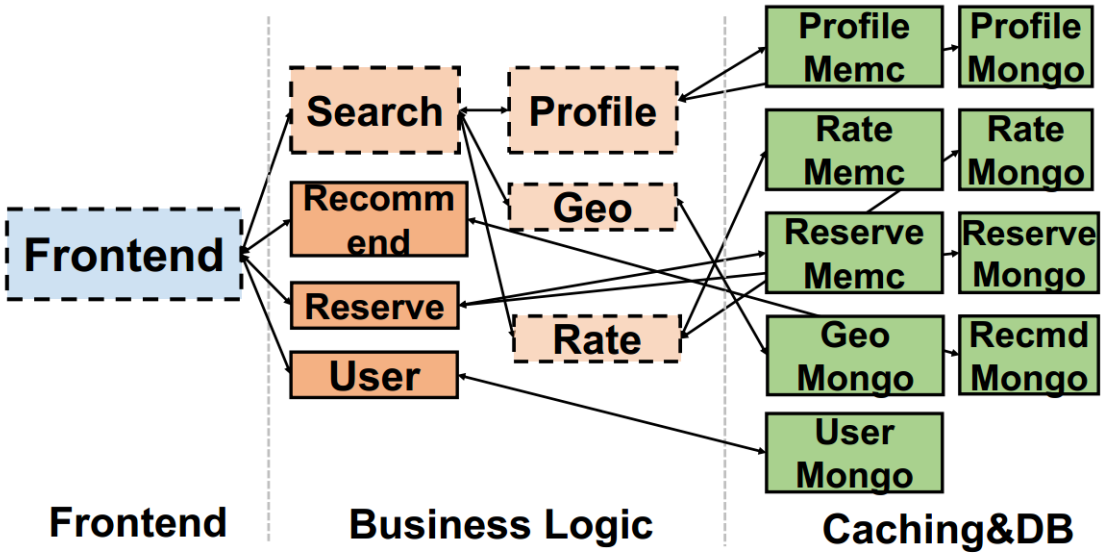


Σχήμα 3.5: Social network



Σχήμα 3.6: Media Microservices

Η προώθηση του αιτήματος είναι παρόμοια με το social network, το αίτημα θα βρεθεί πρώτα στον load balancer και από εκεί θα προωθηθεί σε κάποιον nginx webserver και στη συνέχεια στην υπεύθυνη για το αίτημα μικροϋπηρεσία, όπως φαίνεται στο σχήμα 3.6.



Σχήμα 3.7: Hotel reservation

3.5.3 Hotel reservation

Η εφαρμογή hotel reservation αποτελεί μία εφαρμογή για την έρευση ξενοδοχείων και τοποθέτηση κράτησης σε αυτά. Συγκεκριμένα οι χρήστες μπορούν να συνδεθούν με τα στοιχεία τους, να ψάξουν διαθέσιμα ξενοδοχεία με βάση κριτήρια όπως ημερομηνία, τοποθεσία, να λάβουν προτάσεις καθώς και να δουν και να τοποθετήσουν κριτικές και αξιολογησείς σε κάποιο συγκεκριμένο ξενοδοχείο. Η επικοινωνία μεταξύ των υπηρεσιών της εφαρμογής γίνεται με gRPC και ο γράφος εξαρτήσεων των υπηρεσιών φαίνεται στο σχήμα 3.7.

3.6 Load Genarator

Οι γενήτριες φόρτου είναι συστήματα που χρησιμοποιούνται για να προσομοιώσουν φόρτο ώστε να γίνουν τεστς απόδοσης σε εφαρμογές ή σε κομμάτια αυτών. Άρα η λειτουργία του συστήματος αυτού είναι να στέλνει αιτήματα με συγκεκριμένο ρυθμό στο σύστημα που θέλουμε να ελέγξουμε την ανταπόκριση του σε διάφορα σενάρια φόρτου.

Το wrk2 (Πηγή [20]) αποτελεί μία τετοια γενήτρια φόρτου που στέλνει http αιτήματα.

Είναι βασισμένο στο wrk, πάνω στο οποίο προθέτει κάποιες λειτουργικότητες. Το wrk2 παράγει συνεχές φόρτο με συγκεκρίμενο ρυθμό και παρέχει ακριβές πληροφορίες για τον χρόνο απόκρισης των αιτημάτων και σε υψηλά εκατοστημμόρια, δίνοντας μας μία καλή οπτική για τον χρόνο απόκρισης ουράς. Ακόμα μας επιτρέπει να ορίσουμε με παράμετρο την κατανομή που θέλουμε να έχει η αποστολή των http αιτημάτων. Ένα παράδειγμα χρήσης του φαίνεται παρακάτω:

wrk -t2 -c100 -d30s -R2000 http://127.0.0.1:8080/index.html

Όπου μπορούμε να δούμε ότι ορίζονται τα threads που θα χρησιμοποιηθούν πόσα connections να οριστούν πόση ώρα θα τρέξει ο load generator πόσα αιτήματα το δευτερόλεπτο θα τρέχει και σε ποια διεύθυνση θα τα στέλνει, με τις παραμέτρους t, c, d, R αντίστοιχα.

3.7 Stress testing, IBench suite

Η αξιοποίηση των διακομιστών στα κέντρα δεδομένων σήμερα είναι πασίγνωστα χαμηλή για λόγους που περιλαμβάνουν αλλαγές φορτίου, δυσκολία στην κατάλληλη παροχή διακομιστών, ευρέως ποικίλα χαρακτηριστικά και περιορισμοί του φόρτου εργασίας, και η πλατφόρμα ετερογένεια, το πιο σημαντικό είναι ότι η αύξηση της χρήσης περιλαμβάνει τον συν-προγραμματισμό εφαρμογών στην ίδια μηχανή, η οποία έχει ως αποτέλεσμα την υποβάθμιση της απόδοσης λόγω παρεμβολών στους κοινόχρηστους πόρους.

Επομένως, η κατανόηση της ευαισθησίας των φόρτων εργασίας στον ανταγωνισμό (Πηγή [2]) είναι κρίσιμη για τη μείωση της και τη διαχείριση της παρεμβολής με τρόπο που να επιτρέπει την επίγνωση QoS λειτουργία σε υψηλή χρήση. Για την κατανόηση της ευαισθησίας λοιπόν θα χρησιμοποιηθούν τα τεστ πίεσης.

Τεστ πίεσης (stress testing) είναι η διαδικασία κατά την οποία φαίνεται η δυνατότητα της εφαρμογής (κατ΄ επέκταση κάποιου συστήματος γενικώς) να διατηρεί ένα επιθυμητό επίπεδο αποτελεσματικότητας κάτω από δυσχερείς συνθήκες. Η διαδικασία συνήθως περιλαμβάνει την καταγραφή της συχνότητας παραγωγής λαθών και ανάνηψης του συστήματος από κατάρρευση.

Το εργαλείο που χρησιμοποιήθηκε στην παρούσα διπλωματική είναι το iBench (Πηγή

[3]). Το iBench αποτελείται από μία σειρά από benchmarks που χρησιμοποιούνται για να ποσοτικοποιήσουν την επίπτωση που έχουν διάφορες εφαμοργές σε μία ποικιλία από μοιραζόμενους πόρους αλλά και την πίεση που μπορούν να αντέξουν οι εφαρμογές μέσα σε ένα συγκεκριμένο σύστημα πόρων. Το iBench περιέχει benchmarks που δημιουργούν πίεση στον επεξεργαστή, στην ιεραρχία της κρυφής μνήμης καθώς και στη μνήμη.

Η σουίτα περιλαμβάνει τα παρακάτω στρές τεστς.

Χωρητικότητα μνήμης: Αυτό το τεστ προσπελαύνει προοδευτικά μεγαλύτερα τμήματα μνήμης μέχρι να καταλάβει ολόκληρη τη μνήμη χωρητικότητα της μνήμης. Το μοτίβο προσπέλασης των διευθύνσεων σε αυτή την περίπτωση είναι τυχαίο. Εκκινεί όσες αιτήσεις είναι απαραίτητες για να εγγυηθεί την κατάλληλη κάλυψη της χωρητικότητας σε κάθε σημείο κατά τη διάρκεια της εκτέλεσης.

Εύρος ζώνης μνήμης: Το σημείο αναφοράς σε αυτή την περίπτωση πραγματοποιεί σειριακές προσπελάσεις μνήμης αυξανόμενου σε ένα μικρό τμήμα του χώρου διευθύνσεων. Η ένταση αυξάνεται έως ότου καταναλώθει το 100% της συνεχιζόμενης εύρους ζώνης μνήμης της συγκεκριμένης μηχανής. Η ένταση των προσπελάσεων αυξάνεται γραμμικά με το εύρος ζώνης μνήμης που χρησιμοποιείται. Ο λόγος για τον οποίο οι προσπελάσεις συμβαίνουν σε ένα σχετικά μικρό κλάσμα της μνήμης (π.χ. 10%) είναι για να αποσυνδεθούν οι επιδράσεις των ανταγωνισμού στο εύρος ζώνης της μνήμης από τον ανταγωνισμό στη μνήμη χωρητικότητας.

Χωρητικότητα κρυφής μνήμης τελευταίου επιπέδου (LLC): Το benchmark διαβάζει το /proc/cpuinfo του συστήματος και ρυθμίζει το footprint του, το μοτίβο προσπέλασης και τον ρυθμό με τον οποίο αυξάνεται η έντασή της με βάση το μέγεθος και τη συσχετιστικότητα της συγκεκριμένης LLC. Το benchmark κάνει τυχαίες προσπελάσεις που καλύπτουν ένα αυξανόμενο μέγεθος της χωρητικότητας της LLC.

L1 i-cache: Ένας απλό πρόγραμμα που σαρώνει αυξανόμενα κλάσματα της i-cache, μέχρι να γεμίσει πλήρως την i-cache. χωρητικότητα. Οι προσπελάσεις σε αυτή την περίπτωση είναι και πάλι τυχαίες.

L1 d-cache: Ένα αντίγραφο του προηγούμενου προγράμματος, προσαρμοσμένο στο δομή και το μέγεθος της d-cache (συνήθως το ίδιο με την i-cache).

L2 capacity: Πρόκειται για παρόμοιο benchmark με αυτό της χωρητικότητας LLC. Το

αποτύπωμα σε αυτή την περίπτωση αυξάνεται μέχρι το μέγεθος της κρυφής μνήμης L2 και η συσχετιστικότητα L2 χρησιμοποιείται για τη ρύθμιση του τρόπου με τον οποίο η ένταση μεταβάλλεται κατά τη διάρκεια του εκτέλεσης του προγράμματος.

Τέλος χρησιμοποιήθηκε για stress των ports της cpu το gemm (Πηγή: [18]). Όπου και η πίεση γίνεται με matrix multiplication.

3.8 Δυσκολία μέτρησης απόδοσης

Οι μετρικές μικροαρχιτεκτονικής διαδραματίζουν κρίσιμο ρόλο στην αξιολόγηση της απόδοσης και της αποδοτικότητας των συστημάτων υπολογιστών. Το IPC και οι αστοχίες μνήμης είναι δύο σημαντικές μετρικές που χρησιμοποιούνται για την αξιολόγηση της αποτελεσματικότητας του σχεδιασμού της μικροαρχιτεκτονικής. Σε περιβάλλοντα νέφους, όπου πολλαπλές εικονικές μηχανές (VM) ή containers εκτελούνται σε κοινόχρηστους φυσικούς πόρους, η ακριβής μέτρηση αυτών των μετρικών καθίσταται ένα πολύπλοκο έργο.

3.8.1 Εικονικοποίηση και διαμοιρασμός πόρων

Οι πάροχοι υπολογιστικού νέφους χρησιμοποιούν τεχνικές εικονικοποίησης για τη μεγιστοποίηση της χρήσης των πόρων και την ενεργοποίηση της πολλαπλής μίσθωσης. Ωστόσο, η εικονικοποίηση εισάγει παρεμβολές και διαμάχη μεταξύ συν-τοποθετημένων containers, καθιστώντας δύσκολη την απομόνωση και την ακριβή μέτρηση των μετρικών μικροαρχιτεκτονικής μεμονωμένων instances. Η δυναμική φύση των περιβαλλόντων νέφους περιπλέκει περαιτέρω τη διαδικασία μέτρησης.

3.8.2 Θορυβώδεις γείτονες και μεταβλητότητα επιδόσεων

Σε ένα περιβάλλον νέφους, τα containers μοιράζονται τους υποκείμενους πόρους υλικού, οδηγώντας στο πρόβλημα των ¨θορυβωδών γειτόνων¨. Όταν μια γειτονική περίπτωση παρουσιάζει υψηλή ζήτηση πόρων, μπορεί να επηρεάσει την απόδοση άλλων περιπτώσεων, με αποτέλεσμα ασυνεπή αποτελέσματα μέτρησης. Επιπλέον, η μεταβλητότητα των επιδόσεων σε περιβάλλοντα νέφους μπορεί να προκύψει από παράγοντες όπως οι κοινές κρυφές μνήμες αποθήκευσης και η επιβάρυνση του εικονικού δικτύου.

3.8.3 Περιορισμένη ορατότητα και έλεγχος

Οι χρήστες του νέφους έχουν συχνά περιορισμένη ορατότητα και έλεγχο του υποκείμενου υλικού. Η πρόσβαση σε μετρητές επιδόσεων χαμηλού επιπέδου και σε εργαλεία παρακολούθησης υλικού μπορεί να είναι περιορισμένη, εμποδίζοντας την ακριβή μέτρηση των μετρικών της μικροαρχιτεκτονικής. Αυτή η έλλειψη ελέγχου καθιστά δύσκολη τη συλλογή λεπτομερών δεδομένων που απαιτούνται για ολοκληρωμένη ανάλυση.

3.9 Προσεγγίσεις λύσεων των προκλήσεων μέτρησης

Για να ξεπεραστούν οι προκλήσεις στη μέτρηση των μετρικών μικροαρχιτεκτονικής σε περιβάλλοντα νέφους, μπορούν να υιοθετηθούν διάφορες προσεγγίσεις. Αυτές περιλαμβάνουν:

1. Στατιστική δειγματοληψία: Αξιοποίηση στατιστικών τεχνικών για τη δειγματοληψία ενός υποσυνόλου περιπτώσεων και τη μέτρηση των μετρικών μικροαρχιτεκτονικής. Με τον τρόπο αυτό μειώνεται η επιβάρυνση της μέτρησης, ενώ παρέχεται μια αντιπροσωπευτική εικόνα της συνολικής απόδοσης του συστήματος.

2. Οργανοποίηση και ανίχνευση: Χρήση ελαφρών μηχανισμών ενοργάνωσης ή ανίχνευσης εντός containers για τη συλλογή πληροφοριών που σχετίζονται με την απόδοση. Αυτή η προσέγγιση επιτρέπει καλύτερη ορατότητα και έλεγχο της διαδικασίας μέτρησης.

3. Εικονική επιθεώρηση μηχανών: Αξιοποίηση τεχνικών ενδοσκόπησης για την εξαγωγή πληροφοριών. Αυτό επιτρέπει τη μέτρηση των μετρικών μικροαρχιτεκτονικής χωρίς να απαιτούνται τροποποιήσεις σε φιλοξενούμενα container.

4. Παρακολούθηση με τη βοήθεια υλικού: Αξιοποίηση χαρακτηριστικών παρακολούθησης που βασίζονται σε υλικό, εφόσον είναι διαθέσιμα, για την ακριβή μέτρηση των μετρικών της μικροαρχιτεκτονικής.

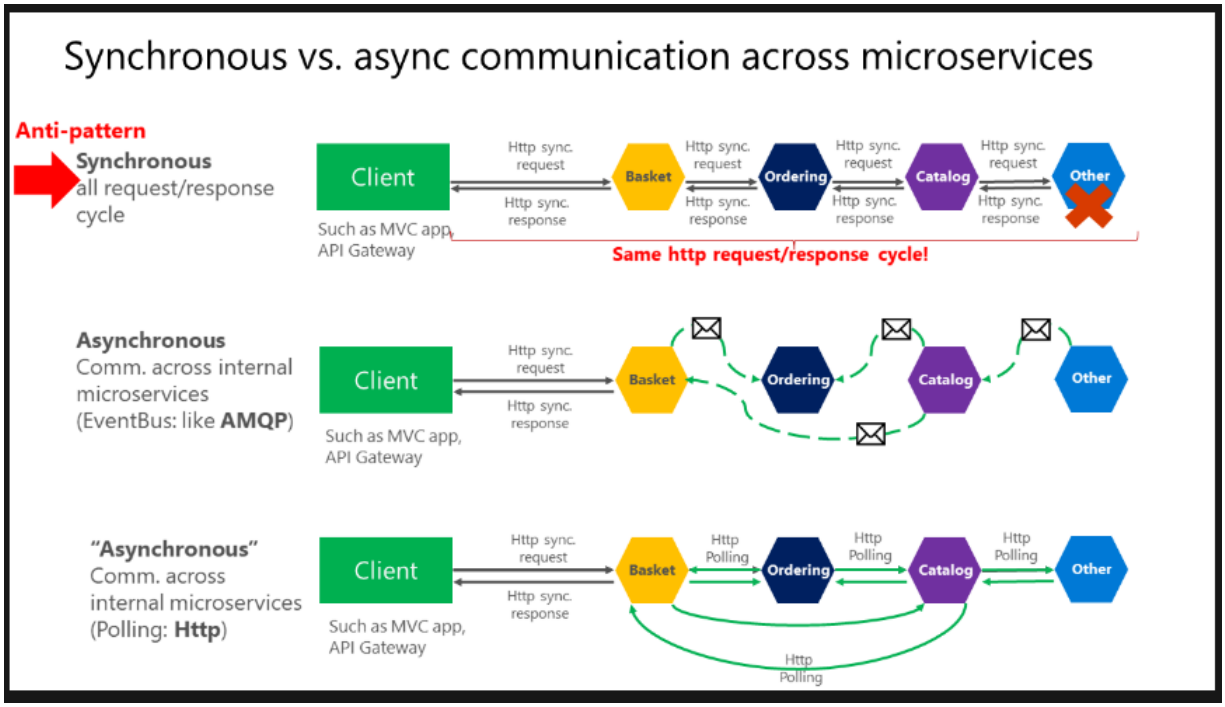
Η μέτρηση των μετρικών μικροαρχιτεκτονικής, όπως το IPC και οι αστοχίες μνήμης σε περιβάλλοντα νέφους, είναι ένα δύσκολο έργο λόγω της εικονικοποίησης, του διαμοιρασμού πόρων, της μεταβλητότητας των επιδόσεων και της περιορισμένης ορατότητας. Ωστόσο, με την υιοθέτηση κατάλληλων προσεγγίσεων λύσεων, είναι δυνατόν να αμβλυνθούν αυτές οι προκλήσεις και να αποκτηθούν ουσιαστικές γνώσεις σχετικά με την

απόδοση της μικροαρχιτεκτονικής στο νέφος.

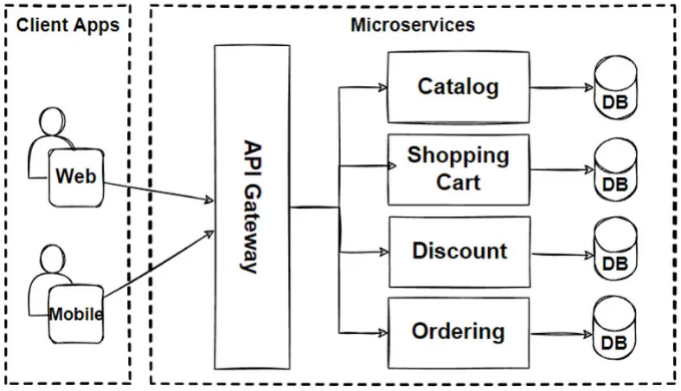
3.10 Microservices architecture communication

Η επικοινωνία μεταξύ των microservices φαίνεται στο Σχ. 3.8. Βασικά υπάρχουν 2 κύριοι τρόποι ο ένας είναι μέσα από βασισμένη σε μηνύματα επικοινωνία όπου και συνήθως υπάρχει κάποια background διαδικασία αποστολής μηνυμάτων για ενημέρωση του παραλήπτη. Ο δεύτερος είναι η άμεση επικοινωνία μεταξύ των microservices με πρωτόκολλα όπως http, rpc όπου εδώ συνίσταται η επικοινωνία να γίνεται ασύγχρονα.

Οπως μπορούμε να δούμε στο Σχ. 3.9 ο σχεδιασμός η αρχιτεκτονική αποτελείται από ένα api gateway. Αυτό ανακατευθύνει τα αιτήματα στα σωστά microservices τα οποία έχουν τις δικές τους βάσεις δεδομένων και δεν τις μοιράζονται με άλλα microservices. Συγκεκριμένα η api gateway παρέχει ένα ενιαίο τελικό σημείο για τις εφαρμογές-πελάτες και αντιστοιχίζει εσωτερικά τα αιτήματα σε εσωτερικές μικρουπηρεσίες. Η api gateway εντοπίζει μεταξύ των εφαρμογών-πελατών και των εσωτερικών μικρουπηρεσιών. Λειτουργεί ως reverse proxy και δρομολογεί τα αιτήματα από τους πελάτες προς τις backend υπηρεσίες. Παρέχει επίσης λύσεις για οριζόντιες ανησυχίες όπως ο έλεγχος ταυτότητας, ο τερματισμός SSL και η κρυφή μνήμη.



Σχήμα 3.8: Microservive communication patterns



Σχήμα 3.9: Microservice architecture

Κεφάλαιο 4

Απόδοση μικροϋπηρεσιών σε σύγχρονα συστήματα

4.1 Εισαγωγή

Η παρούσα διπλωματική θα επικεντρωθεί στη σημασία του χαρακτηρισμού των επιδόσεων μικροϋπηρεσιών, δίνοντας έμφαση στην μικροαρχιτεκτόνικη απόδοση των συστημάτων που φιλεξενούνται. Θα διερευνηθεί η απόδοση σε μετρικές χαμηλού επιπέδου όπως εντολές ανά κύκλους εκτέλεσεις, αστοχίες στις μνήμες cache, ποσοστά κύκλων στα μέρη της σωλήνωσης του επεξεργαστή όπως αναλύθηκαν στο κεφάλαιο 3, καθώς και αν υπάρχει σύνδεση μεταξύ μετρικών χαμήλου επιπέδου (όπως IPC) με μετρικές που επηρεάζουν άμεσα την εμπειρία των χρηστών όπως η χρονική απόκριση καθώς και η χρονική απόκριση ουράς. Και τέλος θα γίνει προσπάθεια να βρεθεί ένα καλύτερο στατικό placement στους διαθέσιμους επεξεργαστές με βάση τα δεδομένα που συλλέχθηκαν.

4.2 Διαδικασία ανάλυσης απόδοσης εφαρμογών

Η διαδικασία με την οποία μπορεί να αναλυθεί η απόδοση μίας εφαρμογής χωρίζεται σε τρία βασικά επίπεδα, όπως αποτυπώνεται και στο Σχ 4.1.

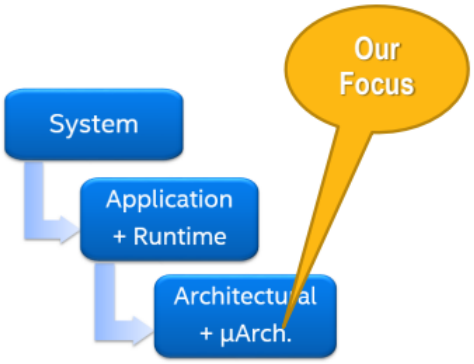
Η διαδικασία ανάλυσης της απόδοσης μιας εφαρμογής είναι ένα πολύπλευρο εγχείρημα που περιλαμβάνει τη διερεύνηση διαφόρων επιπέδων του υπολογιστικού συστήματος. Αυτή η ολιστική προσέγγιση μας επιτρέπει να αποκτήσουμε μια ολοκληρωμένη κατα-

νόηση των παραγόντων που επηρεάζουν την απόδοση της εφαρμογής και να εντοπίσουμε περιοχές για βελτιστοποίηση.

Στο πρώτο επίπεδο, η προσοχή εστιάζεται στο ίδιο το σύστημα. Αυτό περιλαμβάνει την αξιολόγηση κρίσιμων παραμέτρων του συστήματος, όπως το μέγεθος της μνήμης, οι δυνατότητες δικτύου και η ταχύτητα του επεξεργαστή. Ο στόχος είναι να διασφαλιστεί ότι το σύστημα διαθέτει τους απαραίτητους πόρους και τις δυνατότητες για να φιλοξενήσει την εφαρμογή χωρίς να καταστεί σημείο συμφόρησης των επιδόσεων. Αξιολογώντας τη χωρητικότητα του συστήματος και εξετάζοντας τη συμβατότητά του με τις απαιτήσεις της εφαρμογής, μπορούν να εντοπιστούν έγκαιρα πιθανά προβλήματα ή περιορισμοί. Αυτό το επίπεδο ανάλυσης αποσκοπεί στο να εγγυηθεί ότι το σύστημα μπορεί να χειριστεί αποτελεσματικά τις απαιτήσεις της εφαρμογής, επιτρέποντάς της να λειτουργεί βέλτιστα. Άλλοι παράμετροι του συστήματος που είναι υπό εξέταση είναι το μέγεθος της μνήμης, η ταχύτητα του δικτύου και του επεξεργαστή. Δηλαδή αν μπορεί το σύστημα να φιλοξενήσει την εφαρμογή χωρίς να αποτελεί αυτό bottleneck της απόδοσης.

Περνώντας στο δεύτερο επίπεδο, η προσοχή στρέφεται προς τη βελτιστοποίηση σε επίπεδο εφαρμογής. Εδώ, ο πρωταρχικός στόχος είναι η βελτίωση της απόδοσης της εφαρμογής με την αντιμετώπιση των πολυπλοκότητας στους αλγορίθμους που χρησιμοποιούνται ή με τη διερεύνηση ευκαιριών παραλληλισμού. Εδώ γίνεται εμβαθύνση στις περιπλοκές των αλγορίθμων της εφαρμογής, αναζητώντας τρόπους για τον εξορθολογισμό των διαδικασιών, τη μείωση των υπολογιστικών επιβαρύνσεων και τη βελτίωση της συνολικής απόδοσης. Επιπλέον, εάν η εφαρμογή προσφέρεται για παράλληλη επεξεργασία, μπορεί να καταβληθούν προσπάθειες για την αξιοποίηση των διαθέσιμων πόρων υλικού και την κατανομή του φόρτου εργασίας σε πολλαπλούς πυρήνες ή επεξεργαστές. Με τη βελτιστοποίηση των αλγορίθμων και τη διερεύνηση τεχνικών παραλληλισμού, μπορούν να αμβλυνθούν τα σημεία συμφόρησης των επιδόσεων εντός της εφαρμογής, με αποτέλεσμα τη βελτίωση της ταχύτητας και της απόκρισης. Δηλαδή η βελτιστοποίση στις πολυπλοκότητες των αλγορίθμων που χρησιμοποίουνται, ή η χρήση παραλληλοποίσης εφόσον είναι εφικτό ανα περίπτωση.

Το τρίτο και τελευταίο επίπεδο, στο οποίο επικεντρώνεται η παρούσα διπλωματική, εστιάζει στις αρχιτεκτονικές και μικροαρχιτεκτονικές πτυχές του υπολογιστικού συστήματος. Αυτό το επίπεδο εμβαθύνει στην εσωτερική σχεδίαση και οργάνωση της CPU και των συστατικών της, όπως οι καταχωρητές, οι σωληνώσεις, οι ιεραρχίες κρυφής



Σχήμα 4.1: Επίπεδα εξερεύνησης απόδοσης

μνήμης και οι μονάδες εκτέλεσης. Με την προσεκτική εξέταση αυτών των αρχιτεκτονικών στοιχείων, στοχεύεται ο εντοπισμός πιθανών βελτιστοποιήσεων που μπορούν να εφαρμοστούν για τη βελτίωση της απόδοσης της εφαρμογής. Αυτό το επίπεδο ανάλυσης περιλαμβάνει τη μελέτη των αλληλεπιδράσεων μεταξύ των διαφόρων στοιχείων της CPU, τον εντοπισμό πιθανών σημείων συμφόρησης ή αναποτελεσματικότητας και τη διερεύνηση αρχιτεκτονικών βελτιώσεων ή μικροαρχιτεκτονικών βελτιώσεων. Για παράδειγμα, αλλαγές στην ιεραρχία της κρυφής μνήμης για τη βελτίωση των προτύπων πρόσβασης στα δεδομένα ή βελτιστοποιήσεις του αγωγού για την ελαχιστοποίηση της καθυστέρησης εντολών. Με την κατανόηση των περίπλοκων λεπτομερειών του εσωτερικού σχεδιασμού της CPU, μπορούν να ληφθούν τεκμηριωμένες αποφάσεις σχετικά με αρχιτεκτονικές τροποποιήσεις που μπορούν να έχουν σημαντικό αντίκτυπο στη συνολική απόδοση της εφαρμογής.

Με την υιοθέτηση μιας πολυεπίπεδης προσέγγισης, είναι εφικτό να εντοπιστούν και να αντιμετωπιστουν οι περιορισμοί των επιδόσεων, βελτιώνοντας τελικά την αποδοτικότητα και την αποτελεσματικότητα της εφαρμογής.

4.3 Contention Interference

Καθώς η ζήτηση για υπολογιστική ισχύ και επεξεργασία δεδομένων συνεχίζει να αυξάνεται κατακόρυφα, τα κέντρα δεδομένων αντιμετωπίζουν σημαντικές προκλήσεις, μία από τις οποίες είναι ο ανταγωνισμός και οι παρεμβολές (Πηγή [2]). Το παρόν έγγραφο

διερευνά το πρόβλημα του ανταγωνισμού και της παρεμβολής, αναδεικνύοντας τις επιζήμιες επιπτώσεις του και τονίζοντας την κρίσιμη σημασία της εξεύρεσης λύσεων για την αντιμετώπιση αυτού του πιεστικού ζητήματος.

4.3.1 Κατανόηση του ανταγωνισμού και της παρεμβολής

Σε ένα κοινόχρηστο υπολογιστικό περιβάλλον, ο ανταγωνισμός και η παρεμβολή εμφανίζονται όταν πολλαπλές εφαρμογές ή διεργασίες ανταγωνίζονται για κοινόχρηστους πόρους, όπως η CPU, η μνήμη, η αποθήκευση και το εύρος ζώνης δικτύου. Καθώς αυξάνεται ο αριθμός των εφαρμογών και των χρηστών, αυξάνεται και ο ανταγωνισμός για αυτούς τους πόρους. Αυτός ο ανταγωνισμός οδηγεί σε παρεμβολές, όπου η απόδοση μιας εφαρμογής επηρεάζεται αρνητικά από τις δραστηριότητες άλλων εφαρμογών ή διεργασιών.

4.3.2 Επιπτώσεις του ανταγωνισμού και της παρεμβολής

Ο ανταγωνισμός και η παρεμβολή μπορούν να προκαλέσουν σημαντική υποβάθμιση της απόδοσης, οδηγώντας σε αυξημένους χρόνους απόκρισης, μειωμένη απόδοση και συνολικά μειωμένη εμπειρία χρήστη. Καθώς πολλαπλές εφαρμογές διεκδικούν πόρους, η επεξεργαστική ισχύς που είναι διαθέσιμη σε κάθε εφαρμογή μειώνεται, με αποτέλεσμα μη βέλτιστες επιδόσεις και βραδύτερους χρόνους εκτέλεσης.

Εισάγουν ακόμα ένα στοιχείο απρόβλεπτου στις λειτουργίες του κέντρου δεδομένων. Η απόδοση των εφαρμογών γίνεται εξαιρετικά μεταβλητή, καθιστώντας δύσκολη την τήρηση των συμφωνιών επιπέδου υπηρεσιών και την παροχή σταθερής απόδοσης στους τελικούς χρήστες. Αυτή η απρόβλεπτη κατάσταση δυσχεραίνει την ικανότητα αποτελεσματικής διαχείρισης των φόρτων εργασίας και αποδοτικής κατανομής των πόρων.

Ο ανταγωνισμός και οι παρεμβολές οδηγούν συχνά σε υποαπασχόληση των πόρων. Όταν οι πόροι δεν μοιράζονται ή δεν κατανέμονται αποτελεσματικά, ορισμένοι πόροι μπορεί να παραμένουν αδρανείς ή να υπολειτουργούν, ενώ άλλοι κατακλύζονται από υπερβολική ζήτηση. Αυτή η αναποτελεσματική χρήση των πόρων μπορεί να οδηγήσει σε περιττές δαπάνες και να εμποδίσει τα κέντρα δεδομένων να μεγιστοποιήσουν την υπολογιστική τους χωρητικότητα.

4.3.3 Σημασία της επίλυσης της διαμάχης και της παρεμβολής

Η επίλυση προβλημάτων ανταγωνισμού και παρεμβολών είναι ζωτικής σημασίας για τη διασφάλιση βέλτιστων επιδόσεων και εμπειρίας χρήστη. Με τον μετριασμό του ανταγωνισμού και την ελαχιστοποίηση των παρεμβολών, τα κέντρα δεδομένων μπορούν να παρέχουν σταθερή και προβλέψιμη απόδοση, ικανοποιώντας τις αυξανόμενες απαιτήσεις των χρηστών και των εφαρμογών. Η βελτιωμένη απόδοση μεταφράζεται σε μεγαλύτερη ικανοποίηση των πελατών, αυξημένη παραγωγικότητα και καλύτερη αξιοποίηση των διαθέσιμων πόρων.

Η αντιμετώπιση του προβλήματος οδηγεί σε καλύτερη αξιοποίηση των πόρων και βελτιστοποίηση του κόστους. Με την αποτελεσματική διαχείριση της κατανομής πόρων και την ελαχιστοποίηση του ανταγωνισμού, τα κέντρα δεδομένων μπορούν να αξιοποιήσουν στο έπακρο τις επενδύσεις τους σε υποδομές, αποφεύγοντας τη σπάταλη υπερπροσφορά, διατηρώντας παράλληλα υψηλά επίπεδα υπηρεσιών. Αυτή η βελτιστοποίηση συμβάλλει στη μείωση των λειτουργικών δαπανών και στη βελτίωση της συνολικής οικονομικής βιωσιμότητας των λειτουργιών του κέντρου δεδομένων.

Καθώς ο όγκος των δεδομένων και οι υπολογιστικές απαιτήσεις συνεχίζουν να κλιμακώνονται, η επεκτασιμότητα καθίσταται υψίστης σημασίας. Με την επίλυση του προβλήματος του ανταγωνισμού και των παρεμβολών, τα κέντρα δεδομένων μπορούν να κλιμακωθούν αποτελεσματικότερα, φιλοξενώντας έναν αυξανόμενο αριθμό εφαρμογών και χρηστών χωρίς συμβιβασμούς στην απόδοση. Αυτή η επεκτασιμότητα όχι μόνο εξασφαλίζει απρόσκοπτη λειτουργία, αλλά και διασφαλίζει τα κέντρα δεδομένων για το μέλλον, επιτρέποντάς τους να προσαρμόζονται στις εξελισσόμενες τεχνολογικές εξελίξεις και στις απαιτήσεις του κλάδου.

Ο ανταγωνισμός και οι παρεμβολές αποτελούν σημαντικές προκλήσεις για τα σύγχρονα κέντρα δεδομένων, παρεμποδίζοντας την απόδοση, την απρόβλεπτη λειτουργία και την αναποτελεσματική χρήση των πόρων. Η αναγνώριση της σημασίας της επίλυσης αυτών των ζητημάτων είναι ζωτικής σημασίας για τους φορείς εκμετάλλευσης κέντρων δεδομένων και τους ενδιαφερόμενους φορείς του κλάδου. Με την εφαρμογή αποτελεσματικών στρατηγικών και λύσεων, όπως ο ευφυής προγραμματισμός πόρων, η απομόνωση του φόρτου εργασίας και οι προηγμένες τεχνικές διαχείρισης πόρων, τα κέντρα δεδομένων μπορούν να μετριάσουν τον ανταγωνισμό και τις παρεμβολές, παρέχοντας βελτιωμένη απόδοση, καλύτερη αξιοποίηση των πόρων και αυξημένη επεκτασιμότητα.

Η επίλυση του προβλήματος του ανταγωνισμού και της παρεμβολής αποτελεί ουσιαστικό βήμα προς τη δημιουργία εύρωστων και αποδοτικών κέντρων δεδομένων, ικανών να ανταποκριθούν στις ολοένα αυξανόμενες απαιτήσεις της ψηφιακής εποχής.

Κεφάλαιο 5

Μεθοδολογία

5.1 Εισαγωγή

Στο κεφάλαιο αυτό θα αναφερθούν οι μέθοδολογιες που ακολουθήθηκαν για την μέτρηση των μετρικών ενδιαφέροντος και για την προσπάθεια βελτίωσης του placement των microservices στους διαθέσιμους επεξεργαστές.

5.2 Απομώνωση μικροϋπηρεσιών

Για να γίνει κατανόηση του πως επηρεάζεται μία μικροϋπηρεσιά από τον ανταγωνισμό στους κοινούς πόρους, τοποθετήθηκε για αρχή κάθε μικροϋπηρεσία σε απομόνωση. Έτσι έχουμε ένα baseline σενάριο για το πως συμπεριφέρεται μόνη της και πως όταν αρχίζουμε σταδιακά να τοποθετούμε δίπλα τις άλλες εφαρμογές που αρχίζουν να καταναλώνουν και να την ανταγωνίζονται για τους κοινούς πόρους. Έτσι θα δημιουργηθεί ένα προφίλ για όλα τα microservices για την ευαισθησία που παρουσιάζουν σε κάθε διαθέσιμο πόρο.

Τα stressing microbenchmarks, όπου παρουσιάστηκαν στην ενότητα 3, τοποθετούνται ένα-ένα στον ίδιο κόμβο με την προηγουμένως απομωνομένη μικροϋπηρεσία. Ξεκινάμε με τοποθέτηση ενός instance ενός microbenchmark και στη συνέχεια αυξάνουμε τα instances μέχρι το όριο των 8 που έχουμε θέσει.

Η παραγωγή φόρτου εργασίας (αιτημάτων http) γίνεται με το εργαλείο wrk2, με το οποίο και μετράμε το latency και το tail latency των αιτημάτων.

Ενώ πίεζουμε τα διάφορα μέρη του συστήματος, χρήσιμοποίωντας το perf tool μετράμε μετρικές χαμηλού επιπέδου του pipeline για να προσδιοριστεί η απόδοση του επεξεργαστή και να διαμορφώσουμε ένα προφίλ για κάθε μικροϋπηρεσία. Μία ιδιαίτερα σημαντική μετρική που μετριέται είναι οι εντολές ανά κύκλους. Η μετρική αυτή καθώς χρησιμεύει ως μέτρο της αποδοτικότητας ενός επεξεργαστή, υποδεικνύοντας πόσο καλά χρησιμοποιεί τους διαθέσιμους υπολογιστικούς πόρους. Η IPC υπολογίζεται διαιρώντας το συνολικό αριθμό εντολών που εκτελούνται κατά τη διάρκεια μιας συγκεκριμένης χρονικής περιόδου με τον αντίστοιχο αριθμό κύκλων ρολογιού.

Αν και ο IPC είναι μια πολύτιμη μετρική, θα πρέπει να ερμηνεύεται με προσοχή. Διαφορετικοί φόρτοι εργασίας, εφαρμογές μπορούν να δώσουν διαφορετικές τιμές IPC, καθιστώντας απαραίτητο να λαμβάνονται υπόψιν το συγκεκριμένο πλαίσιο και τα χαρακτηριστικά του φόρτου εργασίας. Επιπλέον, η IPC δεν παρέχει μια ολοκληρωμένη εικόνα της απόδοσης και θα πρέπει να χρησιμοποιείται σε συνδυασμό με άλλες μετρικές για να αποκτηθεί μια πιο ολοκληρωμένη εικόνα της απόδοσης.

Επιπλέον με το perf μετράμε αστοχίες ανά χίλιες εντολες (MPKI). Η μετρική MPKI αντιπροσωπεύει τον αριθμό των αστοχιών της κρυφής μνήμης ανά χίλιες εκτελεσμένες εντολές. Ποσοτικοποιεί τη συχνότητα των αστοχιών της κρυφής μνήμης σε σχέση με τον αριθμό των διεκπεραιωμένων εντολών. Η μετρική MPKI υπολογίζεται διαιρώντας τον συνολικό αριθμό των αστοχιών της κρυφής μνήμης με τον αριθμό των εκτελεσμένων εντολών, πολλαπλασιασμένο επί 1000.

Η MPKI βοηθά στην ανάλυση του φόρτου εργασίας, αναδεικνύοντας μοτίβα και χαρακτηριστικά προσπέλασης μνήμης. Βοηθά στη κατανοήση του αντίκτυπου των διαφόρων εφαρμογών στην απόδοση της κρυφής μνήμης και βοηθά στην προσαρμογή των ιεραρχιών μνήμης για συγκεκριμένους φόρτους εργασίας. Συγκεκριμένα μετράμε τις L1I, L1D, L2, L3, DTLB, ITLB για να έχουμε μια συνολική εικόνα των misses στις caches του επεξεργαστή.

Τέλος μετράμε που σπαταλούνται οι περισσότεροι κύκλοι του επεξεργαστή για να μπορέσουμε να καταλάβουμε με την topdown ανάλυση που υπάρχει χώρος για optimazations.

5.3 Topdown analysis

Με το εργαλείο perf μετράμε ταυτόχρονα που έχουμε bottlenecks στο pipeline του επεξεργαστή. Τα events που μετρήθηκαν για το 1ο επίπεδο της μεθοδολογίας topdown αναφέρονται στο κεφάλαιο 3. Στη συνέχεια με τη χρήση του εργαλείου toplev έλεγχουμε το 2ο και 3ο επίπεδο της topdown μεθοδολογίας για να βρούμε ακριβέστερα σε ποιες μονάδες της σωλήνωσης εμφανίζονται καθυστερήσεις.

5.4 Εξευρένηση διαφορετικών placements

Έχοντας όλα τα προηγούμενα δεδομένα μετρήθηκε τώρα εκ νέου ένα baseline σενάριο που ήταν όλη η εφαρμογή των κρατήσεων ξενοδοχείων να τοποθετηθεί χωρίς κανένα περιορισμό σε 2 κόμβους του μηχανήματος sandman. Στη συνέχεια των μετρήσεων χρησιμοποίουμε το 1ο από τα 2 nodes ως γενικής χρήσης όπου θα τοποθετηθούν όλα τα containers που περιλαμβάνουν βασεις δεδομένων και cache συστήματα, καθώς και όσα microservices δεν θεωρούμε critical. Στο 2ο node τοποθετούνται όσα microservices έχουν χαρακτηριστεί ως critical αλλά και όσα απλώς δεν έχουν όμοιες ευαισθησίες στους πόρους με τα ήδη τοποθετημένα microservices στον critical κόμβο.

Αρχικά με γνώμονα το IPC και το μέγιστο cpu utilazation του κάθε microservice, αναθέτουμε σε κάθε microservice ένα value ώστε μέσω της διαδικασίας που περιγράφεται στο κεφάλαιο 6 να χωρίσουμε τα microservises σε nodes. Στη συνέχεια για να ελέγξουμε αν πετύχαμε καλύτερο placement συγκρίνουμε το placement μας με το baseline και μερικά τυχαία σενάρια.

Στη συνέχεια με γνώμονα το tail latency τοποθετούνται αυτά που φαίνονται συμβάτα με βάση την ευαισθησία που δείχνουν στους πόρους, και συγκρίνονται με τις προηγούμενες περιπτώσεις. Συγκεκριμένα τοποθετούνται μαζί αυτά που δείχνουν ευαισθησία σε διαφορετικούς πόρους καθώς άλλες έρευνες έχουν δείξει ότι οι πόροι μεταξύ των containers που τρέχουν μαζί είναι ανταλλάξιμες (Πηγή [2]). Και τέλος συγκρίνουμε μαζί όλα τα προηγούμενα αποτελέσματα για να δούμε αν καταφέραμε να πετύχουμε καλύτερα αποτελέσματα από το baseline σενάριο.

5.5 Τεχνικές δυσκολίες

5.5.1 Perf events

Ένα μεγάλο κεφάλαιο στην εκπόνηση της διπλωματικής εργασίας αποτέλεσε η εξερεύνηση και κατανόηση των hardware events του επεξεργαστή και στην δική μας περίπτωση των events που σχετίζονται με την μικροαρχιτεκτονική Sandy Bridge. Τα perfmon events (Πηγή [13]) που χρησιμοποιήθηκαν ήταν αυτά που φαίνονται στον πίνακα 5.1 και όσα αναφέρθηκαν στην κεφάλαιο 3 για την μέτρηση του που σπαταλήθηκαν οι περισσότεροι κύκλοι στο pipeline. Τέλος χρησιμοποιήθηκαν και τα εξής events που παρέχει το perf: iTLB-load-misses, L1-dcache-load-misses, LLC-load-misses, LLC-store-misses, LLC-prefetch-misses, branch-misses, dTLB-load-misses, dTLB-store-misses.

ICACHE.MISSES	Αυτό το συμβάν μετράει τον αριθμό των αστοχιών της κρυφής μνήμης εντολών, του απομονωτή ροής και της κρυφής μνήμης θύματος. Η καταμέτρηση περιλαμβάνει προσπελάσεις που δεν μπορούν να αποθηκευτούν στην κρυφή μνήμη.
L2 RQSTS.CODE RD MISS	Αστοχίες στην κρυφή μνήμη L2 κατά την ανάκτηση εντολών.
L2 RQSTS.RFO MISS	Αιτήσεις RFO που χάνουν την κρυφή μνήμη L2.
L2 RQSTS.PF MISS	Αιτήματα από τους προεπιτηρητές υλικού L2 που χάνουν την κρυφή μνήμη L2.

Πίνακας 5.1: Perfmon events

Το perf κάνει expose κάποια events, απλώς αυτά εσωτερικά όπως βρέθηκε μπορεί να μην μετράνε ακριβώς αυτό που θέλουμε και νομίζουμε ότι μετράνε με βάση την περιγραγή και την ονοματολογία που δίνει το perf, οπότε πρέπει να εξετάζονται με προσοχή.

5.5.2 Δυσκολίες στην εγκατάσταση

Ένα πόλυ μεγάλο μέρος χρονικά μέρος στην εκπόνηση της διπλωματικής εργασίας αποτέλεσε η ορθή εγκατάσταση και λειτουργία των εφαρμογών στο μηχάνημα sandman. Υπήρξαν αρκετά bugs στις διάφορες εκδόσεις που γινόντουσαν release στο github με

αποτέλεσμα να γίνει προσπάθεια debugging των εφαρμογών, που εξαιτίας και της έλλειψης γνώσης στο τεχνικό τους κομμάτι δεν είχε τα επιθυμητά αποτελέσματα. Σε 2-3 περιπτώσεις που έγιναν clean τα docker images υπήρξαν προβλήματα με τα images που γινόντουσαν εκ νέου install (πχ σε ένα image είχε γίνει delete η python οπότε και δεν μπόρουσε να λειτουργήσει σωστά η εφαρμογή). Τα προβλήματα αυτά λύθηκαν με νέες εκδόσεις κώδικα στο github είτε με solutions στα issues. Εδώ φαίνεται η σημασία του γίνεται η μελέτη σε μία σταθερή έκδοση του λογισμικού, και η δυνατότητα συνεργασίας με τους προγραμματιστές της εφαμοργής.

Έγινε μεγάλη προσπάθεια μέτρησης και των εφαρμογών media microservices και social network, αναπτύχθηκαν και μετρήθηκαν για όλα τα microservices τους τα ίδια που μετρήθηκαν για το hotel reservation όπως αυτό παρουσιάζεται στο κεφάλαιο 6. Αλλά στις 2 αυτές εφαρμογές βρέθηκαν τα εξής προβλήματα:

1. Υπήρχε και στις 2 εφαρμογές ένα κοινό microservice όπου όταν έτρεχε το custom benchmark έκανε fail οπότε και ξεκίναγε διαδικασίες restart. Όλες οι μετρήσεις που γινόντουσαν μετά το restart δεν είχαν ιδιαίτερο νόημα, καθώς τα αιτήματα δεν γινόντουσαν serve. Το συγκεκριμένο πρόβλημα δεν κατάφερε να ξεπεραστεί, έγινε δοκιμή με μικρότερο load, απλώς τότε δεν είχαμε επαρκές stress της εφαρμογής οπότε και εμφάνιζε ίδια αποτελέσματα ακόμα και όταν εισαγώντουσαν τα stressing microbenchmarks.

2. Το perf tool είχε ένα bug στην έκδοση του kernel που έτρεχε ο sandman. Το σφάλμα ήταν το εξής:

Κατά την παρακολούθηση μιας διεργασίας πολλαπλών νημάτων με την επιλογή pid, το perf μερικές φορές μπορεί να επιστρέψει την αποτυχία sys perf event open με 3(No such process) αν κάποια από τις διεργασίες νήματα της διεργασίας πεθαίνουν πριν ανοίξουμε το συμβάν. Ωστόσο, θέλουμε με το perf να συνεχίσουμε την παρακολούθηση των υπόλοιπων νημάτων και χωρίς να οδηγούμαστε σε αποτυχία όλης της μέτρησης. Το bug αυτό έχει επιλυθεί σε νεότερες εκδόσεις kernel.

Για το παραπάνω πρόβλημα έγιναν οι εξής δοκιμές για την επίλυση του. Η πρώτη ήταν αντί να ξεκινάει πρώτα το load generation και μετά το perf να ξεκινάνε ανάποδα μήπως έχοντας ξεκινήσει τις μετρήσεις το perf μετά αγνοωούσε τα νήματα που πέθαιναν, αλλά κάτι τέτοιο δεν συνέβη. Στη ίδια λογική δοκιμάστηκε να ξεκινάει πρώτα το perf και με signals να ξεκινάει η διεργασία που μετράμε με μία καθυστέρηση, χωρίς ούτε αυτό να καταφέρει να λύσει το πρόβλημα. Επόμενη λύση που δοκιμάστηκε ήταν να μετρηθούν

τα events αντί με counting mode με sampling αλλά και πάλι εμφανίστηκε το ίδιο error. Η μόνη λύση ικανή να λύσει το παραπάνω πρόβλημα ήταν να μετρηθούν τα events όχι σε process mode αλλά για συγκεκριμένα cpus με την επιλογή που δίνει το perf με το (- C). Απλώς αυτό προϋποθέτει να τοποθετηθεί το microservice σε τόσα cpus όσα και το μέγιστο cpu utilization του και το perf να μετράει τα συγκεκριμένα cpus. Αυτή η λύση λόγω αύξησης της πολυπλοκότητας και χρονικού περιορισμού δεν αναπτύχθηκε πλήρως αλλά δοκιμάστηκε και ήταν η μόνη που δεν εμφάνιζε error.

Σχετικά με το hotel reservation που και τελικά αποτέλεσε τη βασική εφαρμογή της παρούσας εργασίας υπήρξαν πολλά προβλήματα στην εγκατάσταση της, διορθώσεις που χρειάστηκαν να γίνουν σε lua και python scripts, τα οποία τελικά γινόντουσαν fix σε επόμενες εκδόσεις, αλλά και configuration προβλήματα σχετικά με τη εσωτερική επικοινωνία των microservices όπου και αυτό διορθώθηκε σε επόμενες εκδόσεις τις εφαρμογής.

5.5.3 Reservation Mongodb

Ένα πρόβλημα που παρατήρηθηκε με την τεχνική active benchmarking (Πηγές [1], [9]) δηλαδή με τη χρήση monitoring tools (όπως top, docker stats) κατά τη διάρκεια εκτέλεσης των benchmarks ήταν ότι η βάση δεδομένων mongodb που αντιστοιχούσε στο microservice reservation σήκωνε πάρα πολλά threads και έφτανε μέχρι και 3000 % cpu usage όταν στελνόντουσαν πολλά reservation requests. Η υπόθεση στη συγκεκριμένη περίπτωση είναι ότι έχει γίνει missconfiguration στον κώδικα του πως θα γίνεται η επικοινωνία με τη βάση δεδομένων για κάθε request στο συγκεκριμένο microservice και αντί να χρησιμοποιείται ένας client στη μορφή sigleton με παράλληλο χειρισμό των αιτημάτων, αρχικοποιούνταν νέο instance του client για κάθε request και πιθανότατα δεν γινόταν clean up όταν αυτό δεν χρειαζόταν πλέον καθώς και το connection κρατιόταν ανοιχτό για παραπάνω ώρα από ότι χρειαζόταν. Σε κάθε περίπτωση η εξήγηση ξεφεύγει από της σκοπία της διπλωματικής και αυτό που αξίζει να τονιστεί είναι ότι τελικά το request για reservation φαίνεται τελικά να αποτυγχάνει σε μεγάλο load και περαιτέρω διευρεύνηση είναι απαραίτητη.

5.5.4 Jaegar tracing

Η εφαρμογή χρησιμοποιεί για tracing το εργαλείο jagaer (Πηγή [19]). Όπως παρατηρήθηκε με active benchmarking αυτό το εργαλείο με την πάροδο του χρόνου που είναι

up η εφαρμογή αρχίζει και καταλαμβάνει μεγαλύτερο μέρος των επεξεργαστών φτάνοντας μέχρι και τιμή 1200% δηλαδή 12 threads! Αυτό σημαίνει ότι πιέζει τα υπόλοιπα microservices ως προς τους διαθέσιμους πυρήνες και η απόδοση της εφαρμογής έχει ξεκάθαρη πτώση. Οι μετρήσεις επηρεάζονται δραματικά όταν είναι ενεργοποιημένο αυτό το εργαλείο.

Κεφάλαιο 6

Πειραματική Αξιολόγηση

Στο κεφάλαιο θα παρουσιαστεί η πειραματική αξιολόγηση. Παρακάτω αναπτύσσεται η μεθολογία λήψης των πειραματικών δεδομένων καθώς και η ανάλυση τους για να καταλήξουμε στα απαραίτητα συμπεράσματα.

6.1 Παράμετροι αξιολόγησης

Οι παράμετροι που μετρήθηκαν ήταν το cpu utilization για κάθε μικρόϋπηρεσια ώστε στα διάφορα σενάρια μετρήσης να αποφευχθεί η έλλειψη πόρων του συγκεριμένου service και να μετρήσουμε το πραγματικό interference και την αντοχή της κάθε μικροϋπηρεσίας σε διάφορους πόρους. Βασικός γνώμονας για την αξιολόγηση είναι το IPC και το latency σκορ κάθε μικροϋπηρεσίας.

6.2 Οργάνωση πειραμάτων

6.2.1 Φιλοξενία εφαρμογών

Η ρύθμιση των μετρήσεων πραγματοποιήθηκε σε ένα σύστημα με όνομα Sandman, το οποίο αποτελείται από τέσσερις κόμβους. Κάθε κόμβος ήταν εξοπλισμένος με οκτώ επεξεργαστές, ο καθένας από τους οποίους υποστήριζε δύο νήματα, με αποτέλεσμα να υπάρχουν συνολικά 32 επεξεργαστές δηλαδή 64 νήματα. Ο Sandman αποτελείται από τα μοντέλα επεξεργαστών 4 x Intel Xeon E5-4620 (Sandy Bridge 8-core/16-threads) που είναι χωρισμένοι σε numa κόμβους.

Δύο κόμβοι αφιερώθηκαν στη φιλοξενία της εφαρμογής. Αυτοί οι κόμβοι χρησίμευαν ως η κύρια υποδομή για την εκτέλεση της εφαρμογής και τη διαχείριση των αιτημάτων των χρηστών. Σε κάθε microservice δώθηκε η δυνατότητα να χρησιμοποίησει όσους επεξεργαστές χρειάζεται μέσα σε αυτούς τους κόμβους και δεν υπήρξε κανένας περαιτέρω περιορισμός.

Ο τρίτος κόμβος χρησιμοποιήθηκε για τη φιλοξενία της γεννήτριας φόρτου εργασίας. Με την τοποθέτηση της γεννήτριας φόρτου εργασίας σε ξεχωριστό κόμβο, διασφαλίστηκε ότι η παραγωγή συνθετικού φόρτου εργασίας δεν κατανάλωνε πόρους από την εφαρμογή. Αυτός ο διαχωρισμός επέτρεψε ακριβείς μετρήσεις επιδόσεων χωρίς παρεμβολές.

Ο τέταρτος κόμβος χρησιμοποιήθηκε για την απομόνωση του υπό μελέτη container από την υπόλοιπη εφαρμογή. Η απομόνωση αυτή αποσκοπούσε στην εξάλειψη του θορύβου και των παρεμβολών που προκαλούνταν από κοινούς πόρους, όπως η κρυφή μνήμη και η CPU. Με την απομόνωση του container, κατέστη δυνατή η ανάλυση και η μελέτη των διαφόρων παραμέτρων που επηρεάζουν την απόδοσή του, χωρίς εξωτερικοί παράγοντες να επηρεάζουν τα αποτελέσματα.

6.2.2 Εκτέλεση των μέτρησεων

Για τη διεξαγωγή των μετρήσεων αναπτύχθηκε benchmark και βοηθητικά scripts με τη χρήση της γλώσσας Bash scripting και της Python. Το benchmark δέχεται ως είσοδο τον τρόπο με τη λειτουργία - μέτρηση που θα εκτελέσει, και για κάθε microservice κάνει μετρήσεις για περίπου 2 λεπτά. Αναπτύχθηκαν 3 λειτουργίες για καλύτερη αυτοματοποίηση των μετρήσεων.

Λειτουργία 1: Η πρώτη λειτουργία επικεντρώθηκε στη μέτρηση διαφόρων μετρικών απόδοσης του επεξεργαστή και του χρόνου απόκρισης. Περιλάμβανε την τοποθέτηση ολόκληρης της εφαρμογής σε δύο κόμβους, προσομοιώνοντας μια τυχαία ανάπτυξη της εφαρμογής σε 16 επεξεργαστές.

Λειτουργία 2 - Απομόνωση μικρουπηρεσιών: Η δεύτερη λειτουργία αποσκοπούσε στη μέτρηση της απόδοσης μιας συγκεκριμένης μικρουπηρεσίας σε απομόνωση. Με την απομόνωση της σε έναν μόνο κόμβο, κατέστη δυνατή η αξιολόγηση της απόδοσής της χωρίς να υπάρχει ανταγωνισμός ή παρεμβολή από άλλα στοιχεία της εφαρμογής.

Λειτουργία 3 - Δοκιμή καταπόνησης: Η τρίτη λειτουργία περιλάμβανε την απομόνωση

της στοχευόμενης μικρουπηρεσίας σε έναν κόμβο και την υποβολή της σε διάφορες δοκιμές καταπόνησης. Αυτές οι δοκιμές καταπόνησης σχεδιάστηκαν για να αξιολογηθεί η ευαισθησία της μικρουπηρεσίας σε διάφορους παράγοντες καταπόνησης των πόρων. Με την έγχυση στρες και τη μέτρηση της προκύπτουσας απόδοσης, επέτρεπε την ανάλυση της συμπεριφοράς της μικροεξυπηρέτησης υπό δύσκολες συνθήκες.

Η διαδικασία της απομόνωσης επιτυγχάνεται με τη χρήση της εντολής docker update (Πηγή [4]) που μας επιτρέπει να θέσουμε σε ποιους επεξεργαστές θα εκτελείται το επιθυμητό container.

Η περιγραφόμενη διαδικασία εκτελέστηκε για όλες τις μικρουπηρεσίες και τις δοκιμές καταπόνησης. Ταυτόχρονα, δημιουργήθηκε φόρτος εργασίας με τη χρήση του εργαλείου wrk2 (Πηγή [20]). Αυτή η παραγωγή φόρτου εργασίας περιλάμβανε την αποστολή αιτημάτων στην πύλη εφαρμογής. Τα αιτήματα που εστάλησαν από το wrk2 δεν ήταν ομοιόμορφα ανά διαθέσιμο endpoint της εφαρμογής, καθώς τους αποδόθηκαν διαφορετικά βάρη παραγωγής με βάση τη σημασία τους στο πλαίσιο της εφαρμογής.

Στο πλαίσιο της εφαρμογής κρατήσεων ξενοδοχείων, το αίτημα αναζήτησης, που είναι το πιο κοινό, παίρνει μεγαλύτερη βαρύτητα κατά την παραγωγή φόρτου εργασίας. Αυτή η σταθμισμένη παραγωγή φόρτου εργασίας αποσκοπούσε στην προσομοίωση των προτύπων χρήσης στον πραγματικό κόσμο, όπου ορισμένοι τύποι αιτήσεων είναι πιο συχνοί και κρίσιμοι από άλλους.

Ο τρόπος με τον οποίο πάρθηκε η απόφαση για τις παραμέτρους που θα χρησιμοποιηθούν στο εργαλείο wrk2 όπως αυτό παρουσιάστηκε στο κεφάλαιο 3, ήταν να δωθούν σταδιακά μεγαλύτερες τιμές στον αριθμό των συνδέσεων και στα αιτήματα ανά δευτερόλεπτο μέχρι η χρονική απόκριση της εφαρμογής στα αιτήματα να ξεπεράσει κάποιο χρονικό όριο που είχε τεθεί, το οποίο ήταν τα 50ms. Η αναλογία ανάμεσα στον αριθμό των συνδέσεων και τα αιτήματα ανά δευτερόλεπτο μπορεί να ποικίλει ανάλογα με τη φύση της εφαρμογής και τις αναμενόμενες απαιτήσεις φορτίου. Η αρχική χρήση χαμηλών τιμών για τις παραμέτρους του wrk2 βοηθά στην αποφυγή υπερφόρτωσης της εφαρμογής σε αρχικά στάδια. Στη συνέχεια, η αύξηση του φορτίου αποκαλύπτει τα όρια της εφαρμογής.

Η μικροαρχιτεκτονική ανάλυση αποτελεί μία χρονοβόρρα διαδικασία για να εφαρμοστεί και το benchmark που αναπτύχθηκε μπορεί να χρησιμοποιηθεί σε πολλές εφαρμογές με τα σωστά configuration απλώς η δημιουργία διαφορετικού για κάθε περίπτωση θα μας δώσει πιο ακριβή και πιο σαφή αποτελέσματα.

6.2.3 Εισαγωγή δοκιμών καταπόνησης

Η εισαγωγή των stress tests έγινε στην κατάσταση απομόνωσης του υπό εξέταση microservice. Οι μετρήσεις ήταν κλιμακούμενες, δηλαδή έγιναν με το stress test να τρέχει διαδοχικά σε 1 - 2 - 4 - 8 νήματα για να αποτυπωθεί πως αλλάζουν οι τιμές του IPC και latency ενώ αυξάνουμε την πίεση του συγκεκριμένου πόρου.

Αυτή η ολοκληρωμένη ανάλυση των μεμονωμένων μικροϋπηρεσιών και της συνολικής απόδοσης της εφαρμογής βοηθά στην κατανόηση της συμπεριφοράς του συστήματος, στον εντοπισμό πιθανών σημείων συμφόρησης και στη βελτιστοποίηση της κατανομής των πόρων για τη βελτίωση της συνολικής απόδοσης του συστήματος και της εμπειρίας του χρήστη.

Στον Πίνακα 6.1 φαίνεται ακριβώς η τοποθέτηση στα νήματα του πρώτου node του μηχανήματος sandman των microbenchmarks. Συγκεκριμένα αναφέρονται το gemm που τοποθετήθηκε στα threads 32, 32-33, 32-35, 32-39 στις 4 φορές που έγινε η μέτρηση όπου και τοποθετούταν σε περισσότερα threads για να αυξηθεί η πίεση που ασκεί, ακριβώς την ίδια τοποθέτηση είχαν τα microbenchmarks memBw, memCap. Το stress test l3 τοποθετείται μόνο σε ένα thread γιατί στρεσσάρει όλη την l3 cache σταδιακά, ενώ άμα βάλουμε και άλλα instance να τρέξουν θα αρχίζει να στρεσάρει το memory bandwidth και το capacity που δεν το θέλουμε. Τέλος βλέπουμε το stress test l1i το οποίο τοποθετείται προοδευτικά στα threads 32, 32-35 και 32-39, ίδιο placement εφαρμόζεται και στα stress tests l1d, l2. Το microservice που είναι προς εξέταση είναι τοποθετημένο στο ίδιο node και συγκεκριμένα στα threads 0-7, ενώ όλη η υπόλοιπη εφαρμογή στα threads 8-23, 40-55 (στο σύνολο σε 32 threads).

6.3 Αποτελέσματα της μελέτης

Η υπό εξέταση εφαρμογή είναι η hotel reservation όπως αυτή έχει παρουσιαστεί στο κεφάλαιο 3.

6.3.1 IPC απομονωμένης μικροϋπηρεσίας

Στα σχήματα 6.1, 6.3, 6.5, 6.7, 6.9, 6.11, 6.13, 6.15 απεικονίζονται οι εντολές ανά κύκλους (IPC) για κάθε service όταν αυτό έτρεχε σε απομώνωση από την υπόλοιπη εφαμοργή και όταν εισήχθησαν τα διάφορα stressing microbenchmarks ενώ στα σχήμα-

Microbenchmark	Thread(s) που τοποθετήθηκε
gemm32	32
gemm33	32-33
gemm35	32-35
gemm39	32-39
l3 32	32
l1i32	32
l1i35	32-35
l1i39	32-39

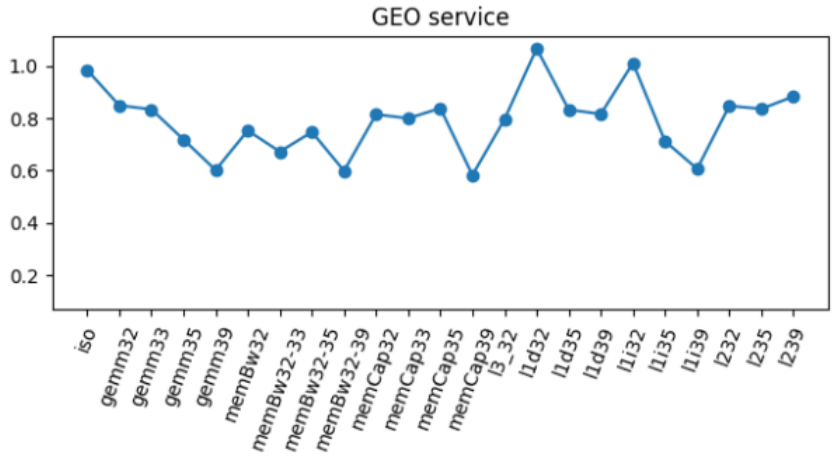
Πίνακας 6.1: Microbenchmarks placement

τα 6.2, 6.4, 6.6, 6.8, 6.10, 6.12, 6.14, 6.16 βλέπουμε την ποσοστιαία μεταβολή του IPC από τη βασική περίπτωση της απομώνωσης.

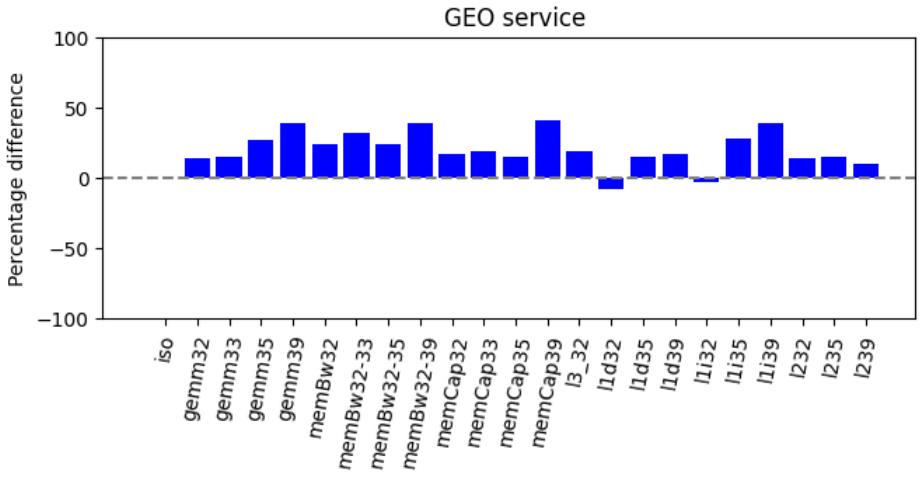
Σχετικά με το IPC του geolocation microservice όπως φαίνεται στο Σχ. 6.1 έχουμε τις υψηλότερες τιμές σε σχέση με τα υπόλοιπα microservices που σε κάποιες περιπτώσεις ξεπερνάει και το φράγμα του 1 IPC, όντας το μοναδικό microservice που το ξεπέρασε. Φαίνεται να σημειώνει τις χαμηλότερες τιμές του όταν βρισκόταν υπό πίεση οι πυρήνες του κόμβου που ήταν τοποθετημένο, καθώς και στις περιπτώσεις που έχουμε πολλαπλά instances που στρεσάρουν το εύρος ζώνης της μνήμης, τη χωρητικότητα της μνήμης καθώς και την L1 cache. Ενώ από το Σχ. 6.2 βλέπουμε μεγαλύτερες μεταβολές στο memory capacity microbenchmark και στο L1 instruction cache microbenchmark ενώ πιο σταδιακή στο gemm και στο memory bandwidth.

Σχετικά με το IPC του frontend microservice όπως φαίνεται στο Σχ. 6.3 η τιμή του οριακά υπερβαίνει το 0,6. Φαίνεται να σημειώνει τις χαμηλότερες τιμές του όταν βρισκόταν υπό πίεση οι πυρήνες του κόμβου που ήταν τοποθετημένο, καθώς και στις περιπτώσεις του που στρεσάρονται το εύρος ζώνης της μνήμης, η χωρητικότητα της μνήμης, με το εύρος ζώνης να έχει τη χαμηλότερη τιμή . Ενώ από το Σχ. 6.4 βλέπουμε καθαρά ότι σε όλες τις περιπτώσεις του που στρεσάραμε το memory bandwidth είχε μεγάλη πτώση του IPC φτάνοντας στο 50%.

Το IPC του profile microservice όπως φαίνεται στο Σχ. 6.5 η τιμή του αγγίζει το 0,8 που το καθιστά το δεύτερο υψηλότερο. Φαίνεται να σημειώνει τις χαμηλότερες τιμές του όταν βρισκόταν υπό πίεση οι πυρήνες του κόμβου που ήταν τοποθετημένο, και



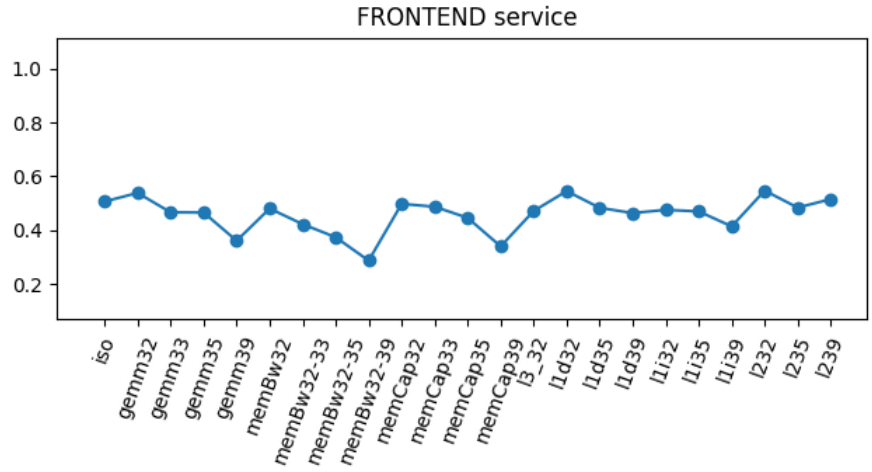
Σχήμα 6.1: Geolocation microservice IPC



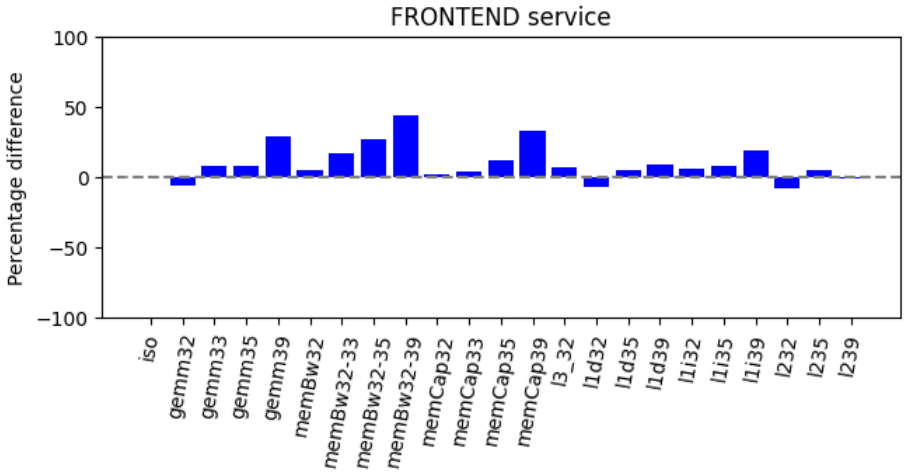
Σχήμα 6.2: Geolocation microservice IPC

στις περιπτώσεις του που έχουμε πολλαπλά instances που στρεσάρουν το εύρος ζώνης της μνήμης, τη χωρητικότητα της μνήμης καθώς και την L1 cache. Με το memory bandwidth να ξεχωρίζει και σε αυτήν την περίπτωση. Ενώ από το Σχ. 6.2 βλέπουμε πτώση σε όλες τις περιπτώσεις που έτρεχε κάποιο microbenchmark.

Το rate microservice όπως φαίνεται στο Σχ. 6.7 έχει χαμηλό αρχικό IPC. Φαίνεται να σημειώνει τις χαμηλότερες τιμές του στις περιπτώσει του gemm, memory bandwidth και memory capacity. Ενώ από το Σχ. 6.8 βλέπουμε μεγαλύτερες μεταβολές στο



Σχήμα 6.3: Frontend microservice IPC

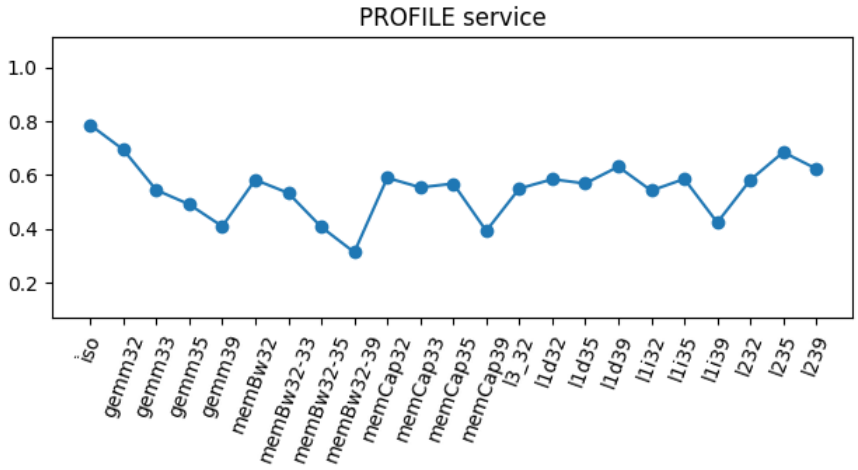


Σχήμα 6.4: Frontend microservice IPC

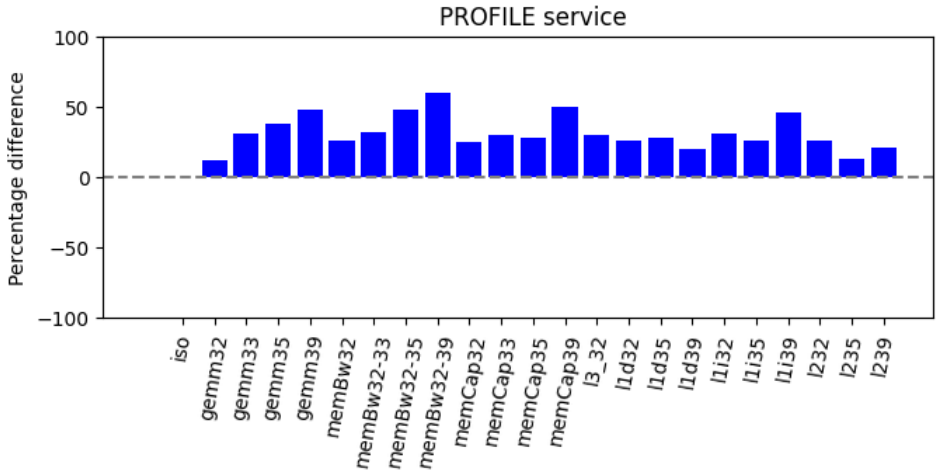
memory capacity microbenchmark και στο L1 instruction cache microbenchmark ενώ πιο σταδιακή στο gemm και στο memory bandwidth.

Το recommendation microservice όπως φαίνεται στο Σχ. 6.9 κινείται σε χαμηλές τιμές IPC. Με βάση και το Σχ. 6.10 φαίνεται να επηρεάζεται μόνο από το memory bandwidth η τιμή του IPC και στις άλλες περιπτώσεις δεν σημειώνει καμία σημαντική πτώση.

Το reservation microservice όπως φαίνεται στο Σχ. 6.11 κινείται σε χαμηλές τιμές IPC. Με βάση και το Σχ. 6.2 φαίνεται να επηρεάζεται από το memory bandwidth και



Σχήμα 6.5: Profile microservice IPC

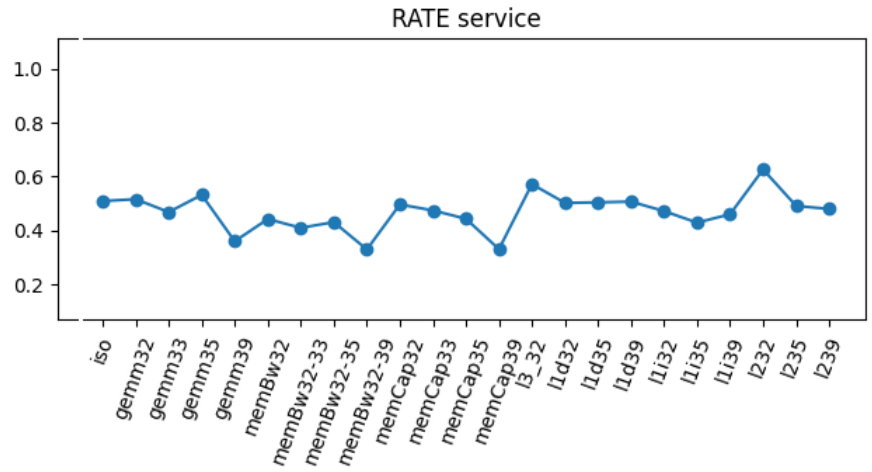


Σχήμα 6.6: Profile microservice IPC

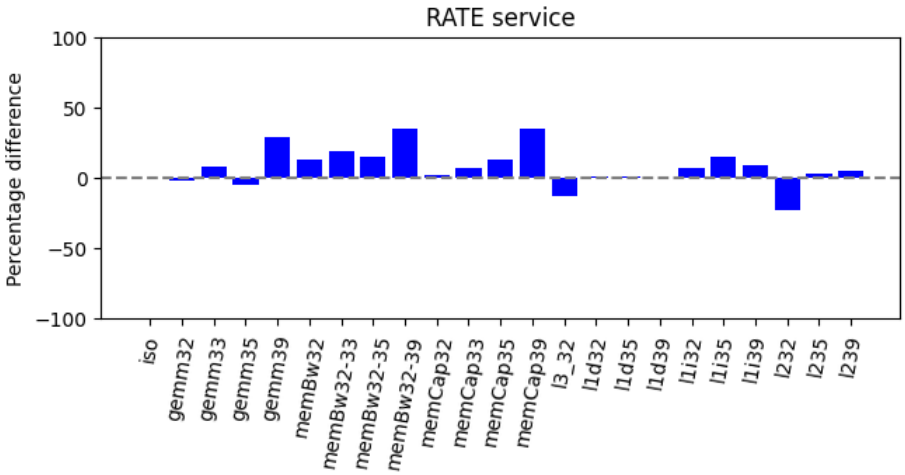
το memory capacity με μεγάλη πτώση οριακή περίπτωση που στρεσάρεται από όλα τα instances.

Το search microservice όπως φαίνεται στο Σχ. 6.13 έχει περίπου 0,6 IPC. Φαίνεται να επηρεάζεται κυρίως από το memory bandwidth και λιγότερο από τα gemm, memory capacity microbenchmarks. Κάτι που επιβεβαιώνεται και απο Σχ. 6.13.

Το user microservice όπως φαίνεται στο Σχ. 6.15 έχει περίπου 0,5 IPC. Φαίνεται να



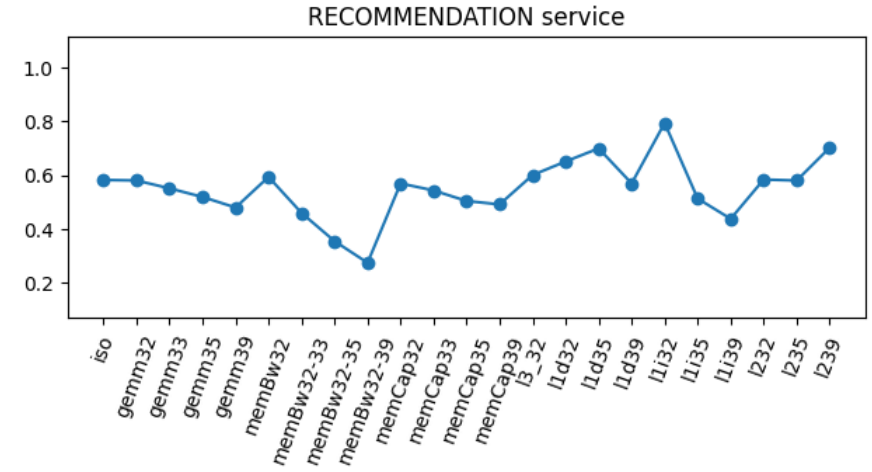
Σχήμα 6.7: Rate microservice IPC



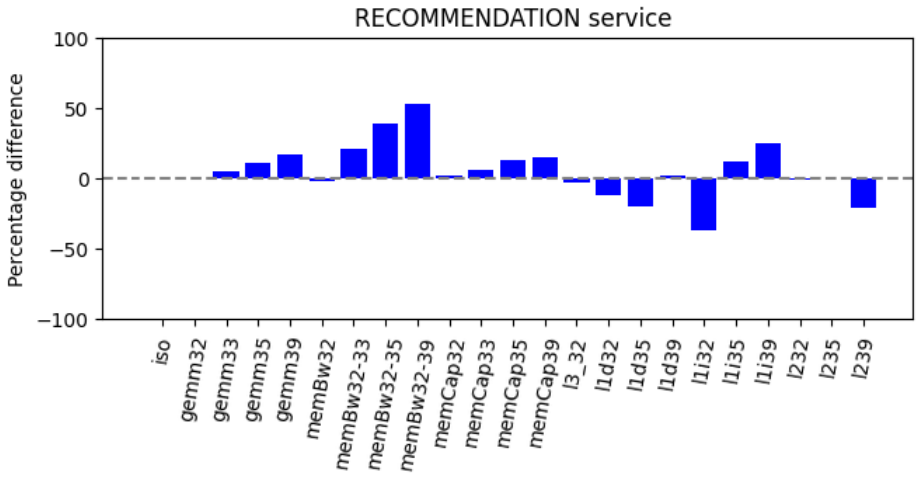
Σχήμα 6.8: Rate microservice IPC

επηρεάζεται δραματικά από το memory bandwidth και όχι ιδιαίτερα από τα υπόλοιπα microbenchmarks. Είναι το μοναδικό το οποίο φτάνει 50% πτώση στο IPC ενώ το memory bandwidth τρέχει μόλις σε 2 φυσικά νήματα και στα 8 νήματα σημειώνει πτώση γύρω στο 80%.

Παρατηρούμε ότι η τιμή του IPC είναι μικρή σε όλες τις περιπτώσεις των μετρήσεων (περίπου 0,5) ενώ το microbenchmark που ξεκάθαρα επηρεάζει όλες τις εφαρμογές είναι το memory bandwidth, με τα gemm και memory capacity να ακολουθούν.



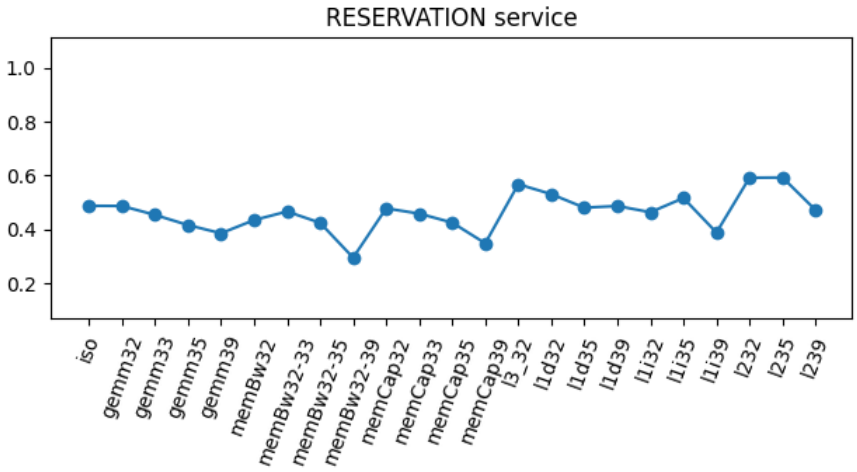
Σχήμα 6.9: Recommendation microservice IPC



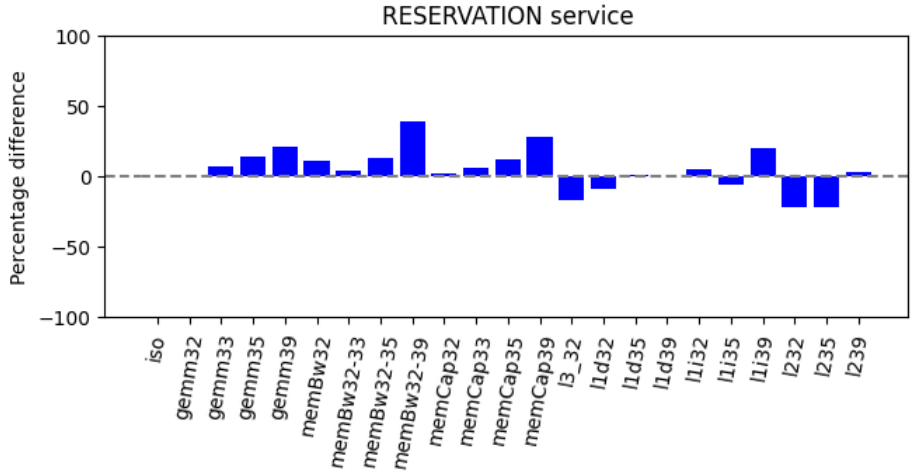
Σχήμα 6.10: Recommendation microservice IPC

6.3.2 TopDown ανάλυση

Στο Σχ. 6.17 βλέπουμε το πρώτο επίπεδο της topdown ανάλυσης, όπου φαίνεται σε ποιο μέρος της σωλήνωσης του επεξεργαστή έχουμε τα περισσότερα cycle stalls. Όπως μπορούμε να παρατηρήσουμε όλα τα microservices εκτός του geolocation έχουν ποσοστό κύκλων λίγο πάνω από 50% στο frontend μέρος και περίπου 25% στο backend. Στην περίπτωση του geolocation microservice οι τιμές frontend και backend είναι πολύ

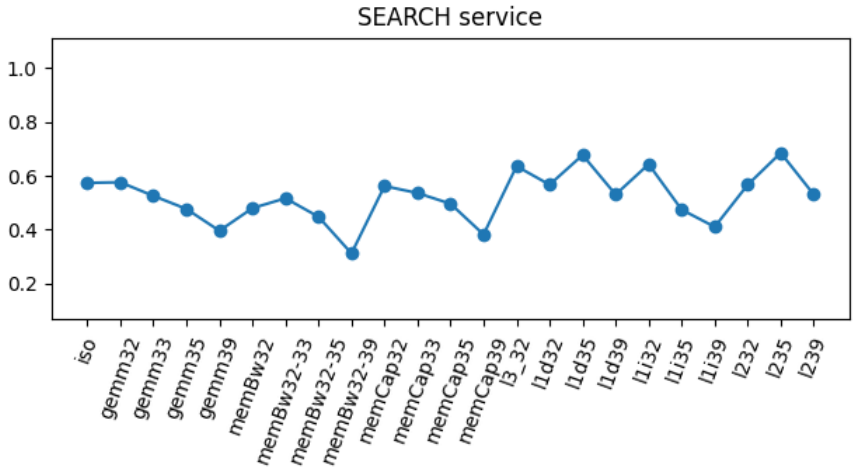


Σχήμα 6.11: Reservation microservice IPC

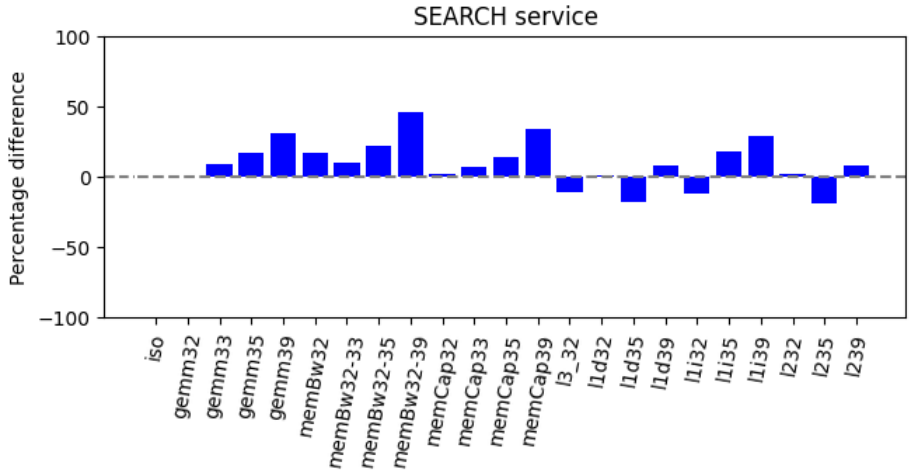


Σχήμα 6.12: Reservation microservice IPC

κοντά και έχουμε σαφή άνοδο στα retiring instructions γεγονός που δικαιολογεί και το υψηλότερο IPC. Το IPC στα υπόλοιπα microservices είναι σχεδόν το ίδιο και έχει τιμή περίπου 0,5 με εξαίρεση το profile microservice που έχει τιμή γύρω στα 0,75. Η κατηγορία bad speculation, η οποία υπενθυμίζεται ότι αφορά τους κύκλους που σπαταλήθηκαν λόγω λανθασμένων εκτιμήσεων, περιλαμβάνει δύο τμήματα: κύκλους που χρησιμοποιούνται για την εκτέλεση uops που τελικά δεν αποσύρονται, καθώς και κύκλους στους οποίους η σωλήνωση μπλοκαρίστηκε λόγω ανάκτησης από προηγούμενες λανθασμένες



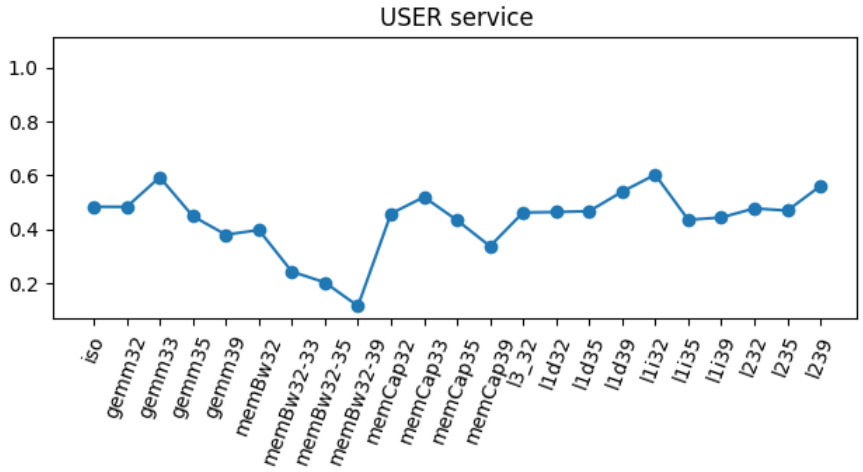
Σχήμα 6.13: Search microservice IPC



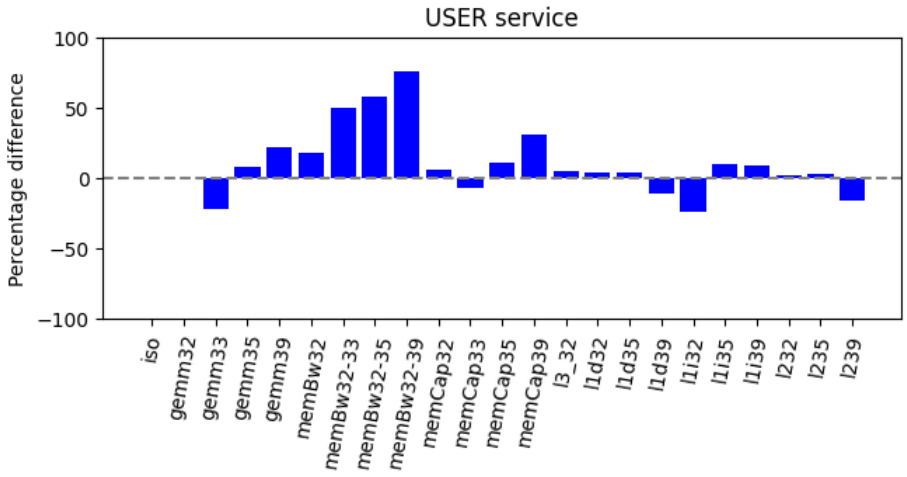
Σχήμα 6.14: Search microservice IPC

υποθέσεις. Η τιμή σε αυτή την περίπτωση βλέπουμε να κυμαίνεται από 5,5% έως 9,5%, τιμές που είναι οριακά αποδεκτές, καθώς αν ξεπερνούσαν το 10% θα χρειαζόταν περαιτέρω έρευνα γιατί υπάρχουν τόσα πολλά miss predictions. Τέλος το retiring για όλα είναι στο 13 με 15% εκτός από το geolocation όπως αναφέρθηκε προηγουμένως.

Στη συνέχεια για να αποκτηθεί καλύτερη εικόνα σε βαθύτερα επίπεδα της σωλήνωσης του επεξεργαστή χρησιμοποιήθηκε το εργαλείο toplev (Πηγή [14]) το οποίο και αυτό εσωτερικά χρησιμοποιεί τους PMCs (perf monitoring counters) και δίνει πολλές επι-

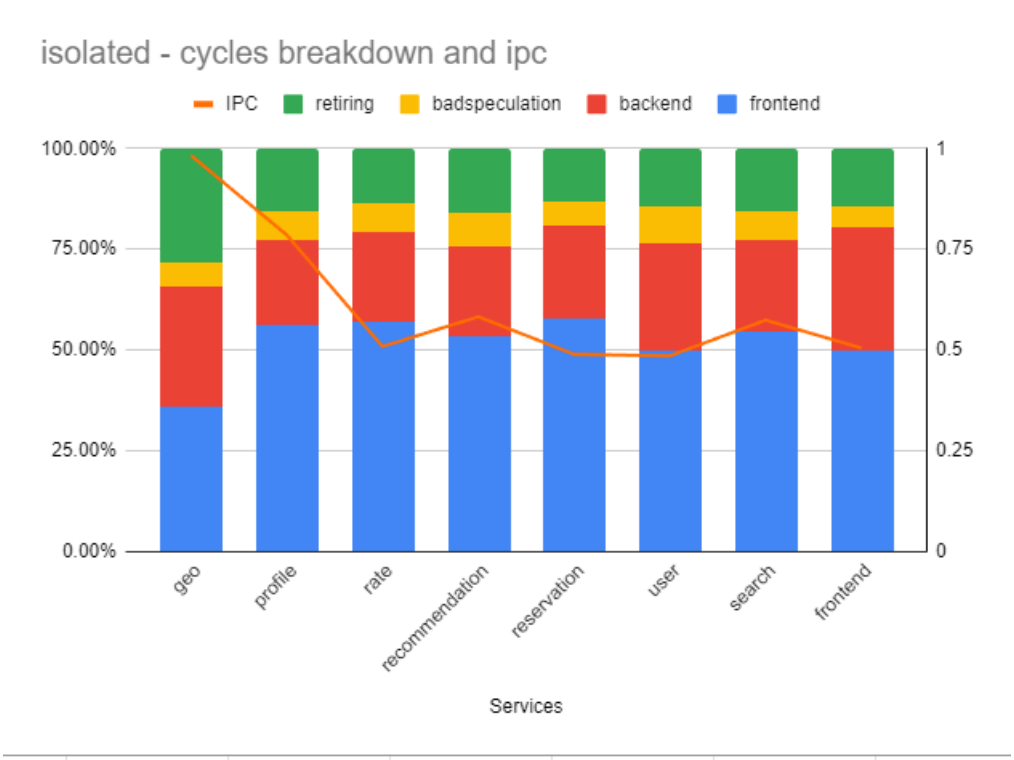


Σχήμα 6.15: User microservice IPC



Σχήμα 6.16: User microservice IPC

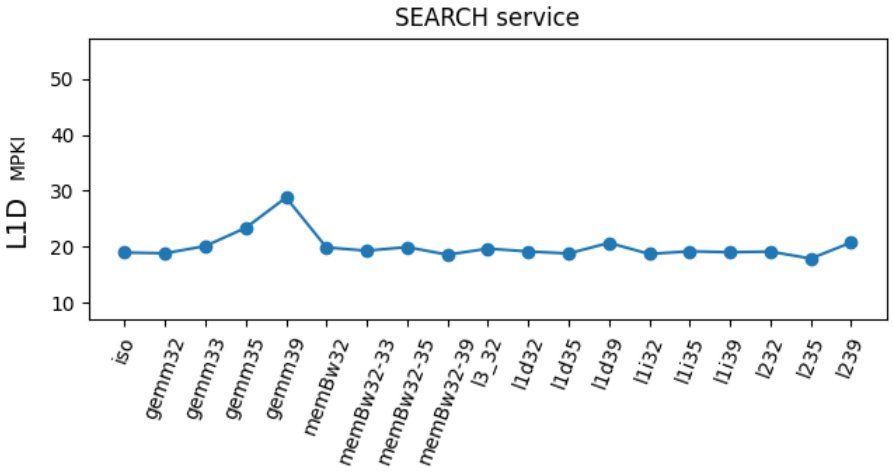
λογές με παραμέτρους για customization και να αποκτηθεί μια καλύτερη εικόνα που βρίσκονται τα bottlenecks. Παρουσιάζονται τα αποτελέσματα για το search microservice στον Πίνακα 6.2 στο επίπεδο 1 μας δείχνει ως bottleneck το frontend, πηγαίνοντας ένα επίπεδο κάτω βλέπουμε να φαίνεται ως bottleneck η κατηγορία fetch latency και τέλος στο τρίτο επίπεδο βλέπουμε ότι οι περισσότεροι κύκλοι σπαταλούνται στην κατηγορία branch resteers. Η κατηγορία fetch latency αφορά βασικά τα i-cache misses ενώ η κατηγορία Branch Resteers (Πηγή [21]) αποτελεί τη μονάδα που εκτιμάει την



Σχήμα 6.17: Cycle breakdown

καθυστέρηση του Front-End στην ανάκτηση λειτουργιών από τη διορθωμένη διαδρομή. Στο Backend που και εκεί έχουμε ένα υψηλό ποσοστό των κύκλων το εργαλείο μας προσανατολίζει στις κατηγορίες core bound στο 2ο επίπεδο και port utilization στο 3ο επίπεδο, αυτή η μετρική αντιπροσωπεύει το πόσο πολύ τα ζητήματα του Core που δεν αφορούν τη μνήμη αποτελούσαν σημείο συμφόρησης. Η έλλειψη υπολογιστικών πόρων υλικού ή οι εξαρτήσεις από τις οδηγίες του λογισμικού κατηγοριοποιούνται και οι δύο στο πλαίσιο του Core Bound.

Η topdown ανάλυση μπορεί να μας υποδείξει σε ποια σημεία του pipeline έχουμε bottlenecks για να εφαρμόστουν σχετικές βελτιστοποιήσεις και ακόμα μεγαλύτερη αξία θα έχει να συνδυαστεί με το perf record και τα flamegraphs (Πηγή [8]) για να μας υποδείξει τα μεγαλύτερα hotspots που υπάρχουν στον κώδικα. Ακόμα η μελέτη του cycle breakdown φαίνεται να συμφωνεί με αντίστοιχες μελέτες σε microservices όπου το μεγαλύτερο ποσοστό των κύκλων έχει καθυστερήσεις στο frontend (Πηγές [6], [7]). Τα frontend stalls κυριαρχούν σε όλα τα microservices γεγονός που το καθιστά το



Σχήμα 6.18: L1D MPKI

bottleneck του pipeline.

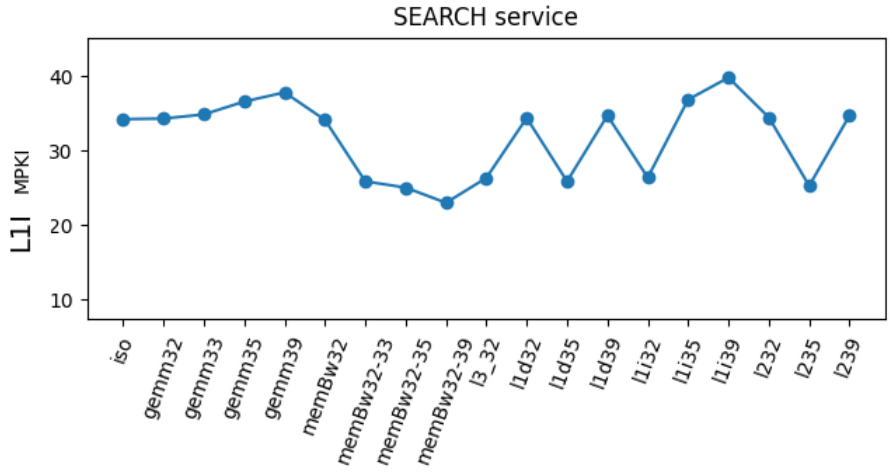
Frontend	47%
Frontend - Fetch Latency	40%
Frontend - Fetch Latency - Branch Resteers	20%
Backend	34%
Backend - Core Bound	25%
Backend - Core Bound - Port Utilization	65%

Πίνακας 6.2: Toplev results

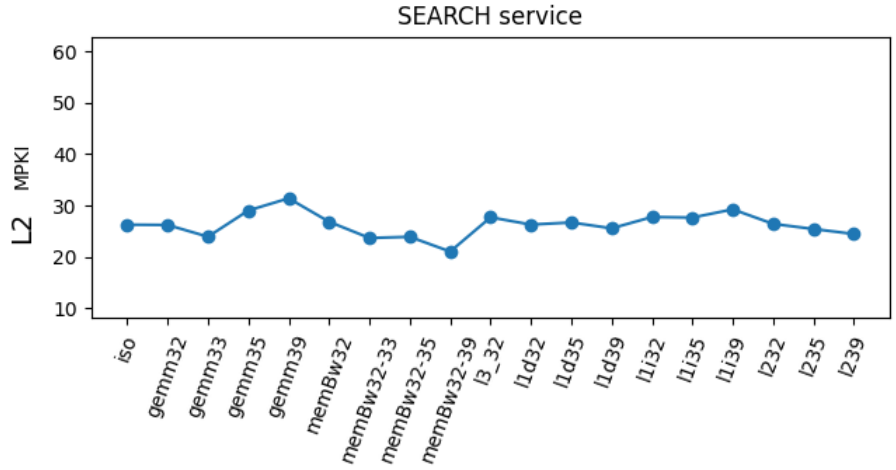
6.3.3 Εξέταση τιμών των διαφόρων cache

Σε αυτήν την ενότητα θα παρουσιαστούν τα αποτελέσματα των cache misses για ένα microservice και τα υπόλοιπα θα παραλειφθούν για να μη δωθεί μεγάλη έκταση σε αυτήν την υποενότητα. Το microservice που θα παρουσιαστεί είναι το search που είναι υπεύθυνο για την αναζήτηση ξενοδοχείων με βάση τα κριτήρια του χρήστη.

Στο Σχ. 6.19 φαίνονται οι αστοχίες ανά 1000 εντολές (MPKI) για την L1D cache όπου βλέπουμε ότι δεν υπάρχουν μεγάλες αποκλίσεις ενώ υπήρχε πίεση από τα διάφορα microbenchmarks και οι τιμές κινούνται γύρω στις 20 MPKI, με μόνη εξαίρεση την περίπτωση που στρεσάρεται ο επεξεργαστής όπου φτάνει τα 30 MPKI.



Σχήμα 6.19: L1I MPKI

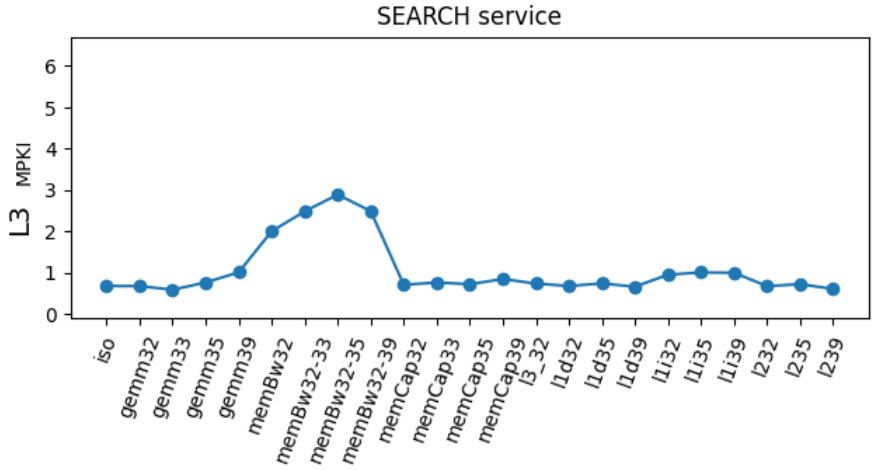


Σχήμα 6.20: L2 MPKI

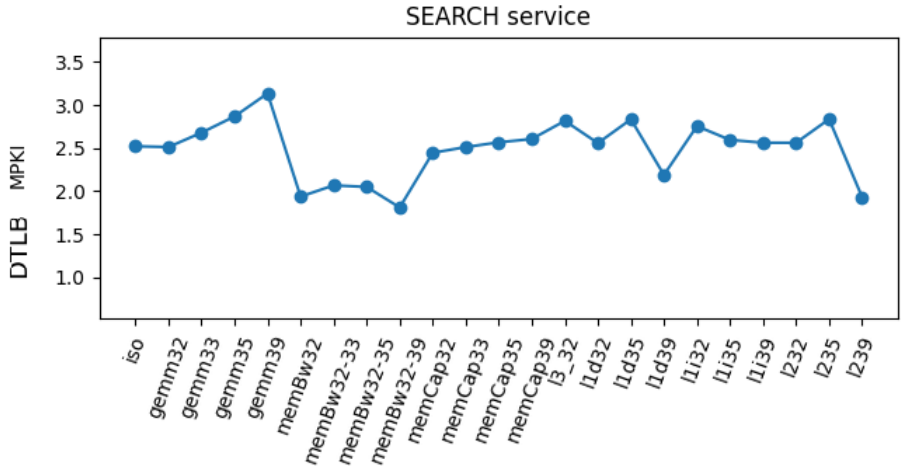
Στο Σχ. 6.18 φαίνονται οι αστοχίες για την L1I cache όπου παρατηρούμε ότι έχουμε υψηλές τιμές αστοχιών κυμαίνονται στις διάφορες περιπτώσεις απο 25 μέχρι και 40 αστοχίες ανά 1000 εντολές.

Στο Σχ. 6.20 φαίνονται οι αστοχίες για την L2 cache όπου βλέπουμε ότι δεν υπάρχουν μεγάλες αποκλίσεις και οι τιμές κινούνται γύρω στις 25 MPKI.

Στο Σχ. 6.21 φαίνονται οι αστοχίες για την L3 cache όπου βλέπουμε ότι δεν υπάρχουν μεγάλες αποκλίσεις ενώ υπήρχε πίεση από τα διάφορα microbenchmarks με εξαίρεση



Σχήμα 6.21: L3 MPKI



Σχήμα 6.22: DTLB MPKI

το memory bandwidth όπου φαίνεται ξαφνική άνοδος από τη 1 MPKI στις 3.

Στο Σχ. 6.22 φαίνονται οι αστοχίες ανά 1000 εντολές για την DTLB cache όπου βλέπουμε τις αστοχίες να κινούνται από 2 έως 3 στις διάφορες περιπτώσεις που μετρήθηκε η τιμή αυτή.

Σε αυτήν την ενότητα φαίνεται ότι τα microbenchmarks δεν επηρεάζουν τις τιμές των αστοχιών στις caches με εξαίρεση την l3 cache που φαίνεται να αυξάνονται τα misses με το memBw stress test.

6.3.4 Tail Latency σύγκριση με IPC

Σε αυτή την υποενότητα θα παρουσίαστεί το tail latency που μετρήθηκε για όλα τα microservices και στη συνέχεια θα συγκριθούν τα αποτελέσματα με αυτά του IPC.

Όπως βλέπουμε στα σχήματα 6.23 με 6.30, υπάρχουν σε όλα τα γραφήματα κάποια spikes στις τιμές του tail latency. Ιδιαίτερο ενδιαφέρον παρουσιάζει το γεγονός ότι έχουμε υψηλό tail latency όταν τα microservices έτρεχαν σε απομώνωση χωρίς κάποιο microbenchmark να έχει τοποθετεί στον κόμβο. Στις περιπτώσεις των recommendation, rate, geolocation αποτελεί και τη μέγιστη τιμή που φτάνουν σε όλα τα τρεξίματα.

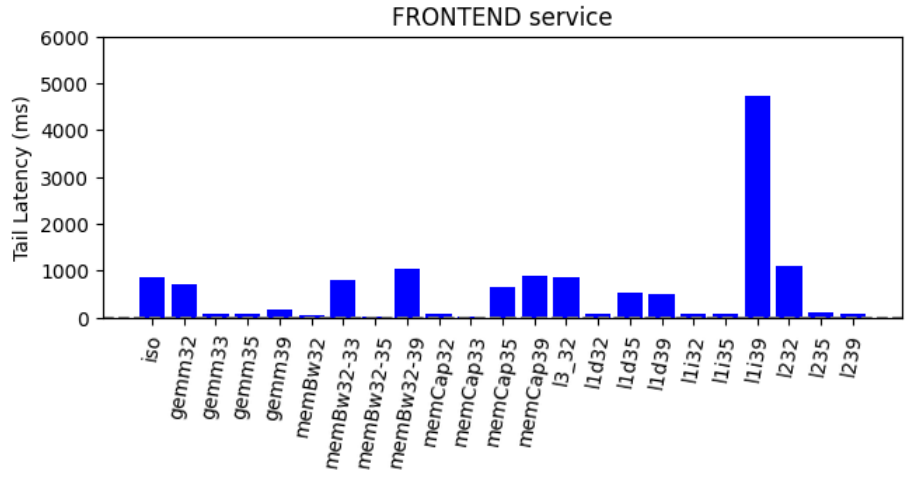
Σε αντίθεση με το IPC το οποίο και έπιανε τις χαμηλότερες τιμές του όταν βρισκόταν ο κόμβος υπό την πίεση του memory bandwidth stress test, το tail latency έχει από τις χαμηλότερες τιμές ακριβώς όταν εισάγουμε αυτό το stressing microbenchmark. Ενώ τα stress tests που φαίνεται σε όλα σχεδόν τα γραφήματα να ανεβάζουν το latency είναι τα memory capacity, L1I-cache, L3-cache. Εξαίρεση σε αυτό αποτελεί το frontend microservice όπου επειδικνυει χαμηλές τιμές tail latency σε όλες τις περιπτώσεις εκτός του L1i-cache.

Αν εδώ αναλογιστούμε ότι το frontend αποτελεί την πιο δαπαναρή σε χρήση νημάτων μικροϋπηρεσία, καθώς χρειάζεται 6 hardware threads ώστε να μην μειωθεί η απόδοσης της λόγω έλλειψης πόρων, τότε θα οδηγηθούμε στο συμπέρασμα ότι αν επιδιώκουμε ένα καλύτερο placement της εφαρμογής στα διαθέσιμα cores τότε σίγουρα αξίζει να ερευνήσουμε τη μετακίνηση αυτού του microservice στα κρίσιμα σε συνδυασμό με το recommendation microservice το οποίο έχει ιδιαίτερα χαμηλό tail latency.

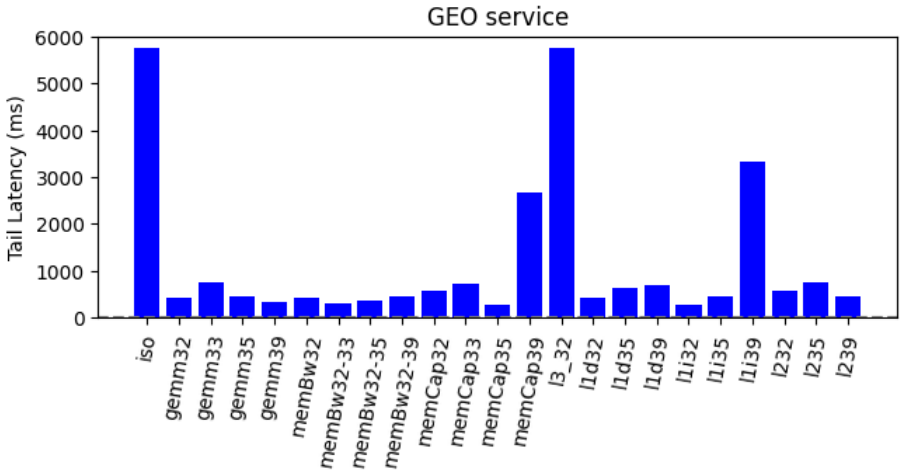
6.3.5 Έλεγχος για βελτίωση του latency

Σε αυτή την υποενότητα θα αξιοποιήσουμε τα δεδομένα και την ανάλυση που έγινε σε αυτά στις προηγούμενες ενότητες με στόχο να βρεθεί ένα καλύτερο placement των μικροϋπηρεσιών στους διαθέσιμους επεξεργαστές του μηχανήματος που έγιναν οι μετρήσεις με σκοπό να επιτευχθεί καλύτερο latency και tail latency που είναι και από τα βασικά ζητούμενα στο optimazation εφαρμογών για καλύτερη εμπειρία χρηστών.

Το latency και tail latency παρουσιάζονται στα Σχήματα 6.31, 6.32 όπου βλέπουμε τα ms για κάθε σενάριο που μετρήθηκε. Η τοποθέτηση των σεναρίων φαίνεται στον Πίνακα 6.3. Όπου βλέπουμε το όνομα του σεναρίου και ποια microsevices τοποθετήθηκαν στο



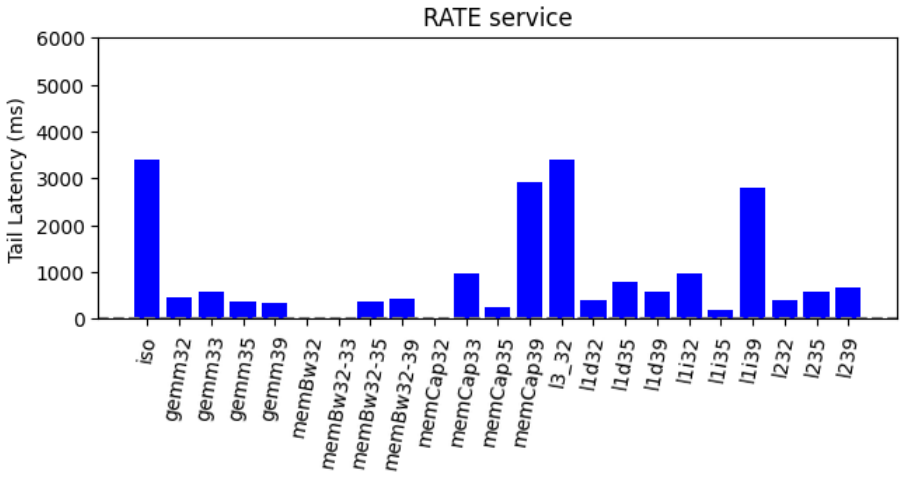
Σχήμα 6.23: Tail Latency Frontend



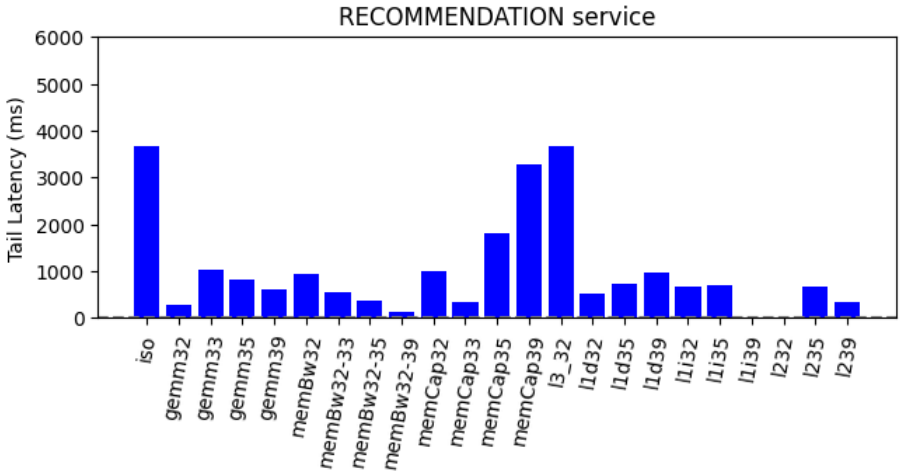
Σχήμα 6.24: Tail Latency Geo

node 1 και ποια στο node 2, σε παρένθεση φαίνονται τα thread που αντιστοιχούν στα συγκρεκιμένα nodes.

Τα σενάρια αποτελούνται για αρχή από το colocated όπου τα microservices έτρεξαν σε 16 cpus χωρίς κανένα περιορισμό. Στα υπόλοιπα σενάρια εφαρμόζεται η λογική του να χωριστούν οι επεξεργαστές σε 2 κατηγορίες, τους γενικούς επεξεργαστές που θεωρήθηκαν οι πρώτοι 8 επεξεργαστές, και τους επεξεργαστές απόδοσης όπου τοποθετήθηκαν οι υπο εξέταση 'critical' εφαρμογές και αποτελούνταν από τους υπόλοιπους

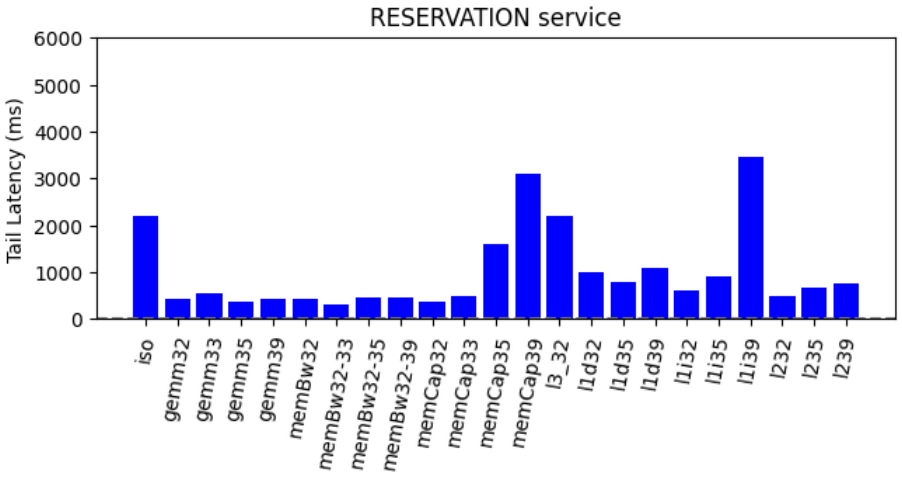


Σχήμα 6.25: Tail Latency Rate

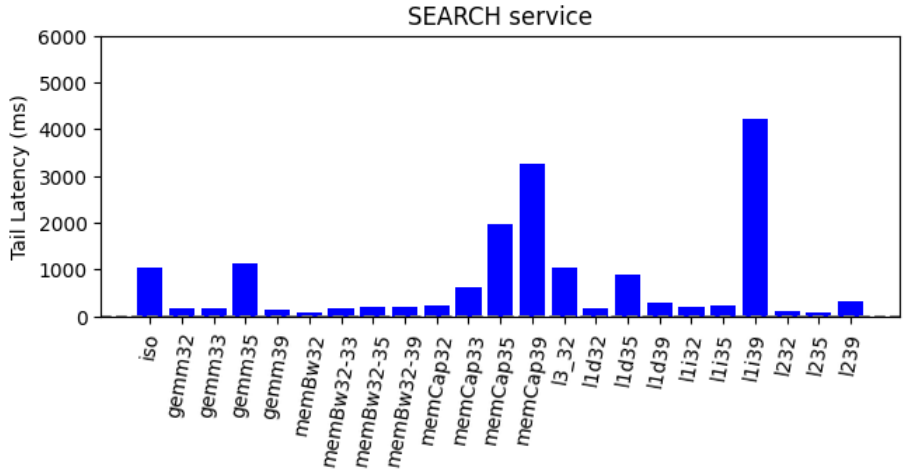


Σχήμα 6.26: Tail Latency Recommendation

8 επεξεργαστές. Στο δεύτερο σενάριο, λοιπόν, που βλέπουμε με όνομα knapsack, επιλέχθηκαν οι εφαρμογές που θα τρέξουν στους performance cores με βάσικο κριτήριο το IPC. Συγκρεκριμένα υπολογίστηκαν αρχικά 7 σκορ για κάθε microservice, όπου 1 σκορ αντιστοιχεί σε συνδυσμό του microservice και stress test (έχουμε 7 stress test στην περίπτωση μας). Ο τρόπος που εξάγεται ένα σκορ θα αναλυθεί με ένα παράδειγμα. Για το microservice geolocation και το microbenchmark gemm υπολογίστηκε 1 σκορ για κάθε φορά που έτρεξε με διαφορετικό αριθμό instances το microbenchmark gemm δηλαδή 4 φορές. Από αυτές τις 4 μετρήσεις βγήκε ο σταθμισμένος μέσος, δίνοντας



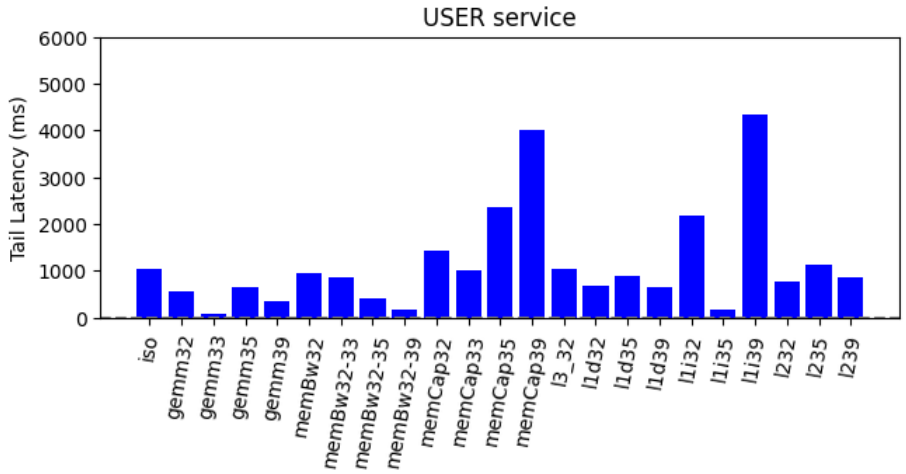
Σχήμα 6.27: Tail Latency Reservation



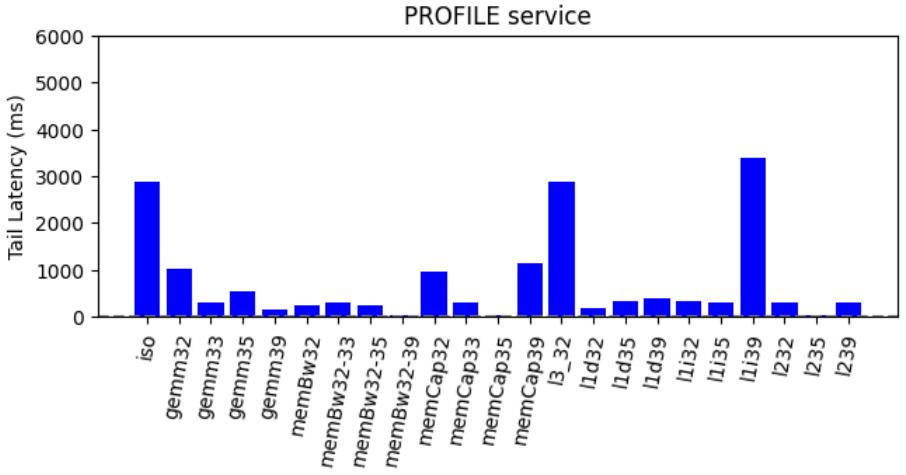
Σχήμα 6.28: Tail Latency Search

το μεγαλύτερο βάρος στην περίπτωση που το microbechmark στρέσαρε με τα μέγιστα διαθέσιμα instances στην περίπτωση μας 8 threads.

Στη συνέχεια από τις 7 αυτές τιμές εξάχθηκε το τελικό σκορ που ήταν 1 για κάθε microservice ξανά με τη τεχνική του σταθμισμένου μέσου, όπου εδώ τα βάρη δώθηκαν εμπειρικά με βάση ποια microbenchmarks έριχναν περισσότερο την IPC των microservices και προφανώς το μεγαλύτερο βάρος δώθηκε στο memory bandwidth που σε όλα τα microservices παρατηρήθηκε μεγάλη πτώση στις τιμές του IPC σε αυτήν την περίπτωση. Τέλος στους παραπάνω υπολογισμούς λήφθηκε υπόψη η τιμή του IPC καθώς



Σχήμα 6.29: Tail Latency User



Σχήμα 6.30: Tail Latency Profile

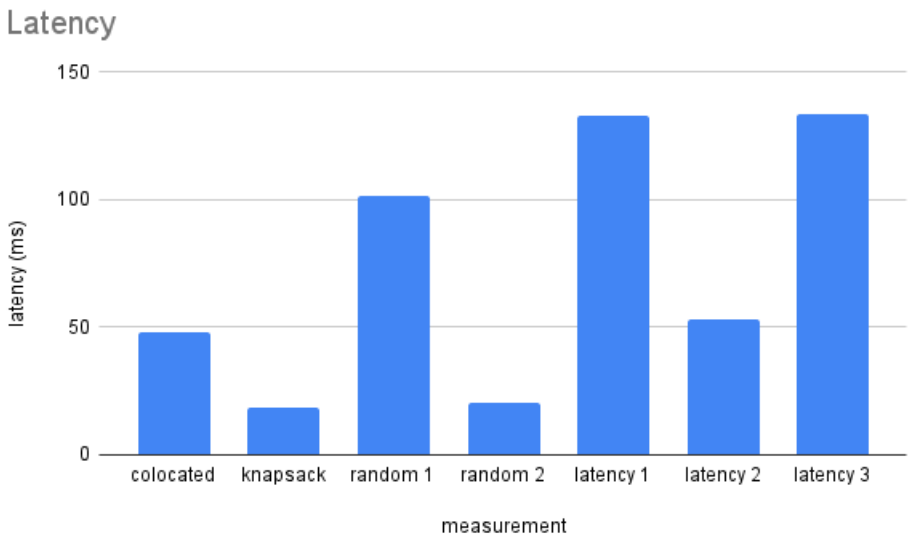
και του ρυθμού πτώσης του IPC στο scope ενός microbenchmark. Έχοντας εξάγει λοιπόν ένα value για κάθε microservice και γνωρίζοντας τις μέγιστες υπολογιστηκές απαιτήσεις του καθενός, έγινε χρήση του κλασσικού αλγόριθμου knapsack 0-1 όπου το βάρος της τσάντας αντιστοιχήθηκε στο μέγεθος των διαθέσιμων performance cores και το βάρος των αντικειμένων και η αξία τους, στις μέγιστες υπολογιστηκές απαιτήσεις του και στην υπολογισθήσα τιμή που αναφέρθηκε προηγουμένως αντίστοιχα. Το αποτέλεσμα του αλγορίθμου ήταν όλα τα microservices να πάνε στα performance cores, εκτός του frontend το οποίο και κρατήθηκε μαζί με τα containers που αποτελούν τις

βάσεις δεδομένων και τις caches των services. Στο τρίτο και τέταρτο σενάριο έγιναν τυχαίες μικρόαλλαγες των microservices μεταξύ genaral και performance cores προσπαθώντας πάντα να μην ξεπεραστεί το μέγεθος των cpus. Στο random1 σενάριο ανταλλάχθηκαν τα search, profile με το frontend όπως είχαν τοποθετηθεί στο knapsack σενάριο. Στο random2 έγινε ανταλλαγή του frontend με τα rate, reservation. Και στα σενάρια latency 1, latency 2 και latency 3 μετρήθηκαν 3 περιπτώσεις placement που έγιναν εμπειρικά με βάση τα αποτελέσματα στο tail latency που είδαμε στις προηγούμενες ενότητες και πάντα με γνώμονα τους διαθέσιμους υπολόγιστικούς πόρους. Η βάση για τις τελεύταιες 3 μετρήσεις αποτέλεσε την παρατήρηση που έχει γίνει σε άλλα papers σχετικά με την ανταλλαξιμότητα των resources (Πηγή [2]), απλώς στην δική μας περίπτωση τα περισσότερα microservices έδειξαν ευαισθησία στην πίεση των ίδιων resources με ελάχιστες εξαιρέσεις όπως αυτή του recommendation όπου δεν αυξανόταν το latency του στην πίεση της L1I cache. (Εδώ να σημειωθεί ότι το L1I benchmark είχε ως βασική διαδικασία την ύψωση αριθμών σε δυνάμεις οπότε αυτό που πρέπει να αναλογιστούμε είναι ότι στρέσαρε ταυτόχρονα το backend του pipeline.)

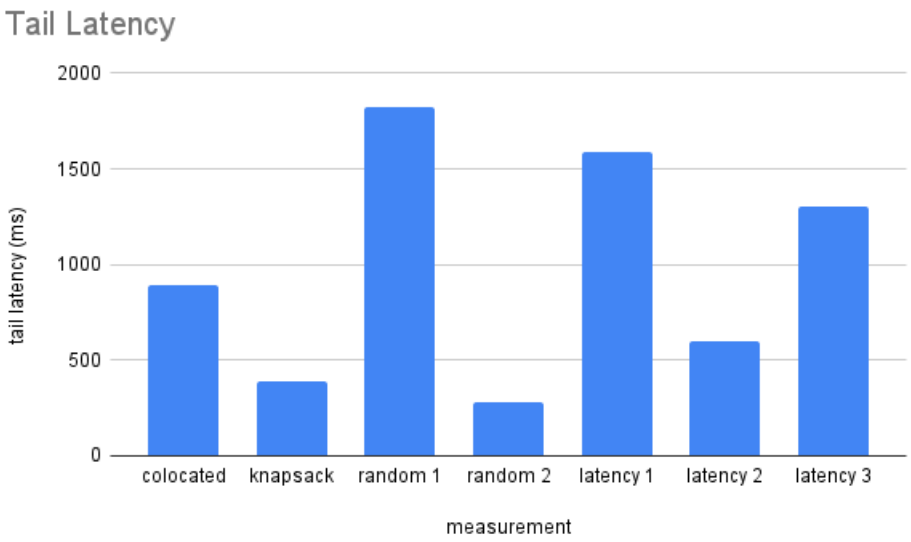
Στον Πίνακα 6.3 φαίνονται οι τοποθετήσεις στα threads του μηχανήματος sandman για τα σενάρια που αναλύθηκαν προηγουμένως.

Έχοντας αναλύσει τι αντιπροσωπεύει κάθε σενάριο μπορούμε να συγκρίνουμε τώρα τα αποτελέσματα των γραφημάτων. Οι περιπτώσεις του knapsack και του random2 φαίνεται να δίνουν τα καλύτερα αποτελέσματα και σε latency και σε tail latency, έχοντας λιγότερο από 30 ms latency. Αντιθέτως οι περιπτώσεις random1, latency1, latency2, latency3 φαίνονται να μην βελτιστοποιούν το baseline σεναρίο που είναι όλα τα microservices να είναι colocated χωρίς περιορισμούς, με τα random1,latency1,latency3 να έχουν και διπλάσιες τιμές από το colocated.

Το IPC είναι μία μετρική που μπορεί να δώσει μία καλή εικόνα για την απόδοση των επεξεργαστών, φαίνεται το σενάριο με τη χρήση του αλγορίθμου του knapsack να βελτιστοποιεί το baseline σενάριο καθώς και το ίδιο επιτεύχθηκε στην τύχη με το σενάριο random2. Η ανάλυση του tail latency σε συνδυασμό με το ότι οι πόροι είναι ανταλλάξιμοι δεν φαίνεται να πετυχαίνει στην περίπτωση μας γιατί όλα τα microservices δείχνουν ευαισθησία σε παρόμοια resources. Μεγάλη αξία στην παραπάνω ανάλυση της εφαρμογής δίνουν και άλλες μετρικές που χρησιμοποιήθηκαν, όπως το cpu usage και το memory usage μας δείνει μια καλή εικόνα της απόδοσης της εφαρμογής απαιτώντας λιγότερο χρόνο για την συλλογή των δεδομένων.



Σχήμα 6.31: Latency



Σχήμα 6.32: Tail Latency

Measurement Name	Node 1 (8-15,40-47)	Node 2 (16-23,48-55)
Colocation	all	all
Knapsack	memcached, mongodb, frontend	geo, rate, user, search, profile, recommendation, reservation
random1	memcached, mongodb, search, profile	geo, frontend, user, search, profile, recommendation, rate, reservation
random2	memcached, mongodb, rate, reservation	geo, frontend, user, search, profile, recommendation
latency1	memcached, mongodb, frontend, recommendation, profile	geo, rate, user, search, reservation
latency2	memcached, mongodb, frontend, recommendation, profile, rate	geo, user, search, reservation
latency3	memcached, mongodb, frontend, recommendation, profile, reservation	geo, user, search, rate

Πίνακας 6.3: Different scenarios placements

Κεφάλαιο 7

Επίλογος

7.1 Σύνοψη και συμπεράσματα

Εδώ συνοψίζουμε τα αποτελέσματα της διπλωματικής και περιγράφουμε τα συμπεράσματα που προέκυψαν, αρνητικά και θετικά. Επιβεβαιώνουμε την συνεισφορά της διπλωματικής στα προβλήματα που αναφέραμε στην εισαγωγή.

Στην παρούσα διπλωματική εργασία εξετάστηκαν θέματα που αφορούν την επίδοση εφαρμογών σχεδιασμένων με την αρχιτεκτονική μικροϋπηρεσιων υπό την σκοπία της μικροαρχιτεκτονικής του συστήματος που είναι εγκατεστημένες. Δείξαμε τη δυσκολία για τη μελέτη και συλλογή των μετρητών απόδοσης του επεξεργαστή, και την ανάγκη δημιουργίας benchmarks ανά περίπτωση. Χρησιμοποιήθηκαν load generator και stress tests για να προσομοιώσουν δύσκολες συνθήκες και να μετρηθεί η συμπεριφόρα των containers υπό πίεση. Δοκιμάστηκε η χρήση ως μετρικής για αξιολόγηση το IPC και το tail latency με τη μέθοδο που εφαρμόστηκε με το IPC να καταφέρνουμε βελτίωση των αποτελεσμάτων του baseline case. Με τις γραφικές παραστάσεις παρατηρήθηκε ότι έχοντας ως μετρική αξιολόγησης το IPC τα microservices έδειξαν όλα ευαισθησία στο memory bandwidth ενώ αντιθέτως έχοντας ως μετρική αξιολόγησης το tail latency φάνηκε ευαισθησία κατά βάση στην πίεση της L1I cache και της L3 cache. Με την topdown ανάλυση είδαμε ότι το μεγαλύτερο μέρος των κύκλων στο pipeline του επεξεργαστή σπαταλήθηκε στο frontend μέρος και με περαιτέρω εξερεύνηση βρέθηκε ότι το bottleneck του frontend είναι η κατηγορία branch reesters ενώ του backend είναι η κατηγορία ports utilization. Τέλος κάνοντας active benchmarking παρατηρήθηκε

ότι το αίτημα για κράτηση δημιουργεί μεγάλη κίνηση στη mongodb του reservation microservice οπότε και αυτό αποτελεί ένα μεγάλο τομέα για optimazations. Καθώς και ότι το εργαλείο jaegar tracing δημιουργούσε θόρυβο στις μετρήσεις μετά από κάποια ώρα λειτουργίας του, που το καθιστούσε δύσκολο στο να διαγνωσθεί.

Συνολικά η διπλωματική έδειξε ότι με βάση το πως θα τοποθετηθούν οι μικροϋπηρεσίες στους διαθέσιμους πυρήνες επηρεάζεται σημαντικά η απόδοση της συνολικής εφαρμογής και υπάρχει μεγάλος χώρος για περαιτέρω βελτιστοποιήσεις. Ενώ η μέθοδος που εφαρμόστηκε με το αλγόριθμο knapsack φαίνεται να βελτίωσε ιδιαίτερα την απόδοση της εφαρμογής ως προς το latency και το tail latency,

7.2 Μελλοντικές επεκτάσεις

Η παραπάνω ανάλυση αποτελεί μία καλή βάση για περαιτέρω ανάλυση και έρευνα της σουίτας DeathStarBench, με βάση τη συγκεκριμένη μεθοδολογία και τις παρατηρήσεις μπορεί να γίνει ακριβώς η ίδια μελέτη και για τις εφαρμογές social network και media microservice ώστε να αποδειχτεί ότι έχουμε τα ίδια αποτελέσματα και στις άλλες εφαρμογές. Στη συνέχεια με βάση τα δεδομένα που μετρήθηκαν θα μπορούσε να γίνει κάποιο δυναμικό placement με machine learning το οποίο θα μπορεί να εφαρμοστεί online. Ακομά θα μπορούσε να εξεταστεί η εφαρμογή hotel reservation υπό το πρίσμα να γίνουν optimazations και μετά να επανεξεταστεί η ευαισθησία των εφαμοργών στα κοινά resources.

Βιβλιογραφία

[1] Brendan Gregg. Active Benchmarking. Iν *IEEE-Compliant: B. Gregg, "Active Benchmarking."*. IEEE, 2014.

[2] Chen, Shuang and Delimitrou, Christina and Martínez, José F. Parties: Qos-aware resource partitioning for multiple interactive services. Iν *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[3] Delimitrou, Christina and Kozyrakis, Christos. ibench: Quantifying interference for datacenter applications. Iν *2013 IEEE international symposium on workload characterization (IISWC)*. IEEE, 2013.

[4] Docker, Inc. Docker containers. . `https://www.docker.com/`.

[5] Eliot Horowitz, Dwight Merriman. MongoDB. `https://www.mongodb.com`.

[6] Ferdman, Michael and Adileh, Almutaz and Kocberber, Onur and Volos, Stavros and Alisafaee, Mohammad and Jevdjic, Djordje and Kaynak, Cansu and Popescu, Adrian Daniel and Ailamaki, Anastasia and Falsafi, Babak. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. *Acm sigplan notices*, 47(4), 2012.

[7] Gan, Yu and Zhang, Yanqi and Cheng, Dailun and Shetty, Ankitha and Rathi, Priyal and Katarki, Nayan and Bruno, Ariana and Hu, Justin and Ritchken, Brian and Jackson, Brendon and others. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. Iν *Proceedings of the Twenty-Fourth International Conference on Ar-*

chitectural Support for Programming Languages and Operating Systems, παγες 3–18, 2019.

[8] Gregg, Brendan. Flamegraphs. . `https://www.brendangregg.com/flamegraphs.html`.

[9] Harshad Sane. Benchmarking for Better Performance Predictions.

[10] Intel. Intel Top-Down Microarchitecture Analysis Method. . `https://www.intel.com/content/www/us/en/docs/vtune-profiler/cookbook/2023-1/top-down-microarchitecture-analysis-method.html`.

[11] Intel. Intel VTune Amplifier. 2012.

[12] Intel, Intel. and IA-32 architectures software developer's manual. *Volume 3B: System Programming Guide, Part 2*, 1(64), 64.

[13] Intel performance events. https://perfmon-events.intel.com/.

[14] Kleen, Andi. Toplev. . `https://github.com/andikleen/pmu-tools`.

[15] Linux Tool. Perf Wiki. Iν *https://perf.wiki.kernel.org/index.php/MainPage IEEE-Compliant: "Kernel.org Performance Wiki." Available: https://perf.wiki.kernel.org/index.php/MainPage*. IEEE, 2014.

[16] Mehmet Ozkaya. Microservices Architecture for Enterprise Large-Scaled Application. https://medium.com/design-microservices-architecture-with-patterns/microservices-architecture-for-enterprise-large-scaled-application-825436c9a78a.

[17] Mrad Fitzpatrick. Memcached. `https://memcached.org/`.

[18] Nikela Papadopoulou. CPU stress gemm. . `https://github.com/nikela/CPUstress/tree/scirouter`.

[19] Uber Technologies. Jaeger. . `https://www.jaegertracing.io/`.

[20] Will Glozer. Wrk2 Suite. `https://github.com/giltene/wrk2`.

[21] Yasin, Ahmad. A top-down method for performance analysis and counters architecture. Iν *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2014.

Γλωσσάριο

Ελληνικός όρος	Αγγλικός όρος
μικροϋπηρεσία	microservice
σωλήνωση	pipeline
κρυφή μνήμη	cache
δειγματοληψία	sampling
μέτρηση	counting
μπροστά μέρος	frontend
πίσω μέρος	backend
εύρος ζώνης	bandwidth
μέγεθος	capacity
νήμα	thread
πολυπλεξία	multiplexing
νεύφος	cloud
σημείο αναφοράς	benchmark
συνάθροιση	aggregation
βελτιστοποίηση	optimazation
φιλτράρισμα	filtering
εντολές ανά κύκλους	instruction per cycles
γεννήτρια φόρτου εργασίας	load generator
δοκιμή καταπόνησης	stressing test