Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Επικοινωνιων, Ηλεκτρονικης και Συστηματων Πληροφορικης

Δυναμική Διαχείριση Πόρων σε Συνθήκες Παρεμβολών με τεχνικές Μηχανικής Μάθησης σε **Kubernetes**

Διπλωματικη Εργασια

του

Γεωργακόπουλου Γεώργιου

Επιβλέπων: Συμεών Παπαβασιλείου
Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2025

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής
Network Management and Optimal Design Laboratory

Δυναμική Διαχείριση Πόρων σε Συνθήκες Παρεμβολών με τεχνικές Μηχανικής Μάθησης σε **Kubernetes**

Διπλωματικη Εργασια

του

Γεωργακόπουλου Γεώργιου

Επιβλέπων: Συμεών Παπαβασιλείου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 19η Σεπτεμβρίου, 2024.

.....	.....	.....
Συμεών Παπαβασιλείου	Ιωάννα Ρουσσάκη	Γεώργιος Ματσόπουλος
Καθηγητής Ε.Μ.Π.	Αν. Καθηγήτρια Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2025

.....

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

Περίληψη

Η ραγδαία ανάπτυξη εφαρμογών υπολογιστικού νέφους έχει οδηγήσει σε εκρηκτική αύξηση των μικροϋπηρεσιών, οι οποίες χρησιμοποιούνται ευρέως τόσο από οργανισμούς όσο και από ιδιώτες και βασίζονται σε τεχνολογίες container. Πολλές από αυτές είναι εφαρμογές με αυστηρές απαιτήσεις καθυστέρησης (Latency Critical), όπως web servers και συστήματα ηλεκτρονικών συναλλαγών, όπου ακόμη και μικρές καθυστερήσεις επηρεάζουν άμεσα την εμπειρία του τελικού χρήστη. Για τη διαχείριση του μεγάλου όγκου container, τα σύγχρονα κέντρα δεδομένων αξιοποιούν μηχανισμούς ορχήστρωσης όπως το Kubernetes, σε συνδυασμό με πρακτικές συντοποθέτησης. Οι πρακτικές αυτές, αν και απαραίτητες για την κλιμάκωση, οδηγούν σε ανταγωνισμό για κοινόχρηστους πόρους του υλικού, όπως CPU και κρυφή μνήμη, με αποτέλεσμα την εμφάνιση του φαινομένου της παρεμβολής και την μείωση της απόδοσης. Ο προεπιλεγμένος scheduler του Kubernetes δεν λαμβάνει υπόψη αυτό το φαινόμενο, και στηρίζει τις αποφάσεις του αποκλειστικά στη χρήση CPU και μνήμης.

Στη συγκεκριμένη εργασία προτείνεται μία interference aware αρχιτεκτονική ελέγχου, η οποία ανακατανέμει δυναμικά τα αντίγραφα εφαρμογών στους κόμβους του cluster σε πραγματικό χρόνο. Αρχικά πραγματοποιείται ανάλυση του φαινομένου της παρεμβολής μέσω πειραμάτων, αναδεικνύοντας τη μη γραμμική και πολύπλοκη φύση του. Στη συνέχεια, με αξιοποίηση μετρικών υλικού από το εργαλείο παρακολούθησης Intel PCM, εκπαιδεύεται ένα μοντέλο XGBoost που επιτυγχάνει διασταυρωμένη μέση απόλυτη απόκλιση **CV_MAE=0.117** στην πρόβλεψη της κανονικοποιημένης απόδοσης υπό διαφορετικά σενάρια παρεμβολής και εισερχόμενου ρυθμού αιτημάτων, ενώ πετυχαίνει επίσης $\mathbf{R^2 = 0.90}$.

Η πειραματική αξιολόγηση σε σενάρια δυναμικής παρεμβολής καταδεικνύει ότι η υλοποίηση μας, με όνομα Marla, υπερτερεί έναντι του προκαθορισμένου scheduler του Kubernetes, μειώνοντας τη μέση καθυστέρηση p_{99} κατά **17.8%** κατά μέσο όρο σε ρεαλιστικά μικτά σενάρια παρεμβολής και έως **34.9%**, ανάλογα με το είδος της παρεμβολής, ενώ παραλληλα οδηγεί το σύστημα σε καλύτερη αξιοποίηση των διαθέσιμων πόρων. Τα αποτελέσματα αποδεικνύουν ότι η ενσωμάτωση μοντέλων πρόβλεψης στο επίπεδο ελέγχου αποτελεί μια ρεαλιστική λύση για την διατήρηση της ποιότητας υπηρεσίας (QoS) χωρίς την δέσμευση περισσότερων πόρων από ότι είναι απαραίτητο.

Λέξεις-κλειδιά— Υπολογιστικό Νέφος, Kubernetes, Παρεμβολές Απόδοσης, Ανταγωνισμός Πόρων, Scheduling, Τοποθέτηση Replicas, Latency-Critical Εφαρμογές, Μηχανική Μάθηση, Αξιοποίηση Πόρων.

Abstract

The rapid growth of cloud computing applications by both businesses and individual users has led to a massive increase in microservices that are based on containerized architectures. Many of these microservices consist of latency-critical (LC) applications, such as Web Servers and online transaction systems, where even small delays can significantly impact user experience. To manage the enormous number of containers, Data Centers rely on orchestrators, such as Kubernetes, and techniques like co-location and multi-tenancy. These practices result in resource contention over shared hardware resources (e.g. CPU, cache, memory bandwidth), severely degrading the quality of a service in comparison to the application running on isolation, a phenomenon called Performance Interference. However, default schedulers make placement decisions based only on static CPU and memory utilization, without being aware of the impact of interference and the dynamic workload of these applications.

In this thesis, we propose an architecture that dynamically redistributes replicas of existing workloads across cluster nodes, while taking into consideration the impact of interference. We first show the effect of applying stress on different shared resources and its impact on the performance of a latency-critical workload. As a representative case study, we focus on the Nginx Web Server, a widely used front-end service in microservice architectures. Using hardware performance counters collected with Intel PCM, we train an XGBoost regression model with cross-validated mean absolute error **CV-MAE=0.117**, capable of predicting performance slowdowns under various traffic and interference conditions. The model also achieves $\mathbf{R^2 = 0.90}$, indicating its explanatory strength. Using this model, we design an interference-aware closed-loop control architecture which adaptively selects replica placement plans across nodes of a cluster in real time. Experimental evaluation in multiple dynamic interference scenarios showed that our implementation, called Marla, outperforms the default Kubernetes scheduler, achieving an average **17.8%** reduction in p_{99} latency in realistic mixed interference scenarios, with reductions going up to **34.9%** depending on the type of interference, while at the same time improving overall resource utilization. These results confirm that interference-aware controllers can meet QoS requirements without relying on costly over-provisioning.

Keywords— Cloud Computing, Kubernetes, Performance Interference, Resource Contention, Scheduling, Replica Placement, Latency Critical Applications, Machine Learning, Resource Utilization.

Ευχαριστίες

Η εκπόνηση αυτής της διπλωματικής εργασίας έγινε υπό την επίβλεψη του καθηγητή του Ε.Μ.Π. κ. Συμεών Παπαβασιλείου, τον οποίο θα ήθελα να ευχαριστήσω θερμά για την ευκαιρία που μου έδωσε να μελετήσω ένα τόσο επίκαιρο και ενδιαφέρον θέμα. Ακόμα θα ήθελα να ευχαριστήσω ιδιαιτέρως τον μεταδιδακτορικό ερευνητή Δημήτρη Σπαθαράκη και τον υποψήφιο διδάκτορα Νίκο Φιλίνη, για τη στήριξη τους σε κάθε βήμα της εργασίας προσφέροντας ουσιαστική βοήθεια οποτεδήποτε χρειαζόταν. Θα ήθελα επίσης να ευχαριστήσω την ομάδα του Anima Cafe για την στηριξη τους καθόλη την ακαδημαική μου πορεία. Τέλος θα ήθελα να ευχαριστήσω απο καρδίας την οικογένεια μου για την αμέριστη συμπαράσταση και την ανιδιοτελή αγάπη τους, καθώς και τους φίλους και τις φίλες μου, οι οποίοι αποτελούν το μεγαλύτερο στηριγμα μου και νιωθω ευγνωμοσύνη που τους έχω δίπλα μου.

Περιεχόμενα

List of Figures

List of Tables

Chapter 1

Εισαγωγή

1.1 Υπολογιστικό Νέφος και Ενορχήστρωση με Kubernetes

Την τελευταία δεκαετία, οι υπηρεσίες του υπολογιστικού νέφους (**Cloud Computing**), αποτελούν θεμέλιο της σύγχρονης ψηφιακής τεχνολογίας, καθώς και αναπόσπαστο κομμάτι της καθημερινότητας, τόσο των απλών χρηστών όσο και των επιχειρήσεων. Με τον όρο Cloud Computing, περιγράφεται το μοντέλο παροχής υπηρεσίων και υπολογιστικών πόρων, όπως αποθηκευτικός χώρο και ισχύς επεξεργασίας, μέσω του διαδικτύου και κατα απαίτηση (on demand). Έτσι, οι τελικοί χρήστες δεν χρειάζεται πλέον να κατέχουν και να διαχειρίζονται φυσικές τοπικές υποδομές.

Η συγχρονη τάση είναι οι επιχειρήσεις, όλων των μεγεθών, να μεταφέρουν εκεί τις υπηρεσίες τους, επιδιώκοντας μείωση του κόστους και αυξηση της ευελιξίας και της αξιοπιστίας. Το υπολογιστικό νέφος παρέχει ταχύτερη ανάπτυξη υπηρεσιών, άμεση επεκτασιμότητα (scalability), απλοποιημένες διαδικασίες συντήρησης οδηγώντας έτσι σε αποτελεσματικότερη διαχείριση των πόρων.

Η αρχιτεκτονική του υπολογιστικού νέφους βασίζεται στο μοντέλο Everything As A Service. Τα κυριότερα μοντέλα, διαχωρίζονται μεταξύ τους με βάση το επίπεδο ελέγχου και αφαίρεσης πάνω στην υποδομή και είναι τα:

- Infrastructure as a Service (IaaS),
- Platform as a Service (PaaS) και
- και Software as a Service (SaaS)

Τα διάφορα μοντέλα ανάπτυξης, επιτρέπουν στους χρήστες του νέφους, επιχειρήσεις ή απλοί χρήστες, να ισορροπούν μεταξύ των επιλογών κόστους, ασφάλειας και ελέγχου. Πλατφόρμες όπως το Amazon Elastic Compute Cloud (EC2) και το Google Cloud Platform (GCP) παρέχουν τη δυνατότητα απομακρυσμένης εκτέλεσης εφαρμογών σε εικονικοποιημένους πόρους, προσφέροντας ευελιξία, οικονομία κλίμακας και αυτόματη κλιμάκωση ανάλογα με το εκάστοτε φορτίο. Η χρήση αυτής της επαναστατκής τεχνολογίας αναπτύσσεται ραγδαία. Σύμφωνα με επίσημα στοιχεία της Eurostat, το 45.2% των επιχειρήσεων στην Ευρωπαϊκή Ένωση χρησιμοποιούσαν υπηρεσίες υπολογιστικού νέφους το 2023, με αύξηση κατά 4.2 μονάδες σε σχέση με το 2021 [1]. Ταυτόγχρονα οι προβλέψεις δείχνουν οτι η ζήτηση για τέτοιες υπηρεσίας, θα συνεχίσει να αυξάνεται ακόμα περισσότερο τα επόμενα χρόνια .

Τεχνολογική βάση για την υλοποίηση του υπολογιστικού νέφους αποτελεί η εικονοποίηση (**Virtualization**). Μέσω της εικονοποίησης, οι υπολογιστικοί πόροι των φυσικών μηχανημάτων (Physical Machines, PMs) διαχωρίζονται σε πολλά σχεδόν απομονωμένα εκτελεστικά περιβάλλοντα (Virtual Machines, VMs), τα οποία διαχειρίζονται απο έναν ελεγκτικό μηχανισμό (hypervisor).

Ωστόσο, η άνοδος των αρχιτεκτονικών μικροϋπηρεσιών (microservices), φανέρωσε την ανάγκη για πιο υπολογιστικά ελαφριές και ταχύτερες μορφές εικονικοποίησης, τα **Containers**. Η τεχνολογία αυτή , αποτέλεσε την φυσική εξέλιξη της εικονοποίησης, παρέχοντας απομόνωση εφαρμογών χωρίς την ανάγκη ύπαρξης ξεχωριστού λειτουργικού συστήματος για κάθε περιβάλλον. Η τεχνολογία των containers έχει χαμηλότερο υπολογιστικό

κόστος, και παρέχει ταχύτερη ανάπτυξη και κλιμάκωση, χαρακτηριστικά που την έκανε καταλληλότερη για την υποστήριξη των απαιτήσεων του σύγχρονου υπολογιστικού νεφους. Αυτό φαίνεται και απο το γεγονός οτι οι πλατφόρμες νέφους επενδύουν περισσότερο σε υποδομές βασισμένες σε containers για την εκτέλεση κρίσιμων υπηρεσιών.

Συμπερασματικά, η ερευνητική προσοχή μετατοπίζεται από την παραδοσιακή εικονικοποίηση με VMs προς την εικονοποίηση με coontainers (containerization). Στο πλαίσιο αυτό, η παρούσα εργασία εστιάζει στην αποδοτική διαχείριση των containers σε περιβάλλοντα του υπολογιστικού νέφους.

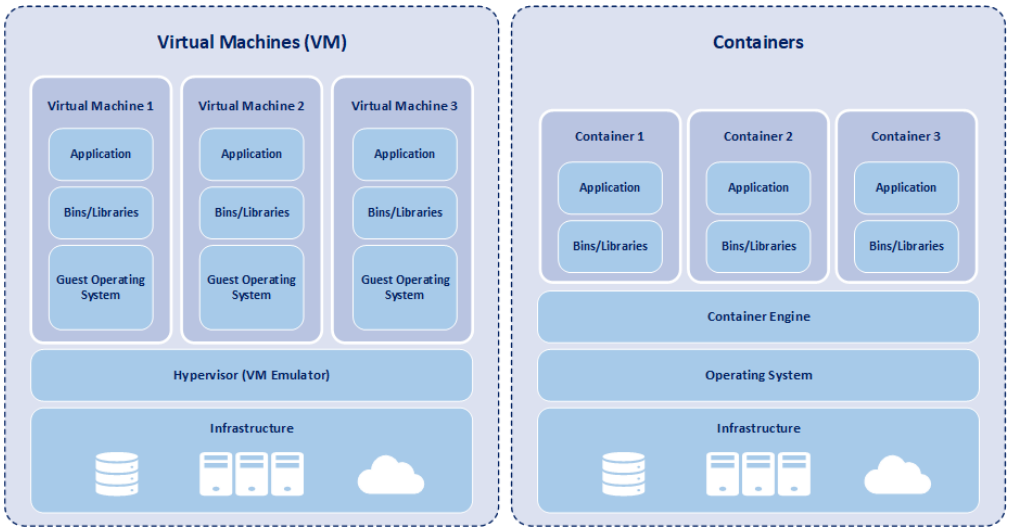

Figure 1.1: Σύγκριση αρχιτεκτονικής Εικονικών Μηχανών (VMs) και Containers.

Σε ένα σύγχρονο κέντρο δεδομένων (Data Center), περιέχονται ταυτόχρονα πολυάριθμα containers, καθιστώντας αναγκαίο ένα σύστημα ενορχήστρωσης τους. Το **Kubernetes** αποτελεί το πιο διαδεδομένο και ευρέως χρησιμοποιούμενο εργαλείο ενορχήστρωσης. Είναι ένα σύστημα ανοιχτού κώδικα το οποίο παρέχει την ομαδοποίηση των κατανεμημένων τμημάτων της εφαρμογής, καθώς και την αυτόματη εκτέλεση, διαχείριση και κλιμάκωση τους. Ενσωματώνει την master-slave αρχιτεκτονική, στην οποία τα κεντρικά τμήματα (master components) διαχειρίζονται την κατάσταση της συστάδας (cluster), ενώ οι κόμβοι εργάτες (worker nodes) εκτελούν τις εφαρμογές. Η μικρότερη μονάδα εκτέλεσης σε αυτό το μοντέλο είναι το **pod** το οποίο αποτελείται απο ένα ή περισσότερα container σε έναν κοινό αποθηκευτικό και δικτυακό χώρο. Ετσι το Kubernetes προσφέρει ένα επίπεδο αφαίρεσης (abstraction layer) το οποίο διευκολύνει την ανάπτυξη συγχρονων containerized εφαρμογών. Η αναλυτική παρουσίαση των επιμέρους δομικών στοιχείων και μηχανισμών του Kubernetes γίνεται στην Ενότητα 2.3.

Ωστόσο, παρά τα πλεονεκτήματα αυτά, ο προεπιλεγμένος μηχανισμός προγραμματισμού των εφαρμογών **default scheduler**, ενα απο τα master components του Kubernetes, λαμβάνει υπόψη κυρίως μετρικές όπως τα ποσοστά χρήσης CPU και μνήμης, χωρίς να εξετάζει πιο σύνθετες παραμέτρους της υποδομής.

1.2 Το πρόβλημα της Παρεμβολής

Η εικονικοποίηση, τόσο σε επίπεδο εικονικών μηχανών (VMs) όσο και σε επίπεδο κιβωτίων (containers), δεν μπορεί να προσφέρει πλήρη απομόνωση πόρων. Ετσι, όταν πολλαπλές εφαρμογές συνυπάρχουν στο ίδιο φυσικό μηχάνημα (multi tenancy), ανταγωνίζονται αναγκαστικά για τους κοινόχρηστους πόρους του υλικού, με αποτέλεσμα η απόδοση να αποκλίνει από αυτήν που θα είχαν αν είχαν εκτελεστεί σε πλήρη απομόνωση πόρων [2]. Αυτή η απόκλιση ονομάζεται **Performance Interference** και συνιστά έναν από τους βασικότερους παράγοντες απρόβλεπτης συμπεριφοράς σε περιβάλλοντα όπου συνυπάρχουν πολλές εφαρμογές.

Η παρεμβολή δημιουργείται από πολλές μορφές ανταγωνισμού στο επίπεδο του υλικού. Το σχήμα 1.2 δείχνει όλες τις μορφές όπως κατηγοριοποιούνται απο τους Lin et al. [3].

Οσον αφορά την παρεμβολή στο επίπεδο της CPU, εξετάζοντας την σύγχρονη αρχιτεκτονική επεξεργαστών, μέσω του σχήματος 1.3, γίνεται αμέσως καταννοητό οτι το φαινόμενο αυτό είναι αναποφευκτο. Αρχικά ο κάθε φυσικός πυρήνας διαθέτει προσωπικές κρυφές μνήμες επιπέδου L1 και L2, ωστόσο μέσω της τεχνικής Hyper-Threading, δύο ξεχωριστά νήματα αναγκαστικά μοιράζονται αυτούς τους πόρους [4]. Παράλληλα, οι φυσικοί πυρήνες που ανήκουν στο ίδιο socket μοιράζονται την κρυφή μνήμη τελευταίου επιπέδου (LLC). Επομένως, η κρυφή μνήμη αποτελεί σημείο συμφόρησης και η διεκδίκηση φυσικών πυρήνων και γραμμών (θέσεων) της κρυφής μνήμης οδηγεί στην αύξηση της μετρικής cache misses και οδηγεί σε καθυστερήσεις εκτέλεσης [5, 6]. Τέλος, η υπερεγγραφή (oversubscription) εικονικών πυρήνων (vCPUs) έχει ως αποτελέσμα την δημιουργία ανταγωνισμού για μερίδια του χρόνου εκτέλεσης, μειώνοντας έτσι την απόδοση.

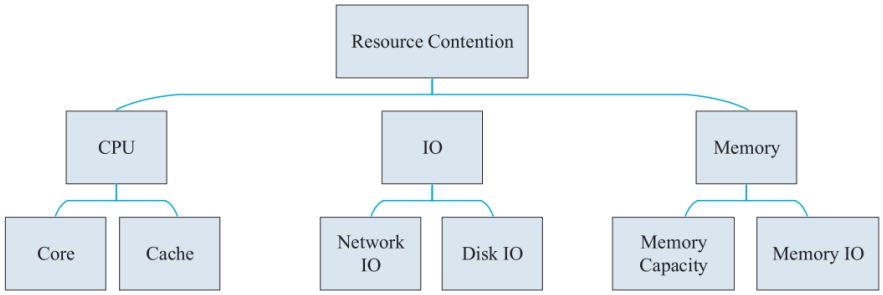

Figure 1.2: Κατηγορίες ανταγωνισμού πόρων που οδηγούν σε παρεμβολή απόδοσης.

Η μνήμη αποτελεί επίσης κρίσιμη πηγή παρεμβολής. Αρχικά, το περιορισμένο εύρος μνήμης (memory bandwidth) αποτελεί κρίσιμη πηγή συμφόρησης, καθώς καθυστερεί τη ροή δεδομένων προς και από τον επεξεργαστή [7, 8]. Επιπλέον, η έλλειψη χωρητικότητας (memory capacity), όπου πολλαπλές εφαρμογές ανταγωνίζονται για περιορισμένο φυσικό RAM και οδηγούνται σε swapping ή throttling. Ο ανταγωνισμός αυτός επιδεινώνεται απο τις συχνές προσβάσεις στη RAM λόγω αυξημένων cache misses, οδηγώντας έτσι στην μείωση της απόδοσης.

Τέλος, η παρεμβολή στις λειτουργίες I/O αφορά κυρίως το δίκτυο και τον δίσκο. Και τα δυο, χρησιμοποιούνται απο όλες τις εφαρμογές, με αποτέλεσμα αυτές που συνυπάρχουν να βλέπουν λιγότερο διαθέσιμο εύρος ζώνης σε σχέση με αυτές όπου εκτελούνται απομονωμένες [9].

Πρέπει να επισημαθεί, ότι η επίδραση αυτών των παραγόντων διαφέρει ανάλογα με τον τύπο της εφαρμογής. Για παράδειγμα, οι CPU–intensive εφαρμογές, δηλαδή οι εφαρμογές οι οποίες χρειάζονται περισσότερο πόρους CPU σε σχέση με άλλες εφαρμογές, πλήττονται κυρίως από τον ανταγωνισμό στον πυρήνα και την κρυφή μνήμη, ενώ I/O–intensive εφαρμογές από την έλλειψη bandwidth [10].

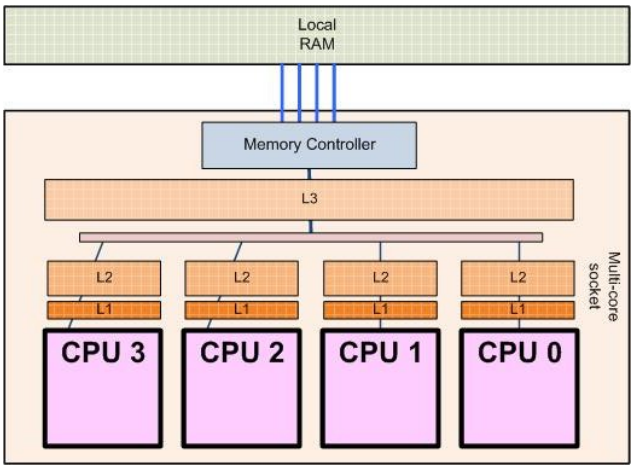

Figure 1.3: Σύγχρονη Αρχιτεκτονική Επεξεργαστών

Η ύπαρξη της παρεμβολής έχει άμεσες συνέπειες στην προβλεψιμότητα και την αξιοπιστία των συστημάτων υπολογιστικού νέφους. Η απόδοσης μια εφαρμογής μπορεί να αποκλίνει σημαντικά απο την θεωρητική της αποδοσης σε απομόνωση, δηλαδή χωρίς να επηρεάζεται απο παρεμβολή, κάνοντας δύσκολη την διατήρηση της ποιότητας υπηρεσίας (**QoS**). Ακόμη και μικρές καθυστερήσεις σε ένα κρίσιμο επίπεδο σε πολυεπίπεδες αρχιτεκτονικές μικροϋπηρεσιών, αρκούν για να επιβραδύνει ολόκληρη η σειρά επεξεργασίας (pipeline), αυξάνοντας αρκετά το κόστος της παρεμβολής. Ενδεικτικά, μετρήσεις σε εμπορικές πλατφόρμες έχουν δείξει ότι η απόδοση σε I/O μπορεί να μειωθεί έως και 50%–60% εξαιτίας παρεμβολών [11, 12], αναδεικνύοντας την σοβαρότητα του φαινομένου και την σημασία διαχείρισης του.

Η παρεμβολή απόδοσης είναι αναπόφευκτο αποτέλεσμα της συντοποθέτησης εφαρμογών και ένας από τους πιο δύσκολους προς διαχείριση παράγοντες σε σύγχρονα container-based clusters. Η διαχείριση της είναι ιδιαίτερα απαιτητική, καθώς πρόκειται για ένα πολύπλοκο και μη γραμμικό φαινόμενο. Κατά συνέπεια, η ανάγκη για κατανόηση και αντιμετώπισή της αποτελούν κρίσιμες προϋποθέσεις για την παροχή αξιόπιστων και προβλέψιμων υπηρεσιών στο υπολογιστικό νέφος.

1.3 Συνεισφορά της Εργασίας

Στην παρούσα εργασία, εξετάζεται αναλυτικά το φαινόμενο της παρεμβολής και πώς επηρεάζει την απόδοση εφαρμογών, με σκοπό τη συμπερίληψή του στις αποφάσεις τοποθέτησης εφαρμογών σε έναν Kubernetes cluster. Συγκεκριμένα, η κύρια συνεισφορά της εργασίας είναι η σχεδίαση και υλοποίηση ενός **Interference Aware Runtime Controller** για το Kubernetes, με την ονομασία **Marla**, το οποίο εστιάζει στη δυναμική ανακατανομή των αντιγράφων μιας υφιστάμενης εφαρμογής σε πολλαπλούς κόμβους ενός cluster. Το Marla αποτελείται από έναν κλειστό βρόχο ελέγχου με συνεχή παρακολούθηση, πρόβλεψη απόδοσης και διορθωτικές παρεμβάσεις σε πραγματικό χρόνο.

Η ανάπτυξη του Marla στηρίχθηκε σε μια μεθοδική ερευνητική πορεία [2, 13, 14]. Αρχικά μελετήθηκε πειραματικά η επίδραση της παρεμβολής στην απόδοση εφαρμογών, η οποία στην συνέχεια ποσοτικοποιήθηκε μέσω του δείκτη **Normalized Performance**. Τα πειράματα εστιάζουν σε εφαρμογές με αυστηρούς περιορισμούς καθυστέρησης **Latency Critical Workloads** (LC), και συγκεκριμένα στον Nginx Web Server, που αποτελεί αντιπροσωπευτικό παράδειγμα τέτοιου είδους εφαρμογής.

Έχοντας συγκεντρώσει ένα μεγάλο σύνολο δεδομένων, τόσο με μετρικές απόδοσης όσο και με μετρικές υλικού που φανερώνουν την παρεμβολή, εκπαιδεύτηκε ένα **Μοντέλο Παλινδρόμησης** ώστε να μπορεί να προβλέπει την υποβάθμιση απόδοσης λόγω παρεμβολής. Με βάση τις δυνατότητες αυτού του μοντέλου σχεδιάστηκε η αρχιτεκτονική του συστήματος ελέγχου και τελικά υλοποιήθηκε ο controller Marla, ο οποίος αξιολογήθηκε πειραματικά σε αντιπαράθεση με τον προεπιλεγμένο μηχανισμό του Kubernetes. Τα αποτελέσματα φανέρωσαν ότι η συμπερίληψη της παρεμβολής στις αποφάσεις τοποθέτησης οδηγεί σε σημαντική βελτίωση της προβλεψιμότητας και της απόδοσης του συστήματος.

1.4 Δομή της Εργασίας

Η παρούσα διπλωματική εργασία είναι οργανωμένη σε πέντε κεφάλαια. Στο Κεφάλαιο 2 παρουσιάζονται εργασίες σχετικά με το αντικείμενο της διπλωματικής. Συγκριμένα παρατίθενται και αναλύονται τεχνικές συλλογής και ερμηνείας μετρικών παρεμβολής καθώς και μέθοδοι κατανομής πόρων στο Kubernetes. Στο Κεφάλαιο 3 περιγράφεται η προτεινόμενη αρχιτεκτονική, η υλοποίηση Marla καθώς και ο αλγόριθμος εύρεσης βέλτιστου πλάνου τοποθέτησης αντιγράφων της εφαρμογής, με βάση την παρεμβολή. Το Κεφάλαιο 4 είναι αφιερωμένο στην πειραματική αξιολόγηση. Αρχικά, παρουσιάζονται τα αποτελέσματα της επίδρασης της παρεμβολής στην απόδοση της εφαρμογής. Στην συνέχεια ακολουθεί η εκπαίδευση του μοντέλου μηχανικής μάθησης και τέλος η πειραματική σύγκριση της προτεινόμενης αρχιτεκτονικής Marla με τον προκαθορισμένο μηχανισμό του Kubernetes. Τέλος, στο Κεφάλαιο 5 εξάγονται τα βασικά πορίσματα της εργασίας και παράλληλα προτείνονται ιδέες και κατευθύνσεις για μελλοντική ερευνα.

Chapter 2

Σχετική Έρευνα

2.1 Συλλογή Μετρικών (Metrics Collection)

Η καταννόηση, η ανάλυση και η πρόβλεψη της παρεμβολής στηρίζεται στην κατάλληλη επιλογή των μετρικών του υλικού. Στην δουλειά των Lin et al. [15], παρουσιάζονται 2 κατηγορίες μετρικών υλικού καθώς και 2 στρατηγικές συλλογής τους.

Κατηγορίες μετρικών:

- **Ανεξάρτητες μετρικές (Independent Metrics)**: Οι μετρικές αυτές συλλέγονται απευθείας από το σύστημα, δηλαδή το φυσικό μηχάνημα (Physical Machine, PM) υποδεικνύοντας την ύπαρξη παρεμβολής. Σχετίζονται με τους κρίσιμους πόρους που καταναλώνουν οι εφαρμογές. Συνήθη παραδείγματα περιλαμβάνουν:
  - CPU utilization, CPU cycles, Instructions Per Cycle (IPC)
  - Cache occupancy και cache misses (L2/L3)
  - Memory bandwidth, TLB misses
  - I/O throughput (read/write)
  - Scheduler's Latency

  Σε κάθε εργασία, η επιλογή των κατάλληλων μετρικών συνδέεται με τον τύπο της εφαρμογής. Για παράδειγμα, στις εφαρμογές με έντονη χρήση του επεξεργαστή (CPU Intensive Workloads), συνήθως επιλέγονται CPU cycles, cache miss rate και vCPU utilization [16].

Η δεύτερη κατηγορία μετρήσεων είναι οι **Παράγωγες Μετρικές (Derived Metrics)**. Αυτές προκύπτουν απο την επεξεργασία μετρικών που σχετίζονται με την απόδοση των εφραμογών, καταφέρνοντας έτσι να αποτυπώνουν καλύτερα την επίδραση της παρεμβολής. Μία από τις πιο διαδεδομένες παράγωγες μετρικές είναι το *Normalized Performance*, το οποίο ορίζεται ως:

$$\text{Normalized Performance} = \frac{\text{Perf}(A, S)}{\text{Perf}(A, 0)}$$

όπου S είναι το σενάριο παρεμβολής και $\text{Perf}(A, 0)$ η απόδοση της εφαρμογής A σε εκτέλεση χωρίς παρεμβολή. Προτάθηκε αρχικά από τους Koh et al. [2] και χρησιμοποιείται σε πολλές μεταγενέστερες εργασίες [17] ως βασικός τρόπος ποσοτικοποίησης της παρεμβολής. Ταυτόγχρονα σε άλλες εργασίες, οι παράγωγες μετρικές αντλούνται απο δείκτες συστήματος χαμηλού επιπέδου, όπως είναι το CPI και το MIPS [18, 19], παρέχοντας μια εκτίμηση της πίεσης στο υλικό και μια εικόνα της υπολογιστικής ικανότητας της εφαρμογής.

Οι στρατηγικές συλλογής αυτών των μετρικών, όπως παρουσιάζονται απο τους Lin et al. [15], στηρίζονται σε 2 βασικές προσσεγγίσεις.

- **Interference Injection**:

  Η πρώτη στρατηγική είναι η δημιουργία τεχνητής πίεσης στο υλικό, χρησιμοποιώντας ελεγχόμενα φορτία ωστε να προκληθεί σκόπιμη παρεμβολή, ταυτόγχρονα με την εκτέλεση της εφαρμογής. Ετσι, για την εξήγηση της παρεμβολής αρκεί η σύγκριση της απόδοσης με σενάρια εκτέλεσης της εφαρμογής χωρίς τεχνητή πίεση. Για τον σκοπό αυτό, χρησιμοποιούνται benchmarks όπως iBench [20], CISBench [21], Cuanta [22] και FECBench [23]. Σε κάποιες εργασίες, αντί της χρήσης εργαλείων δημιουργίας παρεμβολής, η πίεση στο υλικό επιτυγχάνεται με περιορισμό της χωρητικότητας των πόρων του [24].

- **Historical Data**:

  Στην δευτερη στρατηγική αξιοποιούνται μετρήσεις από πραγματικές εκτελέσεις. Σε πολλές εργασίες χρησιμοποιούνται εργαλεία όπως το Intel PCM [25], το Linux perf [26],το cAdvisor [27] και το Prometheus [28] ή system call traces [29]. Με αυτήν την προσέγγιση αποτυπώνεται η ρεαλιστική συμπεριφορά των συστημάτων και των εφαρμογων, ωστόσο απαιτείται μεγάλος όγκος δεδομένων και η ανάλυση του φαινομένου της παρεμβολής γίνεται πιο δυσκολη.

2.2 Κατηγοριοποίηση Παρεμβολών (Interference Classification)

Μια θεμελιώση μέθοδος ερμηνείας της παρεμβολής είναι η μοντελοποίηση των χαρακτηριστικών της. Στην διαδικασία αυτή χρειάζονται οι εννοιολογικές μετρικές *Interference Sensitivity* και *Interference Intensity*, όπως προτάθηκαν από τους Kim et al. [30] και Chen et al. [31].

Η πρώτη δείχνει το πόσο ευάλωτη είναι μια εφαρμογή όταν τοποθετείται μαζί με άλλες, δηλαδή το πόσο εύκολα μπορεί να επηρεαστεί απο αυτές. Η δεύτερη δείχνει τον βαθμό όπου μια εφαρμογή καταναλώνει του πόρους του υλικού, δηλαδή το πόσο μπορεί να επηρεάσει άλλες εφαρμογές που τοποθετούνται μαζί της. Οι δυο έννοιες, αποτελούν ερμηνείες της συμπεριφοράς μιας εφαρμογής λόγω της παρεμβολής, δημιουργώντας ένα ισχυρό πλαίσιο χαρακτηρισμού των εφαρμογών.

Σύμφωνα με το survey των Lin et al. [15], η κατηγοριοποίηση της παρεμβολής ακολουθεί δύο κύριες προσεγγίσεις, οι οποίες εξετάζονται στην συνέχεια.

2.2.1 Ποιοτική Μοντελοποίηση (Qualitative Modeling)

Η ποιοτική μοντελοποίηση έχει ως στόχο την αναγνώριση και την κατηγοριοποίηση της παρεμβολής σε κατηγορίες ή σε διακριτά επίπεδα. Αντί για τον ακριβή υπολογισμό της πτώσης της απόδοσης (performance degradation), η προσέγγιση αυτή χαρακτηρίζει την παρεμβολή ως «χαμηλή», «μέτρια» ή «υψηλή» .Ένας άλλος τρόπος είναι να διαχωρίσει τις εφαργμογές σε διακριτά επίπεδα, και χρησιμοποιώντας τις έννοιες Interference Sensitivity και Interference Intensivity να τις χαρακτηρίσει ως «ευαίσθητες» και «επιθετικές». Με τον τρόπο αυτό παρέχεται μια εκτίμηση της συμπεριφοράς των εφαρμογών ή της παρεμβολής, η οποία μπορεί να αξιοποιηθεί από schedulers για αποφάσεις τοποθέτησης. Η υλοποίηση της ποιοτικής μοντελοποίησης, μπορεί να επιτευχθεί με διαφορετικούς τρόπους.

Οι **Ludwig et al.** [32] προτείνουν την κατασκευή κανόνων βασισμένων σε μετρικές υλικού, μέσω hardware counters, για να ταξινομηθεί ο βαθμός παρεμβολής σε τέσσερις κατηγορίες (Absent, Low, Moderate, High). Δεν χρησιμοποιείται κάποιο μοντέλο μηχανικής μάθησης, αλλά πο αποφάσεις βασίζονται σε σταθερά όρια (thresholds) αναφορικά με μετρικές όπως cache miss rate και CPU usage. Έτσι επιτυγχάνεται χαμηλό κόστος εκτέλεσης και επάρκεια στην ταξινόμηση της παρεμβολής.

Οι **Meyer et al.** .[33] πρότειναν έναν interference aware ταξινομητή για εφαρμογές σε cloud περιβάλλοντα. Αντί για στατικά όρια κατανάλωσης πόρων ο ταξινομητής *προσδιορίζει δυναμικά* τα επίπεδα παρεμβολής, αξιοποιώντας έναν συνδυασμό ομαδοποίησης μέσω K-means για τον εντοπισμό μοτίβων χρήσης και χρήσης Support Vector Machines (SVM) για την ακριβή ταξινόμηση νέων εφαρμογών. Με την μέθοδο αυτή οδηγήθηκαν σε λεπτομερή κατηγοριοποίηση δυναμικών workloads, επιτυγχάνοντας ποσοστά ακρίβειας άνω του 80% σε δείκτες αξιολόγησης όπως Accuracy και F1-score, βελτιώνοντας την αποδοτικότητα τοποθέτησης κατά 23% σε σχέση με προηγούμενες τεχνικές.

Οι **Horchulhack et al.** [34] εστιάζουν στον εντοπισμό της QoS υποβάθμισης σε containerized εφαρμογές, λόγω συντοποθέτησης στο ίδιο υλικό. Η μεθοδολογία βασίζεται στην παρακολούθηση της κατανάλωσης των πόρων σε επίπεδο εφαρμογών, ώστε να ανιχνεύει αποκλίσεις που αντανακλούν φαινόμενα συντοποθέτησης. Στη

συνέχεια, αφου επιλεχθούν τα κατάλληλα χαρακτηριστικά μέσω της τεχνικής feature reduction, εφαρμόζεται κατηγοριοποίση χρησιμοποιώντας ένα μοντέλο LSTM, το οποίο εκπαιδεύεται στο σύνολο χρονοσειρών και εντοπίζει μοτίβα στη συμπεριφορά του συστήματος. Το τελικό αποτέλεσμα είναι μια ποιοτική ερμηνεία της κατάσταση της κάθε εφαρμογής με δύο επίπεδα: *«QoS normal»* και *«QoS degraded»*. Τα αποτελέσματα της πειραματικής αξιολόγησης ήταν: True Positive Rate 90%, False Positive Rate 8% και F1-score έως 0.94, αποδεικνύοντας έτσι ότι η κατηγοριοποίηση μέσω χρονικών μοντέλων μπορεί να προσφέρει αξιόπιστη ανίχνευση της παρεμβολής.

Τελος, οι **Băluță et al.** [35] υλοποιούν ένα pipeline το οποίο βασίζεται αποκλειστικά σε μετρικές επιπέδου εφαρμογής, όπως response time, latency και throughput και υλοποιείται με χρήση εποπτευόμενης μάθησης και μηχανισμό ολίσθησης (sliding window), ώστε να υπάρχει δυναμική προσαρμογή. Το τελικό αποτέλεσμα είναι μια δυαδική κατηγοριοποίηση, *interference vs. no interference*, παρέχοντας έναν αξιόπιστο τρόπο ανίχνευσης παρεμβολής χωρίς την χρήση μετρικών υλικού.

Γίνεται αντιληπτό ότι οι τεχνικές ταξινόμησης και κατηγοριοποίησης μπορούν να συμβάλλουν σημαντικά στην ανίχνευση της παρεμβολής και την μείωση της επίδρασης της χωρις να είναι αναγκαστική η πραγματική της μέτρηση. Η χρήση τέτοιων τεχνικών είναι ιδιαίτερη σημαντική ειδικά σε περιπτώσεις όπου δεν υπάρχει πρόσβαση στις μετρικές του υλικού ή το κόστος του υπολογισμού της πραγματικής μέτρησης είναι απαγορευτικά υψηλό.

2.2.2 Ποσοτική Μοντελοποίηση (Quantitative Modeling)

Η ποσοτική μοντελοποίηση δεν περιορίζεται στην απλή διάγνωση ή κατηγοριοποίηση της παρεμβολής, αλλά στοχεύει στον αριθμητικό προσδιορισμό του κόστους συντοποθέτησης. Αναλύοντας την σχέση των μετρήσεων του υλικού με τους δείκτες απόδοσης, όπως normalized performance, η τεχνική αυτή μπορεί να παράξει εκτιμήσεις της υποβάθμισης της απόδοσης των εφαρμογών. Παρέχει δηλαδή την ικανότητα στον scheduler να υπολογίσει το ρίσκο μιας απόφασης τοποθέτησης πριν την εφαρμογή της.

Το **Rusty** των Masouros et al.'[36] αποτελεί ένα runtime predictive monitoring σύστημα που τροφοδοτεί ένα μοντέλο LSTM με τις χρονοσειρές των hardware counters. Το μοντέλο αυτό εκπαιδεύεται να προβλέπει την αναμενόμενη υποβάθμιση της απόδοσης. Η λειτουργία του μοντέλου βασίζεται τα μοτίβα των hardware counters, μέσω των οποίων μπορούν να κωδικοποιηθεί η εξέλιξη της παρεμβολής, επιτρέποντας στον ελεγχτή να παρεμβαίνει έγκαιρα πριν εμφανιστούν SLO παραβιάσεις και πτώσεις του QoS.

Οι **Buchaca et al.** [37] χρησιμοποιούν ένα νευρωνικό δίκτυο sequence-to-sequence RNN για να προβλέψει την αλληλεπίδραση των εργασιών που συντοποθετούνται σε batch data centers. Βασισμένο σε προφίλ μεμονωμένων εργασιών το μοντέλο προβλέπει την κατάσταση της χρήσης πόρων όταν οι εργασίες συντοποθετηθούν. Ετσι, ο scheduler χρησιμοποιεί τα forecast που παράγονται για να αποφύγει τις πιο βλαβερές για τους πόρους συντοποθετήσεις.

Στο **dCCPI-predictor** οι Li et al.'[38] εισάγουν μια μορφή γραμμικής παλινδρόμησης με state aware χαρακτηριστικά. Σκοπός του μοντέλου είναι ανά να αναδείξει την παρεμβολή ανάμεσα στους πυρήνες του υλικού. Παρότι γραμμικό, το μοντέλο καταφέρνει να αποτυπώσει μη-γραμμικές σχέσεις ανάμεσα στην χρήση πόρων και την επιβράδυνση της απόδοσης, επιτυγχάνοντας σημαντικά καλύτερη ακρίβεια από στατικές ευρετικές συναρτήσεις. [38]

2.3 Kubernetes ως Βασική Υποδομή Ενορχήστρωσης

Οι μέθοδοι κατηγοριοποίησης που παρουσιάστηκαν συνδέονται άμεσα με τεχνικές προγραμματισμού εργασιών και δυναμικής κατανομής πόρων, οι οποίες εφαρμόζονται σε συστήματα ενορχήστρωσης. Τα συστήματα αυτά είναι αναγκαία για την διαχείριση μεγάλου αριθμού εφαρμογών, όπως αναλύθηκε στο Κεφάλαιο 1. Το Kubernetes αποτελεί την επικρατέστερη πλατφόρμα ενορχήστρωσης κοντέινερ ανοιχτού κώδικα, σε σύγχρονα περιβάλλοντα υπολογιστικού νέφους [39]. Προσφέρει ένα abstraction layer πάνω από το υποκείμενο υλικό και παρέχει μηχανισμούς για την αυτοματοποιημένη ανάπτυξη, κλιμάκωση και διαχείριση εφαρμογών, ενώ παράλληλα υποστηρίζει υψηλό βαθμό επεκτασιμότητας.

Η αρχιτεκτονική του διαχωρίζεται σε 2 βασικά τμήματα, το **Control Plane** και τους **Worker Nodes**. Το Control Plane περιλαμβάνει:

- τον **API Server**, μέσω του οποίου επικοινωνούν χρήστες και components,
- το **etcd**, μια κατανεμημένη βάση κλειδιού/τιμής όπου αποθηκεύεται η κατάσταση του cluster,
- τον **Controller Manager**, που εφαρμόζει την λογική ελέγχου για replication, nodes και endpoints και

- τον **Scheduler**, που είναι υπεύθυνος για την απόφαση τοποθέτησης των Pod στους κόμβους.

Κάθε Worker Node ενσωματώνει το εργαλείο **Kubelet**, το οποίο είναι υπεύθυνο για την εκτέλεση των Pods, καθώς και το **Kube-Proxy** για την υλοποίηση του δικτυακού επιπέδου.

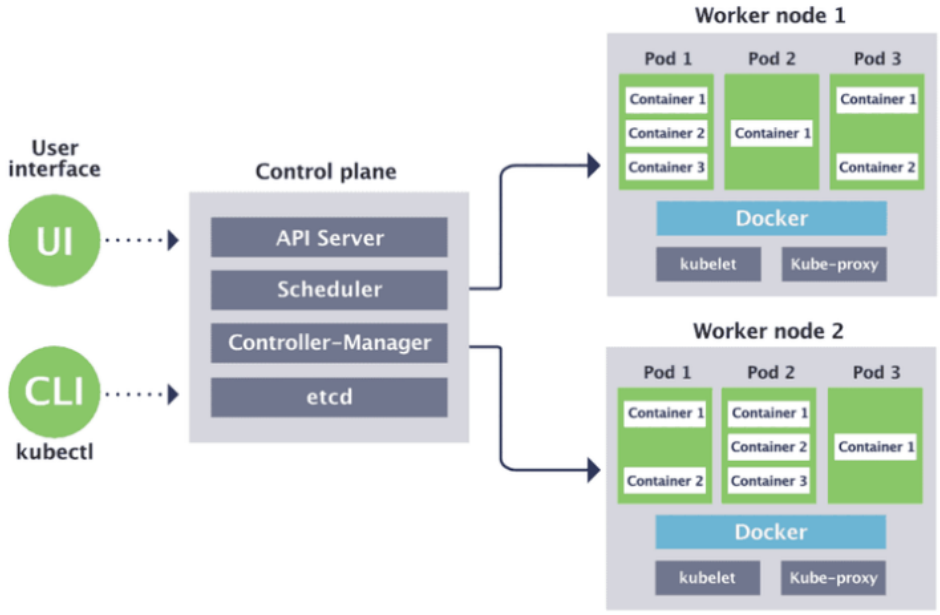

Figure 2.1: Αρχιτεκτονική του Kubernetes

Το **Pod** αποτελεί τη μικρότερη μονάδα εκτέλεσης στο Kubernetes. Περιέχει ένα ή περισσότερα containers τα οποία μοιράζονται αποθηκευτικό και δικτυακό χώρο. Αυτά, διαχειρίζονται από τα **ReplicaSets**, τα οποία εξασφαλίζουν επιθυμητό πλήθος αντιγράφων που ο χρήστης επιθυμεί να εκτελείται μια δεδομένη στιγμή. Επιπλέον, το αντικείμενο **Deployment** αποτελεί ένα abstraction υψηλότερου επιπέδου, και είναι υπεύθυνο για την διατήρηση της επιθυμητής κατάστασης της εφαρμογής στον cluster. Ο χειριστής του cluster, μπορεί να καθορίσει τους πόρους CPU και μνήμης που καταναλώνει ένα Pod, μέσω των παραμέτρων requests και limits.

Το Kubernetes, εισάγει την έννοια το **Service** που ορίζει ένα λογικό σύνολο pods και μια πολιτική με την οποία θα έχει πρόσβαση σε αυτά. Με αυτό τον τρόπο, παρέχεται ένα σταθερό σημείο πρόσβασης προς τα pods. Πιο συγκεκριμένα, υπάρχουν 4 διαφορετικοί τύποι Service:

- Cluster IP. Ο προεπιλεγόμενος τύπος Service. Το καθιστά προσβάσιμο μόνο από το εσωτερικό του cluster.
- NodePort: Εκθέτει το Service στην διεύθυνση IP κάθε κόμβου σε μια στατική θύρα. Το καθιστά προσβάσιμο απο το εξωτερικό του cluster, ζητώντας <NodeIP>:<NodePort>.
- LoadBalancer: Οπως και στην περίπτωση του NodePort, εκθέτει το service εξωτερικά, ωστόσο χρησιμοποιεί επιπλέον τον εξισορροπιστή φορτίου (load balancer) ενός παρόχου cloud.
- ExternalName: Γίνεται μια αντιστοιχία σε μια εξωτερική διεύθυνση DNS, επιστρέφοντας εγγραφή τύπου CNAME η οποία κατευθύνει την κίνηση εκτός cluster.

Οι τύποι service είναι ιδιαίτερα κρίσιμοι για latency-critical εφαρμογές, όπου η σταθερότητα και η εξισορρόπιση φορτίου επηρεάζουν άμεσα την ποιότητα υπηρεσίας (QoS).

Επιπλέον, μια σημαντική λειτουργία που η πλατφόρμα υποστηρίζει είναι το **autoscaling**, μια έννοια που θα μας απασχολήσει αρκετά σε πλαίσια της εργασίας. Αρχικά, ο Horizontal Pod Autoscaler (HPA) αυξομειώνει τον αριθμό των Pods βασισμένος αποκλειστικά στα ποσοστά χρησιμοποίησης της CPU και της μνήμης. Ο Vertical Pod Autoscaler (VPA), ακολουθεί μια διαφορετική στρατηγική, προσαρμόζοντας δυναμικά τα resource requests/limits ενός Pod, επανεκκινώντας το όταν χρειάζεται. Τέλος, ο Cluster Autoscaler (CA) επηρεάζει τον cluster αλλάζοντας το πλήθος worker nodes όταν τα διαθέσιμα δεν επαρκούν.

Χάρη στην ευέλικτη και επεκτάσιμη αρχιτεκτονική του, το Kubernetes αποτελέι βάση για την ανάπτυξη πληθώρας ερευνητικών εργασιών που επεκτείνουν τις δυνατότητες του scheduling των εφαρμογών και της δυναμικής κατανομής πόρων.

2.4 Application Scheduling

Η διαδικασία scheduling καθορίζει πού θα τοποθετηθεί κάθε workload στο cluster και έχει κρίσιμη σημασία για την αποδοτικότητα και την αποφυγή παρεμβολών.

Ο **kube-scheduler** [40, 41] αποτελεί τον προεπιλεγμένο scheduler του Kubernetes και λειτουργεί ως η βάση σύγκρισης (baseline) για κάθε μελέτη scheduling. Η λειτουργία του στηρίζεται σε 2 βασικές φάσεις, το Filtering και το Scoring, οι οποιές συμβαίνουν για κάθε καινούργιο Pod που πρέπει να τοποθετηθεί σε έναν κόμβο. Αρχικά, στην φάση του Filtering, αποκλείονται όλοι οι κόμβοι οι οποίοι δεν πληρούν τους περιορισμούς που θέτει το νέο Pod, όπως resource requests/limits, taints και affinity rules. Στην φάση του Scoring, οι υπόλοιποι κόμβοι βαθμολογούνται με βάση προκαθορισμένες συναρτήσεις, όπως CPU load, memory usage, balanced resource allocation. Πέραν αυτής της απλής υλοποίησης, ο scheduler υποστηρίζει προεκτάσεις, επιτρέποντας την εισαγωή custom plugins σχεδόν σε κάθε στάδιο της διαδικασίας. Ωστόσο, πρέπει να επισημανθεί ότι ο kube scheduler λαμβάνει αποφάσεις βασισμένος εξολοκλήρου στις μετρικές του υλικού, αγνοώντας τα φαινόμενα συντοποθέτησης όπως η παρεμβολή και οι SLO στοχοι, οδηγώντας συχνά στην μείωση της απόδοσης και την υποβάθμιση της ποιότητας υπηρεσίας (QoS).

Για την αντιμετώπιση αυτών των περιορισμών, οι Δελημήτρου και Κοζυράκης πρότειναν το σύστημα **Paragon** [42] , έναν μηχανισμό που εισάγει έννοιες interference awareness και hardware heterogeneity στο scheduling. Στην εργασία αυτή εφαρμόζεται ενας μηχανισμός collaborative filtering για την τοποθέτηση των εφραμογών. Στην μέθοδο αυτή, χρησιμοποιούνται δεδομένα απο προηγούμενες εκτελέσεις μιας εφαρμογής, με στοχο της κατηγοριοποίηση της ως προς την ευαισθησία στην παρεμβολή και στην ετερογένεια του υλικού. Στην συνέχεια το σύστημα, εκτελεί μία greedy ανάθεση τους σε κόμβους με σκοπό την μεγιστοποίηση της απόδοσης και της χρησιμοποίησης του κόμβου. Αυτή η προσέγγιση χωρίς την ανάγκη εκτενούς profiling κατέστησε δυνατή την επίτευξη επιτυχούς QoS σε ποσοστά έως και 91%

Προχωρώντας πέρα των στατικών προφίλ, προτάθηκε ο μηχανισμός **Bubble-FLux** [43] ο οποίο παρέχει ακριβή έλεγχο ποιότητας υπηρεσίάς (QoS) συνδυάζοντας δύο στάδια λειτουργίας. Αρχικά, ο μηχανισμός Dynamic Bubble, εκτιμά την πίεση στους υπολογιστικούς πόρους του υλικού και επιστρέφει μια πρόβλεψη της επίδραση συντοποθετημένως εργασιών σε μια Latency Critical (LC) εφαρμογή. Επειτα στο δευτερο στάδιο, ο μηχανισμός Bubble Flux Engine, βασισμένος σε αυτές τις προβλέψεις, παρακολουθεί την ποιότητα υπηρεσίας (QoS) σε πραγματικό χρόνο και προσαρμόζει την ροή της εκτέλεση εφαρμογών στο ίδιο υλικό. Με αυτό τον τρόπο επιτρέπει την αύξηση της χρησιμοποίησης (utilization) παρέχοντας ταυτόχρονα ικανοποιητικές τιμές του QoS.

Για να ληφθεί υπόψην και το κόστος μετανάστευσης μιας εφαρμογής, οι Romero και Δελημήτρου προτείνουν το **Mage** [44]. Στην εργασία αυτή παρουσιάζεται ένας runtime μηχανισμός, η λειτουργία του οποίου στηρίζεται σε έναν συνεχή κύκλο απο υπολειτουργίες (closed loop system). Αρχικα, κάνει χρήση της τεχνικής data mining, ώστε να εξερευνήσει αρχκές πιθανές τοποθετήσεις. Στην συνέχεια, παρακολουθεί την απόδοση των εφαρμογών, στην τοποθέτηση την οποία αποφάσισε και αξιολογεί εναλλακτικές τοποθετήσεις, συμπεριλαμβάνοντας το κόστος μεταναστευσης εφαρμογής (application migration). Εφαρμόζοντας προσαρμοστικές αποφάσεις σε πραγματικό χρόνο, πετυχαίνει σημαντικές βελτιώσεις απόδοσης, συγκριμένα 38% σε σχέση με τον προκαθορισμένο scheduler και 11% σε σχέση με το Paragon.

2.5 Dynamic Resource Allocation

Η δυναμική κατανομή πόρων επεκτείνει τον απλό προγραμματισμό εργασιών (scheduling). Πέρα απο την αρχική απόφαση τοποθέτησης κατα την εκκίνηση της εφαρμογής, επιβάλλει συνεχή προσαρμογή των εφαρμογών στους κόμβους, βασισμένη στην παρακολούθηση των μετρικών σε πραγματικό χρόνο και σε μοντέλα πρόβλεψης. Πλήθος ερευνητικών εργασιών έχουν προσεγγίσει πρόβλημα αυτό, έχοντας ως στόχο τη διατήρηση της υψηλής επίδοσης και του QoS υπό μεταβαλλόμενες συνθήκες.

Το **Quasar** [13] αποτέλεσε ένα από τα πρώτα συστήματα που εφάρμοσαν αυτή τη λογική σε κλίμακα cluster. Αποτελεί μια QoS aware cluster management αρχιτεκτονική, η οποία δεν στηρίζεται σε στατικές δεσμεύσεις πόρων, όπως προηγούμενες αρχιτεκτονικές, αλλά υπολογίζει δυναμικά την κατάλληλη ποσότητα πόρων με βάση

τους στόχους QoS που ορίζουν οι χρήστες. Χρησιμοποιεί τεχνικές classification για την ταχεία εκτίμηση των απαιτήσεων της κάθε εφαρμογής και της επιδρασης της απο παρεμβολές. Ετσι, υλοποιεί μια αποδοτική κατανομή και ανάθεση των πόρων στον cluster, η οποία προσαρμόζεται με βάση την συνεχή παρακολούθηση των εφαρμογών και την τήρηση των περιορισμών QoS.

Το **FIRM** των Qiu et al. [14] αποτελεί ένα σύστημα δυναμικής διαχείρισης πόρων, σχεδιασμένο για Latency-Critical (LC) εφαρμογές που αποτελούνται απο αλληλοεξαρτώμενα microservices. Η λειτουργία του βασίζεται στην συνεχή παρακολούθηση των εφαρμογών και σε ένα κλειστό feedback loop για την ανίχνευση της παρεμβολής. Το FIRM, αντι να βασίζεται σε στατικές τοποθετήσεις, εφαρμόζει τεχνικές προσαρμοστικής τοποθέτησης και resource throttling. Αξιοποίώντας online μετρικές και μοντέλα μηχανικής μάθησης, ανιχνεύει τα microservices που οφείλονται για την υποβάθμιση του QoS και εφαρμόζει δυναμικό reprovisioning για άμεση αποκατατάση του συστήματος και του επιθυμητού QoS, μειώνοντας δραστικά τις καθυστερήσεις ουράς (tail latencies).

Βασισμένο σε παρόμοια φιλοσοφία, το **ERMS** [45] προσεγγίζει τη δυναμική κατανομή πόρων μέσω προσαρμογής της CPU και της μνήμης που αναθέτει σε κάθε εφαρμογή, χρησιμοποιώντας εκτιμήσεις για την απόδοση. Καθώς διαφορετικά micorservices μιας εφαρμογής, επηρεάζουν διαφορετικά την απόδοδοη, το ERMS κατηγοριοποιεί την καθυστέρηση του καθενός ως γραμμική συνάρτηση του εισερχόμενου φορτίου, της χρήσης πόρων και των παρεμβολών. Βασισμένο πάνω σε αυτή την κατηγοριοποίση κατασκευάζει scaling μοντέλα κατανομής πόρων με τα οποία στοχεύει να πετύχει τα QoS targets. Επιπλέον, προσθέτει καινούργιες scheduling πολιτικές στα microservices που χρησιμοποιούνται απο περισσότερες απο μια εφαρμογές (shares microservices) ωστε να ενισχύσει περαιτέρω την απόδοσης του συστήματος.

Τελος, η υλοποίηση **TraDE** [46] πρόκειται για έναν adaptive scheduling framework για microservices. Για να επιλύσει το πρόβλημα των δυναμικά μεταβαλλόμενων φορτίων αιτημάτων και των διαφορετικών καθυστερήσεων επικοινωνίας μεταξύ κόμβων, το TraDE παρακολουθεί την κυκλοφορία και το δίκτυο σε πραγματικό χρονο και επανατοποθετεί προσαρμοστικά τα microservices ωστε να διατηρεί την επιθυμητή απόδοση στο δίκτυο. Αντί να βασίζεται σε ευρετικές συναρτήσεις, αναπτύσεις πολιτικές που ενσωματώνουν τη χρονική μεταβλητότητα των εφαρμογών καθώς και τις διακυμάνσεις της ζήτησης. Ετσι το σύστημα ανταποκρίνεται με υψηλή απόδοσης σε μη προβλέψιμες συνθήκες, γεγονός που το καθιστά ιδανικό για dynamic edge clusters.

2.6 Η Προσέγγισή μας

Η εργασία αυτή ακολουθεί την ποσοτική μοντελοποίηση της παρεμβολής, στοχεύοντας στην ακριβή εκτίμηση της μείωσης της απόδοσης που προκαλείται σε latency-critical (LC) εφαρμογές. Χρησιμοποιώντας την σουίτα iBench [20] για την δημιουργία διαφορετικών σεναρίων πίεσης των κόμβων και το εργαλείο Intel PCM για την παρακολούθηση των μετρικών υλικού, δημιουργήθηκε ένα σύνολο δεδομένων (Historical Data). Πάνω σε αυτό εκπαιδεύτικε το μοντέλο μηχανικής μάθησης, υπευθυνο για την εκτίμηση της μείωσης της απόδοσης. Ετσι, προτείνεται μια αρχιτεκτονική κλειστού βρόχου ελέγχου (closed-loop control system) που αξιοποιεί αυτό το μοντέλο για να αξιολογεί σε πραγματικό χρόνο την τοποθέτηση των αντιγράφων της εφαρμογής στον Cluster. Σκοπός του συστήματος είναι να επιλέξει το σενάριο τοποθέτησς που θα επιφέρει την καλύτερη απόδοση της εφαρμογής και την διατήρηση της ποιότητας υπηρεσίας (QoS).

Chapter 3

Σχεδιασμός και Αρχιτεκτονική του Προτεινόμενου Μηχανισμού

3.1 Πλαίσιο Προβλήματος και Κίνητρο

Οι σύγχρονες υποδομές υπολογιστικού νέφους φιλοξενούν εφαρμογές με διαφορετικές απαιτήσεις, αναφορικά με την απόδοση. Ορισμένες στοχεύουν κυρίως στην επίτευξη υψηλού throughput, όπως οι best–effort εφαρμογές. Η παρούσα εργασία επικεντρώνεται σε εφαρμογές όπου προτεραιότητα η άμεση απόκριση και η εμπειρία του χρήστη, δηλαδή σε **Latency Critical** (LC) εφαρμογές [47]. Οι LC εφαρμογές χαρακτηρίζονται από αυστηρές απαιτήσεις ποιότητας υπηρεσίας (QoS) και στόχους επιπέδου υπηρεσίας (SLOs), οι οποίοι συχνά συνδέονται άμεσα με κρίσιμους επιχειρηματικούς δείκτες. Βρίσκονται συνήθως στην πρώτη γραμμή εξυπηρέτησης αιτημάτων σε μια αλυσίδα επεξεργασίας (pipeline) και λειτουργούν ως σημεία συμφόρησης (bottlenecks), καθώς αν καθυστερήσουν επηρεάζεται ολόκληρη η αλυσίδα. Ταυτόχρονα, μικρές διαταραχές σε επίπεδο πόρων ή απότομές αλλάγές κυκλοφορία (burstiness) μπορούν να οδηγήσουν σε απότομη αύξηση της καθυστέρησης, με άμεση αρνητική επίδραση στην ποιότητα υπηρεσίας. Οι LC εφαρμογές έχουν εφαρμογή από web servers και βάσεις δεδομένων OLTP μέχρι real–time analytics και υπηρεσίες πολυμέσων.

Στο πλαίσιο της παρούσας εργασίας, ως αντιπροσωπευτική LC εφαρμογή επιλέγεται ο **Nginx** web server. Η εφαρμογή Nginx είναι ιδιαίτερα διαδεδομένη, ειδικά σε αρχιτεκτονικές μικροϋπηρεσιών [48], αφού χρησιμοποιείται ευρέως ως HTTP reverse proxy, load balancer καθώς και static content server. Το γεγονός ότι είναι front facing υπηρεσία και λειτουργεί ως σημείο συμφόρησης (bottleneck) που καθορίζει την εμπειρία του τελικού χρήστη, την θέτει ως την καλύτερη επιλογή για την ανάλυση και αξιολόγηση της προτεινόμενης αρχιτεκτονικής.

Επιπλέον, για να αξιολογηθεί η πραγματική εμπειρία του χρήστη, δηλαδή η ποιότητα υπηρεσίας (QoS), δεν αρκεί ο μέσος χρόνος απόκρισης των LC εφαρμογών. Χρειάζεται να γίνει έλεγχος της λεγόμενης "ουράς" καθυστέρησης, δηλαδή των ακραίων τιμών των πιο αργών αιτημάτων. Στην διεθνή βιβλιογραφία [47, 49], προτείνεται πως ο στόχος ποιότητας υπηρεσίας (QoS) για τις latency–critical εφαρμογές ορίζεται με βάση την καθυστέρηση στο 99ο εκατοστημόριο (**p99 latency**). Η μετρική p99 latency είναι ο χρόνος απόκρισης κάτω από τον οποίο ολοκληρώνεται το 99% όλων των αιτημάτων που φτάνουν στην εφαρμογή. Η μετρική αυτή δίνει μια πιο ρεαλιστική εικόνα της εμπειρίας, καθώς σε εφαρμογές με μεγάλη ταυτόχρονη χρήση, το 1% των πιο αργών απαντήσεων μπορεί να οδηγήσει σε εκαντοντάδες ή ακόμα και χιλιάδες δυσαρεστημένους χρήστες. Αποτυπώνει, λοιπόν, με συνέπεια τη συμπεριφορά της «ουράς» καθυστερήσεων και αποτελεί κατάλληλο κριτήριο αξιολόγησης των LC εφαρμογών.

Η δυσκολία στη διαχείριση τέτοιων εφαρμογών οφείλεται σε **δυναμικές παραμέτρους** που αλληλεπιδρούν και έχουν σημαντική επιρροή στην ποιότητα υπηρεσίας (QoS):

- **Παρεμβολή από συντοποθετημένες εφαρμογές.**

  Το φαινόμενο της παρεμβολής προκύπτει από τον ανταγωνισμό μεταξύ εφαρμογών που συνυπάρχουν στο ίδιο φυσικό μηχάνημα (PM), διεκδικώντας κοινόχρηστους πόρους (resource contention). Η παρεμβολή μπορεί να δημιουργηθεί απο ανταγωνισμό σε επίπεδο CPU, μνήμης καθώς και I/O λειτουργιών, οδηγώντας

στην μείωση της απόδοσης της εφαρμογής. Μια αναλυτική παρουσίαση των αιτιων και των συνεπειών αυτού του φαινομένου δίνεται στην Ενότητα 1.2

- **Μεταβλητότητα στον εισερχόμενο ρυθμό αιτημάτων ανα δευτερόλεπτο (RPS).**

  Ο ρυθμός αυτός παρουσιάζει συχνές διακυμάνσεις, είτε με περιοδικά μοτίβα, είτε με απότομες αιχμές (bursts) λόγω απρόβλεπτων εξωτερικών γεγονότων, όπως η εμφάνιση ενός γεγονότος με υψηλή ταση σε ένα κοινωνικό δίκτυο. Έτσι μπορούν να δημιουργηθούν φαινόμενα underprovisioning, όπου οι διαθέσιμοι πόροι δεν επαρκούν να διαχειριστούν τα αιτήματα και αυτά συσσωρεύονται, ή φαινόμενα overprovisioning, όπου δεσμεύονται περισσότεροι πόροι από όσους πραγματικά χρειάζονται και άρα μένουν ανεκμετάλλευτοι [50, 51]. Ο συνδυασμός αυτών των φαινομένων καθιστά ασταθή την ποιότητα υπηρεσίας (QoS).

Παρά την υψηλή σημασία αυτών των δυο παραμέτρων, οι υπάρχουσες προκαθορισμένες πολιτικές βασίζονται σε απλοποιημένα κριτήρια. Για παράδειγμα ο kube-scheduler του Kubernetes, χρησιμοποιεί κριτήρια όπως η δεσμευμένη CPU ή μνήμη, αγνοώντας τόσο το φαινόμενο της παρεμβολής όσο και τη δυναμική φύση της κυκλοφορίας [40]. Επομένως, οι αποφάσεις τοποθέτησης συχνά αποδεικνύονται αναποτελεσματικές, έχωντας ως άμεση συνέπεια την αύξηση της p99 latency και την υποβάθμιση της συνολικής εμπειρίας.

Η προσπάθεια διαχείρισης του perfomance interference, είναι ιδιαίτερα απαιτητική καθώς εμφανίζονται σημαντικές προκλήσεις. Αρχικά, αποτελεί ένα πολύπλοκο και μη γραμμικό φαινόμενο, καθώς οι αιτίες του είναι σύνθετες και διαφορετικοί τύποι εφαρμογών αντιδρούν διαφορετικά σε κάθε μορφή ανταγωνισμού υπολογιστικών πόρων. Επιπλεόν, η φυσή των LC εφαρμογών, καθιστά απαραίτητη την λήψη αποφάσεων σε πραγματικό χρόνο, γεγονός που απαιτεί τεχνικές γρήγορης ανίχνευσης και πρόβλεψης. Πρέπει επίσης, να ληφθούν υπόψην οι περιορισμοί σχετικά με την ορατότητα του συστήματος στα εσωτερικά χαρακτηριστικά των εφαρμογών ή των αρχιτεκτονικών των μικροϋπηρεσιών, οι οποίοι δυσκολεύουν την διαδικασία. Τέλος, οι μηχανισμοί ανίχνευσης καθώς και οι μηχανισμοί ελέγχου πρέπει να βρίσκονται κοντα στο κοντά στο επίπεδο του υλικού και να επιφέρουν όσο το δυνατόν χαμηλότερο υπολογιστική επιβάρυνση, ώστε να μην μειώσουν περαιτέρω την απόδοση [52]. Συνολικά, η δημιουργία παρεμβολής λόγω ανταγωνισμού των εφαρμογών στους πόρους των φυσικών μηχανημάτων, αποτελεί έναν από τους πιο δύσκολους προς διαχείριση παράγοντες σε σύγχρονα container–based clusters.

Για την αντιμετώπιση των προκλήσεων που αναλύθηκαν παραπάνω και την αποτελεσματική διαχείριση των LC εφαρμογών, απαιτείται μια αρχιτεκτονική ελέγχου που πρέπει να:

- Παρακολουθεί μετρικές σε επίπεδο υλικού για την εκτίμηση της παρεμβολής.
- Προβλέπει τις μεταβολές της κυκλοφορίας.
- Ενοποιεί τις αποφάσεις κλιμάκωσης και τοποθέτησης.
- Λειτουργεί δυναμικά, σε πραγματικό χρόνο.

Η ανάγκη για ένα τέτοιο μηχανισμό είναι το βασικό κίνητρο για τον σχεδιασμό της προτεινόμενης αρχιτεκτονικής ελέγχου. Τα διακριτά υποσυστήματα αυτής της αρχιτεκτονικής καθώς και οι προκλήσεις που αυτά παρουσιάζουν, αναλύονται στην επόμενη ενότητα.

3.2 Προτεινόμενη Αρχιτεκτονική και Προκλήσεις

Η αρχιτεκτονική που προτείνεται βασίζεται σε έναν περιοδικό βρόχο ελέγχου, ο οποίος έχει σχεδιαστεί ώστε να λειτουργεί πάνω από το **Kubernetes**, το οποίο αποτελεί την επικρατέστερη πλατφόρμα ενορχήστρωσης container. Στόχος του βρόχου ελέγχου είναι η λήψη δυναμικών αποφάσεων κλιμάκωσης και τοποθέτησης αντιγράφων για LC εφαρμογές σε διαφορετικά φυσικά μηχανήματα στα οποία παρατηρούνται φαινόμενα παρεμβολής, με σκοπό τη διατήρηση των στόχων ποιότητας υπηρεσίας (QoS).

Η **ακριβής υλοποίηση** και τα βασικά του υποσυστήματα θα παρουσιαστούν στην επόμενη ενότητα. Σε αυτή την ενότητα παρουσιάζεται η θεωρητική οργάνωση του ελεγκτή.

Ο βρόχος ελέγχου ενεργοποιείται περιοδικά και περιλαμβάνει τέσσερα διακριτά υποσυστήματα:

- **Monitoring**
- **Scaling Decision**

- **Placement Decision** και
- **Controller**

Κάθε υποσύστημα κατέχει κρίσιμο ρόλο στη ροή πληροφορίας και λήψης αποφάσεων, δημιουργώντας έτσι ενα συνεκτικό μηχανισμό δυναμικής διαχείρισης πόρων, όπως απεικονίζεται στο Σχήμα 3.1.

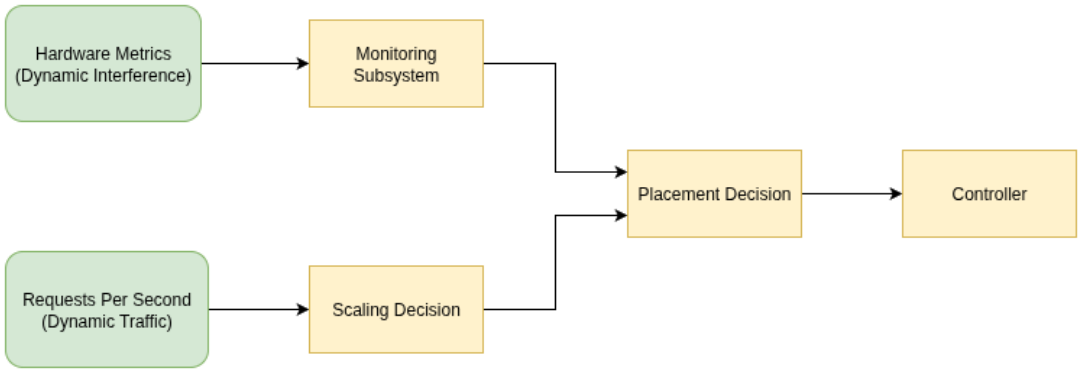

Figure 3.1: Θεωρετική Προσέγγιση Προτεινόμενης Αρχιτεκτονικής

Monitoring

Το πρώτο και θεμελιώδες συστατικό του μηχανισμού ελέγχου είναι το υποσύστημα παρακολούθησης της κατάστασης κόμβου, δηλαδή του φυσικού εξυπηρετητή. Κεντρικό ρόλο σε αυτή τη διαδικασία παίζουν οι **hardware counters**, καθώς μέσω αυτών το υποσύστημα θα συλλέξει τις μετρικές χαμηλού επιπέδου που αποτυπώνουν την κατάσταση των κόμβων και των υπολογιστικών πόρων του και αποτελούν το κύριο μέσο για την μελέτη του ανταγωνισμού. Στόχος του υποσυστήματος δεν είναι η επεξεργασία ή ανάλυση των δεδομένων, αλλά απλώς η πρόσβαση σε αυτά.

Πρέπει να επισημανθεί οτι η παρακολούθηση μέσω των δεικτών του υλικού, συνοδεύεται από σημαντικές προκλήσεις. Τα δεδομένα εμφανίζουν θόρυβο και απαιτούν κανονικοποίηση πριν μπορέσουν να αξιοποιηθούν περαιτέρω. Επιπλέον, αναφορικά με την ακρίβεια των μετρήσεων, η χρονική ευθυγράμμιση μεταξύ των μετρικών και των εφαρμογών που περιέχονται στον κόμβο κρίνεται αναγκαία. Εάν οι μετρήσεις δεν αντιστοιχηθούν σωστά στη δυναμική συμπεριφορά της εφαρμογής, το αποτέλεσμα μπορεί να είναι λανθασμένες εκτιμήσεις για την παρεμβολή.

Scaling

Δευτερο στην σειρά έρχεται το υποσύστημα κλιμάκωσης, το οποίο είναι υπεύθυνο για την εκτίμηση του πλήθους των αντιγράφων (**replicas**) της εφαρμογής που απαιτούνται ώστε αυτή να παραμείνει εντός των στόχων ποιότητας υπηρεσίας (QoS). Αντί να βασίζεται σε ποσοστά χρήσης CPU, όπως κάνει ο κλασικός Horizontal Pod Autoscaler, εδώ η απόφαση πηγάζει απο την αντιστοίχιση του αριθμού αντιγράφων με το ρυθμό εισερχόμενων αιτημάτων (RPS).

Η δημιουργία αυτής της συσχέτισης βέβαια δεν αποτελεί εύκολη διαδικασία. Η σχέση RPS και καθυστέρησης είναι μη γραμμική και επηρεάζεται έντονα από εξωτερικούς παράγοντες όπως η παρεμβολή. Η ανάπτυξη αξιόπιστων μεθόδων (πειραματική χαρτογράφηση ή μοντέλα μηχανικής μάθησης), είναι κρίσιμη για να επιτευχθεί ακριβής πρόβλεψη. Παράλληλα, σε αυτή την διαδικασία πρέπει να λαμβάνεται υπόψη ο χρόνος εκκίνησης των αντιγράφων, ώστε να αποφεύγονται ταλαντώσεις στην απόδοση, στην περίπτωση αυξομείωσης του αριθμού των αντιγράφων.

Το αποτέλεσμα του υποσυστήματος κλιμάκωσης αποτελεί μια αρχική πρόταση, η οποία μπορεί να αναθεωρηθεί στο επόμενο υποσύστημα (Placement), όταν αξιολογηθεί η επίδραση της παρεμβολής.

Placement

Το υποσύστημα τοποθέτησης είναι υπεύθυνο για τη δημιουργία του τελικού **πλάνου τοποθέτησης αντιγράφων** στους κόμβους του cluster. Λαμβάνει ως είσοδο την πρόταση του υποσυστήματος κλιμάκωσης και τις μετρικές υλικού που έχουν συλλεχθεί στο υποσύστημα παρακολούθησης και εξετάζει όλους τους δυνατούς τρόπους κατανομής των αντιγράφων στους διαθέσιμους κόμβους. Στόχος του είναι η επιλογή του συνδυασμού που επιτυγχάνει τη μεγαλύτερη απόδοση και άρα την καλύτερη ποιότητα υπηρεσίας (QoS).Ετσι, η τελική απόφαση μπορεί να διαφέρει εκείνη του υποσυστήματος κλιμάκωσης, στην περίπτωση που η αξιολόγηση δείξει ότι ένας μικρότερος αριθμός αντιγράφων σε λιγότερο επιβαρυμένους κόμβους με χαμηλότερη παρεμβολή οδηγεί σε καλύτερη συνολική επίδοση.

Ο υπολογισμός της απόδοσης του κάθε πλάνου, αποτελεί το σημαντικότερο βήμα αυτού του υποσυστήματος και μπορεί να γίνει με διαφορετικές μεθόδους. Οι απλές υλοποιήσεις βασίζονται σε ευρετικούς αλγορίθμους που υπολογίζουν ένα σκορ παρεμβολής ανα κόμβο [53], ενώ πιο εξελιγμένες προσεγγίσεις αξιοποιούν μοντέλα μηχανικής μάθησης τα οποία προβλέπουν την αναμενόμενη απόδοση.

Η ακρίβεια της εκτίμησης, επηρεάζεται έντονα απο την δυναμική φύση της παρεμβολής. Ταυτόγχρονα η κλιμάκωση σε μεγαλύτερα clusters μπορεί να δημιουργήσει προκλησεις, καθώς όσο μεγαλώνει το πλήθος κόμβων και αντιγράφων, ο αριθμός των πιθανών σεναρών αυξάνεται εκθετικά, καθιστώντας έτσι την αξιολόγηση τους εξαντλητική, τόσο χρόνικά όσο και σε σχέση τους υπολογιστικούς πόρους.

Σε κάθε περίπτωση, το υποσύστημα τοποθέτησης αποτελεί το σημείο όπου η παρακολούθηση των κόμβων και η κλιμάκωση συνδυάζονται για να παραχθεί η πιο αποδοτική κατανομή αντιγράφων.

Controller

Το υποσύστημα ελέγχου είναι υπεύθυνο για τη σύνδεση της προτεινόμενης αρχιτεκτονικής με το εργαλείο ορχήστρωσης Kubernetes. Στόχος του είναι η **εφαρμογή** του βέλτιστου πλάνου τοποθέτησης αντιγράφων που υπολογίστηκε στους κόμβους του Cluster.

Η εφαρμογή αυτή μπορεί να υλοποιηθεί με αρκετούς τρόπους.

- Ως Scheduler Extender[54], που επεκτείνει τον προκαθορισμένο kube-scheduler.
- Μέσω απευθείας τροποποίησης των αντικειμένων Deployment από το Kubernetes API [55], εισάγοντας κανόνες τοποθέτησης, όπως Node Affinity και Node Selector.
- Ως Custom Operator[56], με χρήση ενός Custom Resource Definition (CRD) [57] για την αναπαράσταση του πλάνου τοποθέτησης και reconciliation loop.

Ανεξάρτητα από την προσέγγιση που θα επιλεγεί, όλες οι μέθοδοι υλοποίησης του υποσυστήματος ελέγχου συνοδεύονται από προκλήσεις. Καταρχάς, η εφαρμογή αλλαγών κρύβει τον κίνδυνο προσωρινής αστάθειας του cluster δημιουργώντας καθυστερήσεις. Επιπλέον, κρίσιμη κρίνεται η αντιμετώπιση των φαινομένων race condition ή των αποτυχημένων εντολών στο API, τα οποία μπορούν να προκαλέσουν ασυνέπεια μεταξύ του πλάνου τοποθέτησης αντιγράφων και της πραγματικής κατάστασης του cluster. Για να αποφευχθούν τέτοια φαινόμενα, το υποσύστημα ελέγχου πρέπει υλοποιηθεί με τρόπο τέτοιο ωστε να ενσωματώνει μηχανισμούς απόσβεσης (cooldown intervals) και κατώφλια (thresholds) που να αποτρέπουν τις υπερβολικά συχνές αλλαγές.

Συνολική πρόκληση. Η δυσκολία της υλοποίησης δεν περιορίζεται τόσο στην ακρίβεια του κάθε μεμονωμένου υποσυστήματος, αλλά περισσότερο στη ενορχήστρωση τους σε έναν προβλεπτικό και χαμηλού κόστους μηχανισμό ελέγχου. Ο Πίνακας 3.1 συνοψίζει τις βασικές προκλήσεις κάθε υποσυστήματος.

Υποσύστημα	Κύριες Προκλήσεις Υλοποίησης
Monitoring	– Συγχρονισμός και συνέπεια μετρικών υλικού – Αποθορυβοποίηση και κανονικοποίηση δεδομένων σε πραγματικό χρόνο
Scaling	– Μη γραμμική συσχέτιση μεταξύ RPS και καθυστέρησης – Προσαρμογή σε χρονικές καθυστερήσεις εκκίνησης αντιγράφων – Αντιμετώπιση απότομων διακυμάνσεων ρυθμού αιτημάτων
Placement	– Εκτίμηση επιβάρυνσης λόγω παρεμβολών – Σχεδιασμός ευρετικών ή πρόβλεψη μέσω μοντέλου μηχανικής μάθησης – Κλιμάκωση μεγαλύτερους cluster με περισσότερους κόμβους
Control	– Ασφαλής και συνεπής εφαρμογή αλλαγών μέσω του Kubernetes API – Αντιμετώπιση συχνού reconfiguration μέσω cooldown/thresholds – Αντιμετώπιση race conditions και latency spikes

Table 3.1: Προκλήσεις Υλοποίησης Προτεινόμενης Αρχιτεκτονικής

3.3 Πραγματοποίηση Υποσυστημάτων

Σε αυτή την ενότητα παρουσιάζεται αναλυτικά η υλοποίηση της προτεινόμενης αρχιτεκτονικής, η οποία έχει ονομαστεί **Marla**. Η Marla, αποτελεί μια πρακτική εφαρμογή των τεσσάρων υποσυστημάτων που αναλύθηκαν στην προηγούμενη ενότητα. Στόχος της είναι να λειτουργεί ως εξωτερικός ελεγκτής του cluster, παρακάμπτοντας τον προκαθορισμένο kube-scheduler και χρησιμοποιώντας το Kubernetes API να εφαρμόσει το βελτιστο πλάνο τοποθέτησης αντιγράφων.

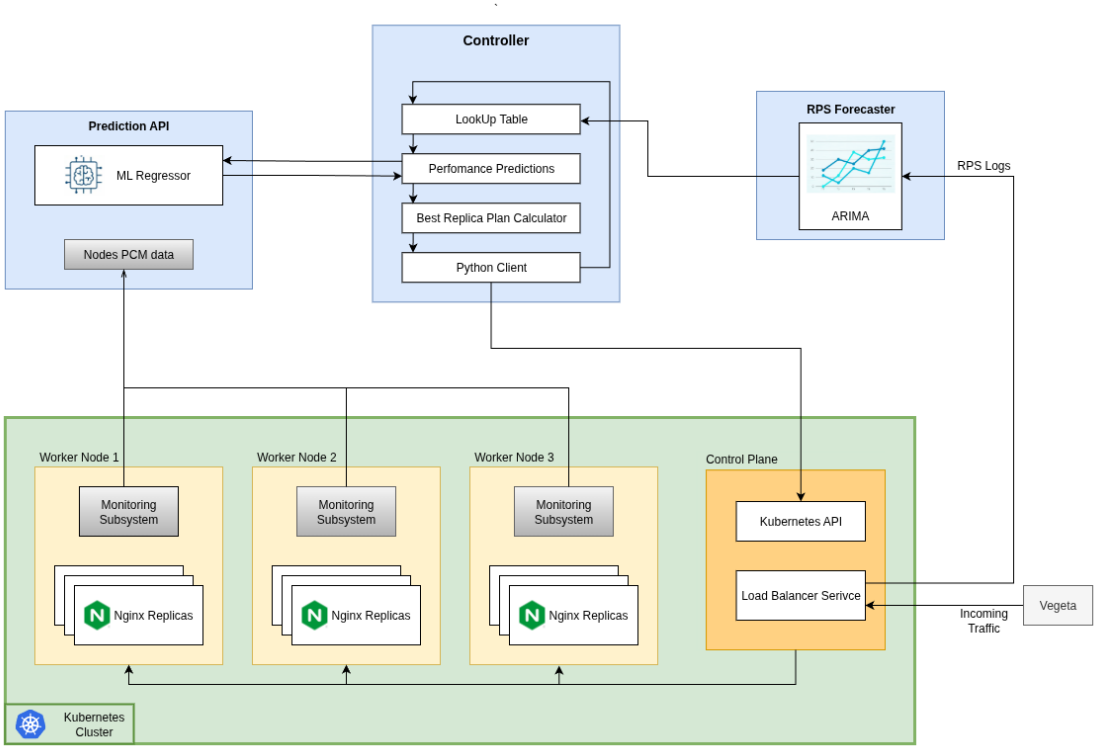

Figure 3.2: Αρχιτεκτονική της Υλοποίησης Marla

Monitoring Subsystem

Για την παρακολούθηση και καταφραφή των μετρικών υλικού, επιλέχθηκε το εργαλείο Intel Performance Counter Monitor (Intel PCM) [25]. Συγκεκριμένα η Marla καταγράφει τις ακόλουθες μετρικές:

- `IPC`
- `L2MISS`
- `L3MISS`
- `C0res%`
- `C1res%`
- `C6res%`
- `PhysIPC`

Το εργαλείο αυτό χρειάζεται τα κατάλληλα δικαιώματα στο επίπεδο του host. Επομένως πρέπει να εκτελείται πάνω στο υλικό και όχι σε καποια εικονοποιημένη εφαρμογή. Οι μετρικές αποθηκεύονται στο αρχείο buffer_metrics.csv, στο οποίο χρησιμοποιείται η τεχνική ολίσθησης παραθύρου (sliding window) ανα 2 δευτερόλεπτα, έτσι ώστε να περιέχονται πάντα οι 40 πιο πρόσφατες εγγραφές, όπως φαίνεται στο αλγόριθμο 1

Algorithm 1: Monitoring Subsystem Data Collection

Input: Intel PCM binary path,
Sampling interval $t_s = 2s$,
Buffer length $L = 40$,
Output directory D.
Output: Sliding buffer file `buffer_metrics.csv` with most recent L samples.

```
create directory D if not exists;
load kernel module msr;
start PCM in background, logging to raw_metrics.csv at interval t_s;
while true do
    copy header lines of raw_metrics.csv to temporary file;
    append last L data lines of raw_metrics.csv to temporary file;
    atomically replace buffer_metrics.csv with temporary file;
    sleep for 2s;
```

Παράλληλα, αναπτύχθηκε μια υπηρεσία REST, το οποίο διαβάζει τα περιεχόμενα του αρχείου και τα εκθέτει μέσω ενός HTTP endpoint. Για την υλοποίηση του χρησιμοποιήθηκε η γλώσσα προγραμματισμού python (Flask API), ώστε να μην καταναλώνει σε μεγάλο βαθμό τους υπολογιστικούς πόρους. Καθώς το εργαλείο ορχήστρωσης είναι το Kubernetes, μπορεί να χρησιμοποιηθεί ένα αντικείμενο Deployment που να περιέχει αυτή την υπηρεσία, με κατάλληλο volume mount προς το αρχείο στο host, ώστε τα δεδομένα να είναι διαθέσιμα σε οποιοδήποτε σημείο του cluster τοποθετηθεί η υλοποίηση Marla.

Scaling

Το υποσύστημα κλιμάκωσης χρησιμοποιεί ένα μοντέλο πρόβλεψης τύπου **ARIMA** (AutoRegressive Integrated Moving Average) [58]. Το μοντέλο εκπαιδεύεται συνεχώς με δεδομένα σε πραγματικό χρόνο, καταγράφοντας τις πρόσφατες μεταβολές στον ρυθμό αιτημάτων (RPS) και παρέχοντας προβλέψεις για το επόμενο χρονικό διάστημα. Ετσι, είναι ικανό να αξιοποιεί τις αλλαγές στον εισερχόμενο ρυθμό αιτημάτων ώστε να ανταποκρίνεται δυναμικά σε αιφνίδιες αυξήσεις ή μειώσεις.

Η πρόβλεψη για την επόμενη τιμή του RPS αντιστοιχίζεται με το βέλτιστο αριθμό αντιγράφων μέσω ενός προκαθορισμένου πίνακα **lookup table** ο οποίος έχει προκύψει πειραματικά, απο προηγούμενες εκτελέσεις της εφαρμογής κάτω απο διαφορετικό ρυθμό αιτημάτων και με διαφορετικό πλήθος αντιγράφων.

Το υποσύστημα κλιμάκωσης, έχει υλοποιηθεί σε Python ως συνάρτηση, η οποία επστρέφει την προτεινόμενη τιμή για τον αριθμό των αντιγράφων.

Placement (Prediction API)

Στην υλοποίηση της Marla, το υποσύστημα τοποθέτησης έχει την μορφή μιας υπηρεσίας REST, το **Prediction API**, το οποίο είναι υπεύθυνο για την πρόβλεψη της απόδοσης σε διαφορετικά σενάρια τοποθέτησης αντιγράφων.

Η είσοδος της υπηρεσίας πρέπει να περιλαμβάνει

- (α) τις μετρικές υλικού από το Monitoring
- (β) την πρόταση αντιγράφων από το Scaling
- και (γ) τον προβλεπόμενο ρυθμό αιτημάτων (RPS).

,

Το Prediction API παράγει για ένα σενάριο τοποθέτησης, ένα αποτέλεσμα αποδοσης (perfomance score) για κάθε κόμβο, το οποίο εκφράζει την αναμενόμενη μείωση της απόδοσης λόγω της παρεμβολής σε έναν κόμβο.

Σε θεωρητικό επίπεδο, για R αντίγραφα σε Y κόμβους θα πρέπει να αξιολογηθούν

$$\binom{R+Y-1}{Y-1}$$

διαφορετικοί συνδυασμοί, το οποίο είναι υπολογιστικά απαγορευτικό.

Για τον λόγο αυτό, το Prediction API υπολογίζει

$$R \times Y$$

προβλέψεις, οι οποίες είναι αρκετές για την δημιουργία ενός λεξικού προβλέψεων, **Prediction Dictionary**. Αυτό περιέχει την αναμενόμενη απόδοση για κάθε κόμβο n_i και για κάθε πιθανό πλήθος αντιγράφων $r \leq R$, με βάση ένα προεκπαιδευμένο μοντέλο μηχανικής μάθησης (Αλγόριθμος 2). Το λεξικό θα επιστραφεί στο υποσύστημα ελέγχου για την σύνθεση του τελικού πλάνου τοποθέτησης.

Algorithm 2: Prediction Dictionary Construction in Marla

Input: Pretrained ML model M,
Set of nodes $\{n_1, n_2, \ldots, n_Y\}$,
Recommended number of replicas R from Scaling subsystem and
Predicted RPS rps.
Output: Dictionary of Predictions $\mathcal{P}$ with entries $(n_i, r) \mapsto$ predicted performance.

```
P ← ∅ ;
foreach node n_i ∈ {n_1, n_2, ..., n_Y} do
    for r ∈ {1, ..., R} do
        features ← collect(PCM metrics of n_i, replicas r, predicted RPS);   // From Monitoring Subsystems
        np ← M.predict(features) ;                                           // Normalized Performance Prediction
        P[n_i][r] ← np ;
return P
```

Το μοντέλο μηχανικής μάθησης έχει εκπαιδευτεί σε ένα μεγάλο σύνολο δεδομένων απο προηγούμενες εκτελέσεις της εφαρμογής, για διαφορετικά σενάρια παρεμβολών, ρυθμού εισερχόμενων αιτημάτων και αριθμού αντιγράφων και έχει αποθηκευτεί ως αρχείο **.pkl**. Η ακριβής διαδικασία δημιουργίας του, καθώς και η διαδικασία επιλογής του μοντέλου αναλύονται στην Ενότητα 4.3

Λόγω του υπολογιστικού κόστους, στην υλοποίηση της Marla το Prediction API εκτελείται εξωτερικά απο τον Cluster, ωστε να μην δημιουργήσει επιπλέον παρεμβολή στους υπόλοιπους κόμβους εργασίας. Εναλλακτικά, μπορεί να υλοποιηθεί ως ένα αντικείμενο Deployment στο Kubernetes, με κατάλληλο volume mount προς το αρχείο του προεκπαιδευμένου μοντέλου.

Controller

Στην υλοποίηση Marla, το υποσύστημα ελέγχου αξιοποιεί το **Prediction Dictionary** που λαμβάνει απο το Prediction API για να επιλέξει το τελικό πλάνο τοποθέτησης των αντιγράφων, στοχεύοντας στη μεγιστοποίηση

του **Aggregate Score**. Καθώς το πλήθος των κόμβων και των αντιγράφων ήταν μικρό και το υπολογιστικό κόστος αμελητέο, η υλοποίηση βασίστηκε σε **brute force** προσέγγιση, όπως φαίνεται στον Αλγόριθμο 3

Ταυτόγχρονα, στην πειραματική αξιολόγηση χρησιμοποιείται cluster με 2 κόμβους, όπου ενσωματώθηκε στην υλοποίηση μια απλή ποινή στην περίπτωση που όλα τα αντίγραφα συγκεντρώνονταν σε έναν μόνο κόμβο, ώστε να αποτρέπονται οι συχνές αλλαγές. Σε μεγαλύτερα cluster, απαιτείται μια πιο γενική ποινή με βάση τη σύγκριση του νέου πλάνου με το προηγούμενο.

Η υλοποίηση Marla εφαρμόζει το τελικό πλάνο τοποθέτησης αντιγράφων στον Cluster μέσω του Kubernetes API, ενημερώνοντας τα αντικείμενα Deployment σε κάθε κόμβο και βασίζεται στον Python client του Kubernetes [59], ο οποίος προσφέρει αξιόπιστη και προγραμματιστικά ελεγχόμενη τροποποίηση του αριθμού των αντιγράφων.

Για να διασφαλιστεί η ομαλή μετάβαση, η εφαρμογή του τελικού πλάνου γίνεται με προσεκτικά βήματα.

- Η αύξηση των αντιγράφων γίνεται πρώτη (scale up), ώστε να αποφευχθεί το φαινόμενο των cold starts.
- Εισάγεται μια τεχνητή μικρή καθυστέρηση πριν απο την μείωση αντιγράφων (scale down).

Algorithm 3: Best Replica Plan Selection in Marla - Brute Force

Input: Dictionary of predictions from Prediction API $\mathcal{P}$,
Recommended number of replicas R from Scaling subsystem,
Number of nodes Y and their identifiers $\{n_1, n_2, \ldots, n_Y\}$.
Output: Best replica distribution plan Π^* across all nodes.

$best\_score \leftarrow -\infty$;
$best\_plan \leftarrow \varnothing$;
foreach $total\_replicas \in \{1, \ldots, R\}$ **do**
  foreach *valid distribution* $(r_1, r_2, \ldots, r_Y)$ *such that* $\sum_i r_i = total\_replicas$ **do**
    foreach *node* n_i **do**
      if $r_i > 0$ ***and*** $r_i \in \mathcal{P}[n_i]$ **then**
        $np_i \leftarrow \mathcal{P}[n_i][r_i]$
      else
        $np_i \leftarrow 0$;
    $score \leftarrow \dfrac{\sum_i r_i \cdot np_i}{\sum_i r_i}$;
    if $score > best\_score$ **then**
      $best\_score \leftarrow score$;
      $best\_plan \leftarrow (r_1, r_2, \ldots, r_Y)$;
return $best\_plan$

Chapter 4

Πειραματική Αξιολόγηση

4.1 Πειραματική Διάταξη και Υποδομή

Στο παρόν κεφάλαιο εστιάζουμε στις λεπτομέρειες της πειραματικής διάταξης, στη διαδικασία δημιουργίας του συνόλου δεδομένων, στην εκπαίδευση του μοντέλου παλινδρόμησης και, τέλος, στην πειραματική αξιολόγηση της αρχιτεκτονικής **Marla**.

Intel NUC: Τα πειράματα εκτελέστηκαν σε φυσικό σύστημα τύπου Intel NUC, το οποίο περιέχει 8 πυρήνες CPU και 8 GB RAM. Το σύστημα αυτό παρέχει το απαιτούμενο υπόβαθρο για την προσομοίωση ενός περιβάλλοντος multi-node cluster πάνω σε έναν μόνο εξυπηρετητή.

Minikube: Ως ενορχηστρωτής επιλέχθηκε το Minikube [60], το οποίο θεωρείται μία από τις πλέον καταλληλότερες επιλογές όταν η ανάπτυξη της συστάδας (cluster) περιορίζεται σε έναν μόνο φυσικό εξυπηρετητή. Το Minikube προσφέρει πλήρη υποστήριξη για multi-node clusters μέσω του driver **Docker** [61], επιτρέποντας την προσομοίωση ενός ρεαλιστικού περιβάλλοντος Kubernetes πάνω σε μία μόνο μηχανή. Στην παρούσα εργασία, ολόκληρος ο cluster εκτελείται στο ίδιο φυσικό σύστημα, με αυστηρή απομόνωση των υπολογιστικών πόρων ώστε να διασφαλίζεται επαναληψιμότητα και να αποτρέπονται ανεπιθύμητες παρεμβολές μεταξύ κόμβων και στοιχείων ελέγχου.

Η απομόνωση επιτυγχάνεται δεσμεύοντας πυρήνες της CPU μέσω περιορισμών που προσφέρει το εργαλείο Docker, όπως (`cpuset-cpus`), σε κάθε κόμβο του Minikube Cluster. Κάθε κόμβος λαμβάνει αποκλειστικά τρεις προκαθορισμένους πυρήνες, ενώ οι υπολειπόμενοι πυρήνες δεσμεύονται για τα εξωτερικά στοιχεία ελέγχου. Συγκεκριμένα:

- **Node 1**: Πυρήνες 0–2, 3 GB RAM — λειτουργεί ως *master + worker*.
- **Node 2**: Πυρήνες 3–5, 3 GB RAM — επιπλέον *worker* node.
- **Στοιχεία ελέγχου** (Marla και βοηθητικές υπηρεσίες): Πυρήνες 6–7 του host, εκτός του Kubernetes cluster.

Πρέπει να επισημανθεί ότι, παρά την αντιστοίχιση πυρήνων μέσω cpuset και τον επιμερισμό μνήμης ανά κόμβο, δεν επιτυγχάνεται πλήρης απομόνωση επιπέδου υλικού καθώς οι δύο λογικοί κόμβοι εκτελούνται στον ίδιο φυσικό επεξεργαστή και μοιράζονται κοινή κρυφή μνήμη τελευταίου επιπέδου (LLC) καθώς και κοινό memory controller. Ωστόσο, τόσο η συλλογή των δεδομένων εκπαίδευσης όσο και η τελική αξιολόγηση πραγματοποιούνται στο ίδιο ακριβώς περιβάλλον, ενισχύοντας την εσωτερική εγκυρότητα των ευρημάτων.

Nginx: Ως εφαρμογή στόχος χρησιμοποιήθηκε ο διακομιστής Nginx, ένας ευρέως διαδεδομένος web server που αποτελεί αντιπροσωπευτικό παράδειγμα latency-critical workload. Για την αξιολόγηση της απόδοσης επιλέχθηκε η μετρική p99 latency, η οποία θεωρείται καθιερωμένος δείκτης ποιότητας υπηρεσίας (QoS) σε συστήματα με αυστηρές απαιτήσεις απόκρισης, όπως έχει ήδη συζητηθεί στο Κεφ. 3

Η ανάπτυξη του Nginx πραγματοποιείται μέσω Deployment object, με μεταβλητό αριθμό **αντιγράφων (replicas)**. Κάθε αντίγραφο συνοδεύεται από ρητούς περιορισμούς στους πόρους που καταναλώνει, μέσω των requests

και limits που ορίζονται στο YAML αρχείο ανάπτυξης. Συγκεκριμένα, κάθε pod δεσμεύει σταθερά `500m CPU` και `512MiB μνήμης`, τόσο ως ελάχιστη απαίτηση (requests) όσο και ως μέγιστο επιτρεπτό όριο (limits). Με τον τρόπο αυτό εξασφαλίζεται ομοιόμορφη και ελεγχόμενη χρήση των διαθέσιμων πόρων.

Η πρόσβαση στα αντίγραφα του Nginx γίνεται μέσω **LoadBalancer Service**, η οποία κατανέμει ισομερώς τα εισερχόμενα αιτήματα στα διαθέσιμα pods και διασφαλίζει ότι το workload παραμένει ομοιόμορφα κατανεμημένο στο cluster.

Στο πλαίσιο της πειραματικής αξιολόγησης, ο αριθμός των αντιγράφων (replicas) μεταβάλλεται και η τοποθέτησή τους στους δύο nodes γίνεται δυναμικά, βάσει των αποφάσεων του επιπέδου ελέγχου Marla. Ο στόχος είναι να επιλεγεί η κατανομή των αντιγράφων που οδηγεί στον ελάχιστο δυνατό επιβραδυντικό παράγοντα (slowdown) λόγω παρεμβολών, μεγιστοποιώντας έτσι την απόδοση.

Vegeta: Για τη δημιουργία της κίνησης, δηλαδή των HTTP αιτημάτων προς τον διακομιστή Nginx, χρησιμοποιήθηκε το εργαλείο Vegeta. Το Vegeta δίνει τη δυνατότητα ρύθμισης παραμέτρων όπως ο αριθμός αιτήσεων ανά δευτερόλεπτο (RPS) και η διάρκεια εκτέλεσης, υποστηρίζοντας τόσο σταθερή όσο και δυναμική κυκλοφορία. Στην παρούσα εργασία αξιοποιήθηκε σε δύο στάδια: αρχικά, κατά τη φάση δημιουργίας του συνόλου εκπαίδευσης, όπου χρησιμοποιήθηκαν προκαθορισμένες τιμές RPS ώστε να καλυφθεί ένα εύρος συνθηκών φόρτου, και στη συνέχεια, κατά την τελική αξιολόγηση, όπου το φορτίο μεταβαλλόταν δυναμικά κάθε λεπτό.

iBench: Για τη δημιουργία σεναρίων παρεμβολής αξιοποιήθηκε η σουίτα iBench, η οποία έχει ήδη παρουσιαστεί αναλυτικά στο Κεφ. 2. Στο πλαίσιο της παρούσας εργασίας, οι εφαρμογές του iBench χρησιμοποιήθηκαν ως pods, με τη χρήση Kubernetes Deployment αντικειμένων. Με τον τρόπο αυτό έγινε δυνατή η κλιμάκωση του αριθμού των αντιγράφων τους, ώστε να δημιουργούνται διαφορετικά επίπεδα παρεμβολής στους κόμβους κατά τη διάρκεια των πειραμάτων.

Πιο συγκεκριμένα, αξιοποιήθηκαν οι εξής τύποι παρεμβολής:

- CPU
- L3 cache
- Memory bandwidth

Οι παρεμβολές αυτές χρησιμοποιήθηκαν τόσο κατά τη φάση εκπαίδευσης του μοντέλου (στατικό περιβάλλον), όσο και στην τελική αξιολόγηση του συστήματος Marla (δυναμικό περιβάλλον).

4.2 Επίδραση Παρεμβολής στην Απόδοση Εφαρμογών

Για να κατανοηθεί σε βάθος ο τρόπος και ο βαθμος με τον οποίο οι παρεμβολές επηρεάζουν την απόδοση latency-critical εφαρμογών, πραγματοποιήθηκε μια εκτεταμένη σειρά από **στατικά πειράματα**. Σε αυτά τα πειράματα χρησιμοποιήθηκε μόνο ένας κόμβος του cluster, ώστε να ελεγχθούν μεμονωμένα οι επιπτώσεις διαφορετικών συνθηκών. Η πειραματική διάταξη για την συλλογή των δεδομένων φαίνεται στο Σχ. 4.1

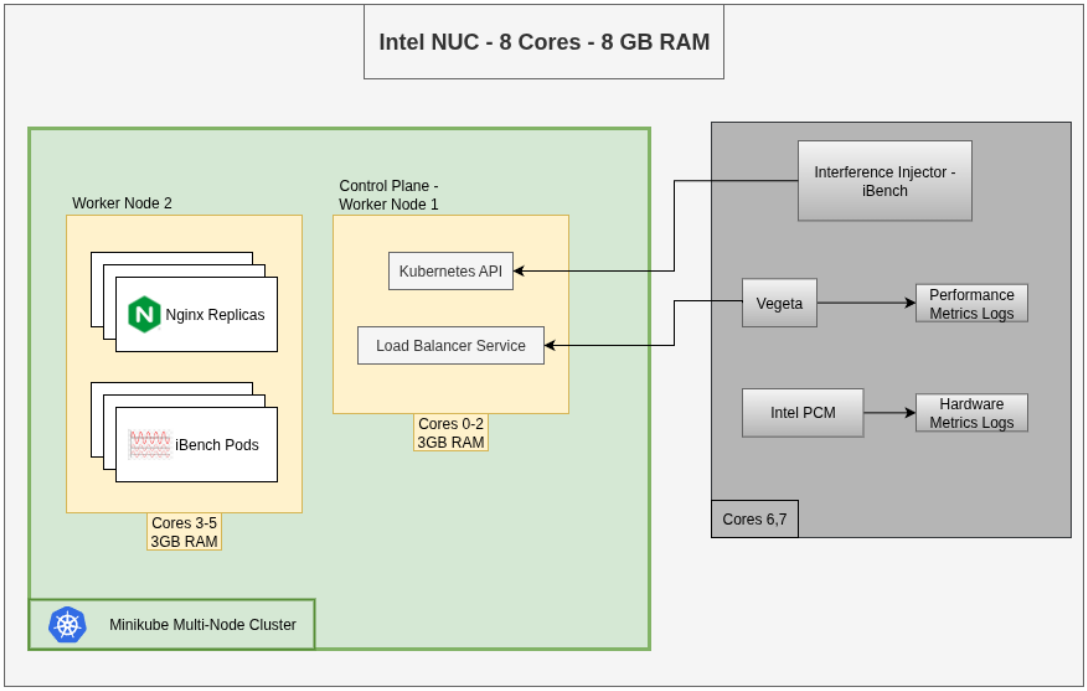

Figure 4.1: Διάταξη Πειραμάτων συλλογής δεδομένων

Οι δοκιμές καλύπτουν όλους τους συνδυασμούς των παρακάτω παραμέτρων, όπως παρουσιάστηκαν στο Κεφ. 3:

- **Αριθμός αντιγράφων (replicas)**: 1, 2, 3, 4
- **Επίπεδο κυκλοφορίας (RPS)**: από 500 έως 4000 με βήμα 500
- **Τύπος παρεμβολής**: συνολικά 23 διαφορετικά σενάρια, ομαδοποιημένα σε 5 κατηγορίες.

Ο πίνακας ακόλουθος πίνακας συνοψίζει τις κατηγορίες παρεμβολής και το πλήθος σεναρίων που μελετήθηκαν:

Κατηγορία Παρεμβολής	**Πλήθος Σεναρίων**
Baseline Scenarios	3
CPU Interference	4
L3 Interference	4
Memory Bandwidth Interference	4
Mixed Interference	8

Table 4.1: Κατηγορίες παρεμβολής και αντίστοιχα σενάρια

Κάθε σενάριο πειράματος συνδύαζε μία επιλογή από κάθε παράμετρο, δηλαδή ένα συγκεκριμένο σενάριο παρεμβολής που παρουσιάστηκε στον Πιν. 4.1, ένα επίπεδο κυκλοφορίας και έναν αριθμό αντιγράφων του Nginx.

Ο **χρονικός ορίζοντας** κάθε δοκιμής είναι 180 sec. Η τιμή των 180 δευτερολέπτων επιλέχθηκε εμπειρικά μετά από αρχικές δοκιμές ως η ελάχιστη διάρκεια ώστε οι μετρικές απόδοσης να σταθεροποιούνται εντός του παραθύρου εκτέλεσης. Μεταξύ διαδοχικών δοκιμών εφαρμοζόταν μια καθυστέρηση (cooldown) 20 sec με σκοπό την εκκαθάριση προσωρινών καταστάσεων. Ως αποτέλεσμα αυτών, κάθε πείραμα είναι ανεξάρτητο από το προηγούμενο. Συνολικά πραγματοποιήθηκαν πάνω από 1800 δοκιμές, λαμβάνοντας υπόψη την επαναληψιμότητα και την επικάλυψη των σεναρίων.

Η απόδοση του συστήματος καταγράφηκε με τη χρήση της **κανονικοποιημένης απόδοσης**, η οποία ορίζεται ως ο λόγος της p99 latency σε σενάριο παρεμβολής B προς τη p99 latency χωρίς παρεμβολή, για την ίδια κυκλοφορία και αριθμό αντιγράφων. Ο ορισμός αυτός έχει προταθεί στη σχετική βιβλιογραφία από τον Koh et al. [2]:

$$Performance = \frac{1}{P99\_latency} \tag{4.2.1}$$

$$NS(F@B) = \frac{Performance(F@B)}{Performance(F@Idle)} = \frac{P99\_latency(F@Idle)}{P99\_latency(F@B)} \tag{4.2.2}$$

Στις επόμενες υποενότητες θα παρουσιαστούν αναλυτικά τα αποτελέσματα της πρώτης φάσης πειραμάτων. Ο συνολικός αριθμός των δοκιμών είναι ιδιαίτερα μεγάλος, επομένως επιλέγεται να παρουσιαστούν μόνο αντιπροσωπευτικά διαγράμματα, τα οποία αντιστοιχούν σε ενδιάμεσες τιμές τόσο του αριθμού αντιγράφων (replicas) όσο και του πλήθους αιτημάτων ανά δευτερόλεπτο (RPS). Με αυτόν τον τρόπο δίνεται μια καθαρή εικόνα των γενικών τάσεων χωρίς να επιβαρυνθεί η ανάλυση με υπερβολικές λεπτομέρειες. Επιπλέον, δεδομένου οτι κάθε δοκιμή των 180 sec έχει ένα διαφορετικό συνδυασμό παραμέτρων, τα διαγράμματα αυτά δεν αποτυπώνουν χρονική εξελιξη, αλλά οι γραφικές απλώς συνδέουν διακριτές κατηγορίες για ευκολία οπτικής σύγκρισης.

4.2.1 Παρεμβολές τύπου CPU

Η κατηγορία παρεμβολών τύπου **CPU** αφορά στην έντονη πίεση των πυρήνων μέσω pods του iBench, τα οποία εκτελούν εφαρμογές με έντονη χρήση CPU. Για να ελεγχθεί η κλιμάκωση της παρεμβολής, αναπτύχθηκαν διαφορετικοί αριθμοί pods (από 1 έως 4), με όλα τα pods να έχουν δεσμευμένους και περιορισμένους πόρους, όπως ακριβώς και τα pods του διακομιστή Nginx. Με τον τρόπο αυτόν η ένταση της παρεμβολής αυξάνεται σταδιακά, από ελαφριά μέχρι έντονη φόρτιση των CPU cores.

Απο το γράφημα 4.2 παρατηρούμε ότι η αύξηση του αριθμού των pods παρεμβολής οδηγεί σε σταθερή πτώση της κανονικοποιημένης απόδοσης. Ένα σαφές «κατώφλι συμφόρησης» εντοπίζεται μεταξύ του πρώτου και του δεύτερου σεναρίου παρεμβολής (CPU1–CPU2), όπου η απόδοση μειώνεται απότομα. Από το σημείο αυτό και μετά, η περαιτέρω αύξηση της παρεμβολής οδηγεί το σύστημα σε ιδιαίτερα χαμηλές τιμές απόδοσης (περίπου 0.2–0.3), δείχνοντας ότι οι διαθέσιμοι πόροι έχουν πλέον κορεστεί, και ότι το σύστημα τείνει να φτάσει σε ένα κατώτατο όριο απόδοσης.

Η προσθήκη περισσότερων αντιγράφων της εφαρμογής Nginx μπορεί να μετριάσει την πτώση της απόδοσης, κυρίως όταν το φορτίο είναι χαμηλότερο και η παρεμβολή περιορισμένη. Ωστόσο, το κέρδος αυτό δεν παρατηρείται σε υψηλότερους ρυθμούς αιτημάτων, ενώ σε κάποιες περιπτώσεις η ίδια η αύξηση των pods επιβαρύνει παραπάνω το σύστημα, καθώς δημιουργεί επιπλέον ανταγωνισμό για τους ίδιους πόρους. Συμπερασματικά, περισσότερα αντίγραφα δεν σημαίνει πάντα καλύτερη απόδοση. Αυτό αποτελεί ένα ιδιαίτερα σημαντικό συμπέρασμα, το οποίο θα διαδραματήσει καθοριστικό ρόλο, στην αξιολόγηση της αποτελεσματικότητας του Επιπέδου Ελέγχου Marla.

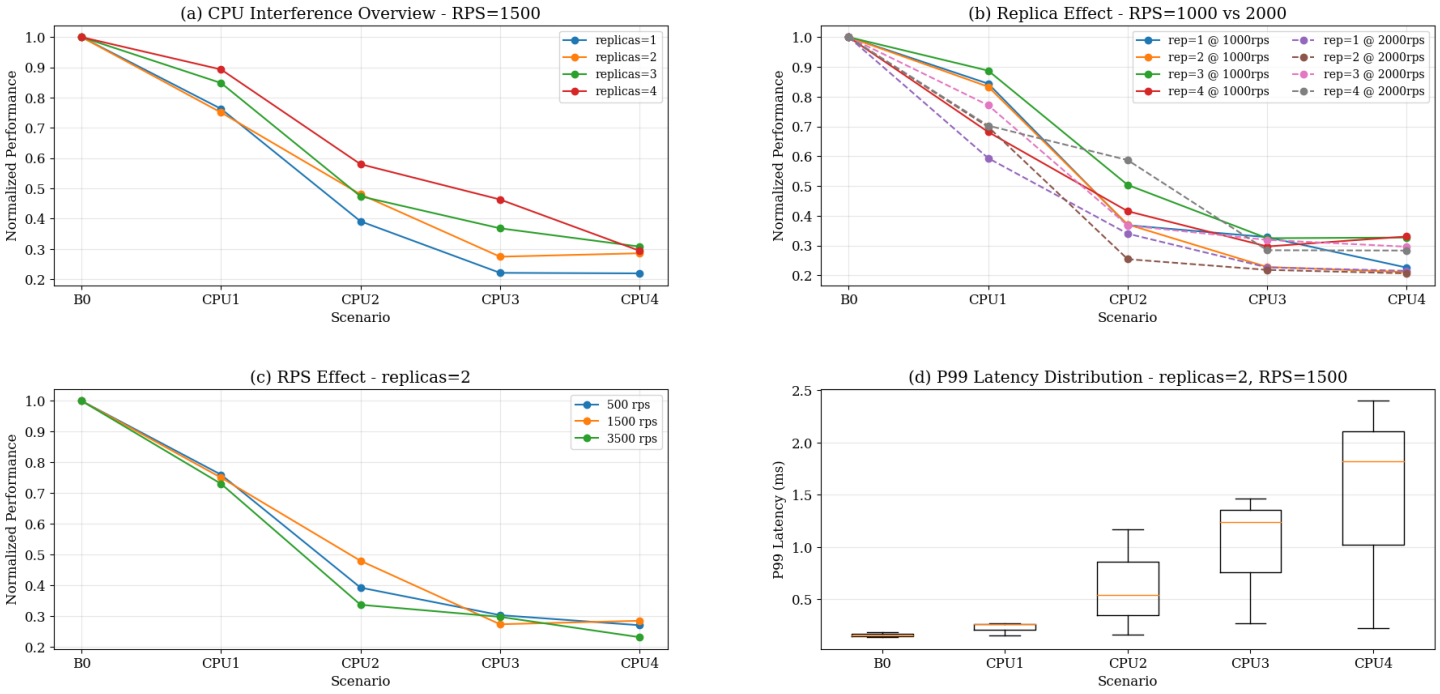

Figure 4.2: Επίδραση παρεμβολών τύπου CPU στην κανονικοποιημένη απόδοση (NP)

Όσον αφορά τον ρυθμό αιτημάτων, η διαφορά μεταξύ 500, 1500 και 3500 RPS φαίνεται σχετικά μικρή σε σύγκριση με την επίδραση της ίδιας της παρεμβολής. Η απόδοση φαίνεται να καθορίζεται περισσότερο από την «ένταση» της παρεμβολής, και λιγότερο απο την αύξηση της κίνησης. Μόνο σε μεσαία επίπεδα παρεμβολής εμφανίζεται κάποια διαφοροποίηση ανάμεσα στις καμπύλες.

Τέλος, αξίζει να σημειωθεί οτι η μετρική p99 καθυστέρηση παραμένει πολύ χαμηλή και σταθερή στα αρχικά σενάρια χωρίς παρεμβολή, αλλά αυξάνεται σημαντικά καθώς προστίθενται pods παρεμβολής. Στα σενάρια υψηλής παρεμβολής, εκτός από τη σημαντική αύξηση της διάμεσης τιμής, παρατηρείται και μεγαλύτερη διασπορά στις μετρήσεις, γεγονός που υποδηλώνει αυξημένη απροσδιοριστία στην απόδοση.

4.2.2 Παρεμβολές στην Cache L3

Η δεύτερη κατηγορία παρεμβολών αφορά στην πίεση της κοινόχρηστης μνήμης cache L3, ενός κρίσιμου πόρου σε πολυπύρηνα συστήματα. Για την προσομοίωση παρεμβολής αυτού του τύπου χρησιμοποιήθηκαν pods τύπου `ibench-l3`, τα οποία εκτελούν εργαλεία σχεδιασμένα να καταναλώνουν επαναλαμβανόμενα τον χώρο της L3 cache. Στις δοκιμές, η κατανομή των πόρων για κάθε pod παρέμεινε σταθερή, με `500m` CPU και `512Mi` μνήμη RAM (`requests` και `limits`). Επιπλέον, δεδομένου ότι στην αρχιτεκτονική του Intel NUC η L3 cache είναι κοινόχρηστη σε όλους τους πυρήνες, υπάρχει πάντοτε ένα περιθώριο σφάλματος στις μετρήσεις· ωστόσο, διατηρώντας την ίδια τοπολογία τόσο στη φάση A όσο και στη φάση B των πειραμάτων, ελαχιστοποιούμε την πιθανότητα συστηματικής μεροληψίας.

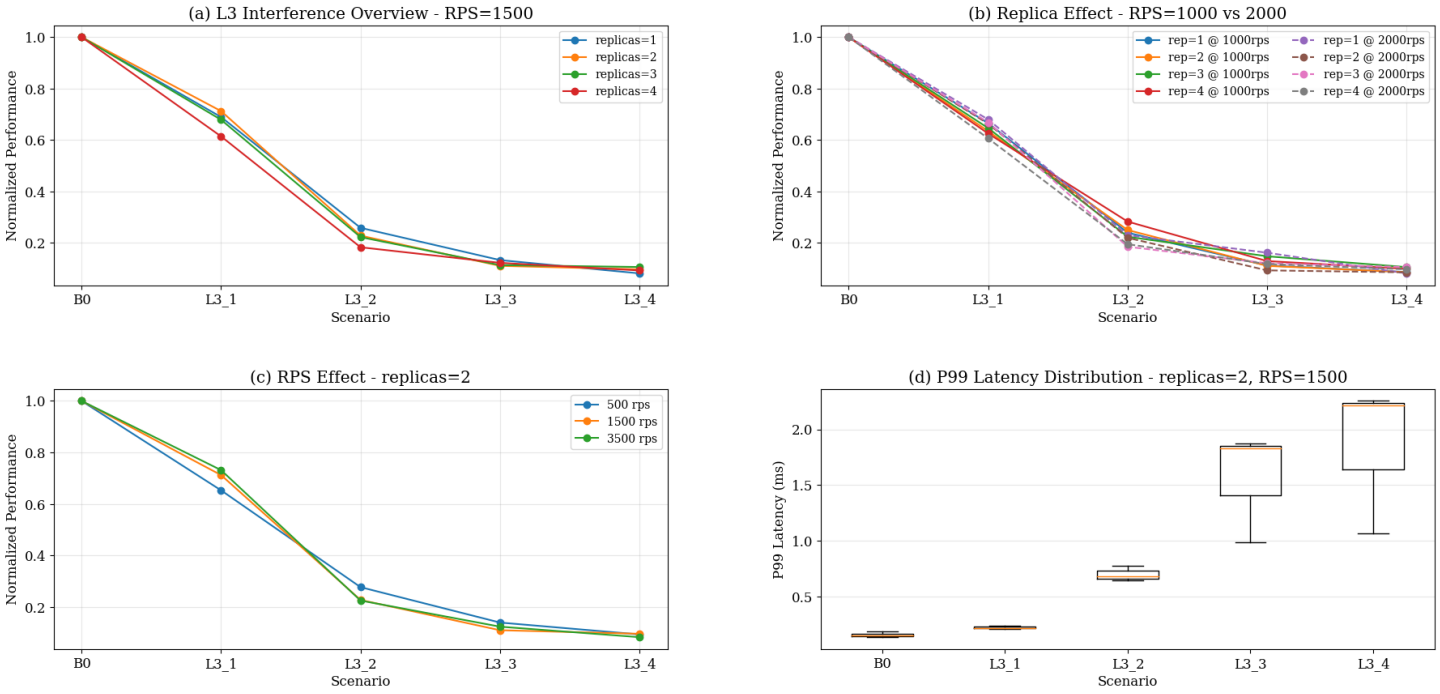

Figure 4.3: Επίδραση παρεμβολών τύπου L3 στην κανονικοποιημένη απόδοση (NP)

Τα αποτελέσματα δείχνουν ότι η παρεμβολή στην L3 cache προκαλεί μια απότομη πτώση της κανονικοποιημένης απόδοσης ήδη μετά το πρώτο σενάριο παρεμβολής, όπου η πτώση της γραφικής γίνεται πολύ απότομη. Από εκείνο το σημείο και μετά, σε όλα τα σενάρια αντιγράφων και για κάθε ρυθμό αιτημάτων, η απόδοση μειώνεται δραστικά και σταθεροποιείται σε τιμές της τάξης του 0.1–0.2, υποδεικνύοντας ότι το σύστημα φτάνει σε ένα κατώτατο όριο απόδοσης, ανεξάρτητα με οποιοδήποτε άλλο δυναμικό παράγοντα. Επιπλέον, η αύξηση του αριθμού αντιγράφων δεν παρέχει κανένα ουσιαστικό όφελος, όπως σε καποιες περιπτώσεις των σεναρίων παρεμβολής CPU, καθώς η κοινόχρηστη φύση της L3 cache επηρεάζει όλα τα pods εξίσου. Συνολικά, η παρεμβολή αυτού του τύπου αποδεικνύεται η πλέον καθοριστική και με την πιο έντονη αρνητική επίδραση στην απόδοση του συστήματος σε σχέση με κάθε άλλη μορφή παρεμβολής.

4.2.3 Παρεμβολές τύπου Memory Bandwidth

Σε αυτή την κατηγορία παρεμβολών, δημιουργείται πίεση στο εύρος ζώνης της μνήμης (memory bandwidth), το οποίο επηρεάζει κυρίως εφαρμογές με έντονη δραστηριότητα Reads/Writes.

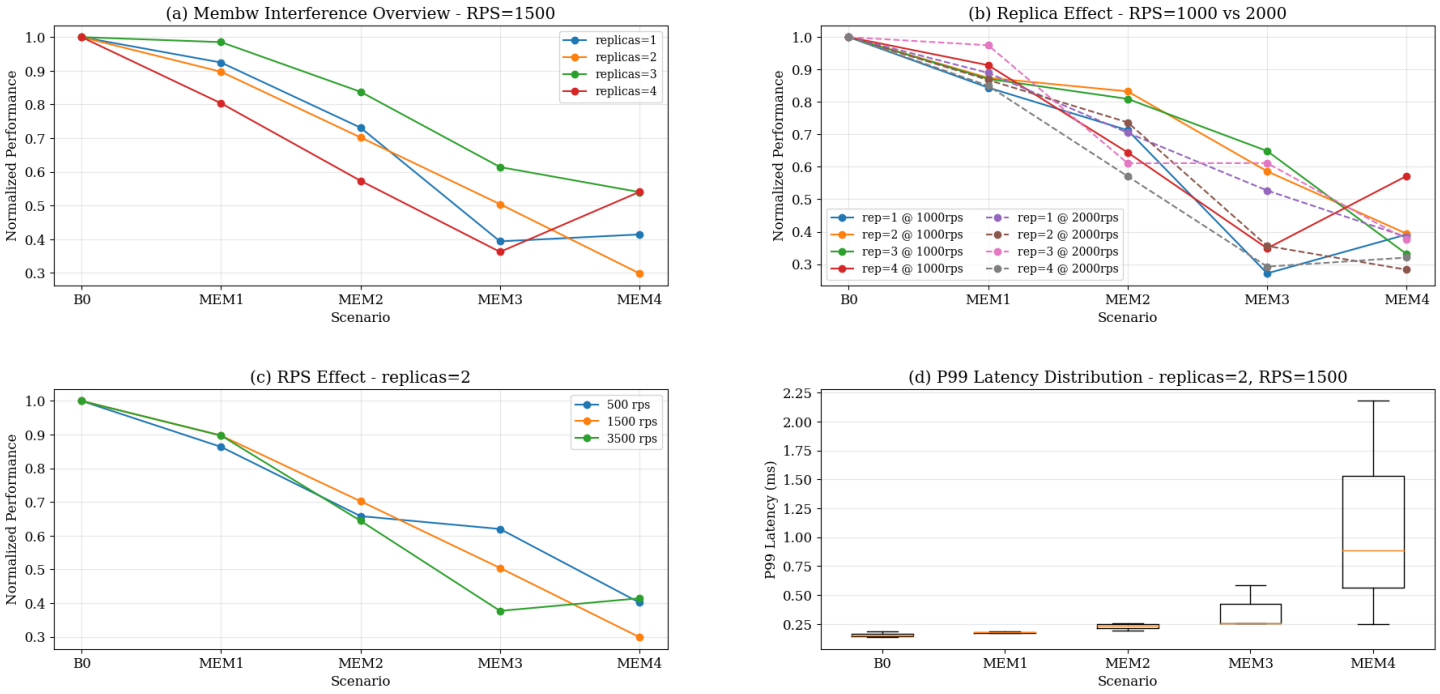

Figure 4.4: Επίδραση παρεμβολών στο *memory bandwidth* (MemBw) στην κανονικοποιημένη απόδοση (NP)

Στα σενάρια παρεμβολής του εύρους ζώνης μνήμης (Memory Bandwidth) παρατηρείται η καλύτερη εικόνα σε σχέση με τα άλλα είδη παρεμβολών, καθώς η κανονικοποιημένη απόδοση μειώνεται μεν σταθερά όσο αυξάνονται τα pods παρεμβολής, αλλά παραμένει σε σχετικά υψηλά επίπεδα ακόμη και στα σενάρια με υψηλή παρεμβολή.

Σημαντικό ρόλο διαδραματίζει ο αριθμός των αντιγράφων: 2 ή 3 αντίγραφα οδηγούν σε καλύτερες επιδόσεις, δείχνοντας ότι η αύξηση των pods μπορεί να αξιοποιηθεί καλύτερα από τους διαθέσιμους πόρους σε αυτά τα σενάρια. Ωστόσο, η περαιτέρω αύξηση σε τέσσερα replicas δεν προσφέρει ουσιαστικό πλεονέκτημα. Σε αρκετές περιπτώσεις μάλιστα εμφανίζει χαμηλότερες τιμές κανονικοποιημένης απόδοσης, γεγονός που αποδίδεται στη παρεμβολή των ίδιων των pods της εφαρμογής nginx. Επιπλέον, παρότι σε ορισμένα σενάρια η καμπύλη των τεσσάρων replicas φαίνεται καλύτερη, η προσεκτική μελέτη της κατανομής της καθυστέρησης P99 φανερώνει ότι στα σενάρια MEM3 και κυρίως στο MEM4 η συμπεριφορά του συστήματος γίνεται ιδιαίτερα απρόβλεπτη.

Έτσι, ενώ η παρεμβολή MemBw φαίνεται να έχει την πιο ήπια επίδραση στην κανονικοποιημένη απόδοση σε σύγκριση με άλλες κατηγορίες και αναδεικνύει τη σημασία της προσεκτικής επιλογής αριθμού αντιγράφων.

4.2.4 Συνδυαστικές Παρεμβολές (Mixed)

Τα συνδυαστικά σενάρια παρεμβολής αποτελούν τον πιο ρεαλιστικό τύπο παρεμβολής, καθώς περιλαμβάνουν ταυτόχρονη πίεση σε πολλαπλούς πόρους: CPU, L3 cache και μνήμη. Κάθε σενάριο προκύπτει από διαφορετικό συνδυασμό pods των προηγουμένως αναφερθέντων τύπων παρεμβολής, όπως φαίνεται και στον επόμενο πίνακα. Η φύση τους τα καθιστά πιο σύνθετα, αφού η συνολική επίδραση δεν είναι πάντα απλώς το άθροισμα των επιμέρους παρεμβολών αλλά προκύπτει από την αλληλεπίδρασή τους.

Scenario	CPU (pods)	L3 (pods)	MemBw (pods)
#1	1	1	0
#2	1	0	1
#3	1	0	2
#4	2	0	1
#5	1	2	0
#6	1	1	1
#7	2	1	0
#8	0	1	3

Table 4.2: Σενάρια Συνδυαστικών Παρεμβολών

Figure 4.5: Επίδραση συνδυαστικών παρεμβολών στην κανονικοποιημένη απόδοση (NP)

Στα σενάρια συνδυαστικών παρεμβολών η συμπεριφορά του συστήματος εμφανίζει μεγάλες διακυμάνσεις στην κανονικοποιημένη απόδοση ανάλογα με τον συνδυασμό των pods CPU, L3 και MemBw που εκτελούνται. Σε περιπτώσεις όπου υπάρχει παρεμβολή L3 (π.χ. MIX6 ή MIX8), η απόδοση υποχωρεί σε πολύ χαμηλά επίπεδα, επιβεβαιώνοντας ότι η L3 cache αποτελεί τον πιο καταστροφικό παράγοντα. Αντίθετα, σενάρια με συνδυασμούς CPU και μνήμης χωρίς L3 Cache(π.χ. MIX2) εμφανίζουν υψηλότερες τιμές απόδοσης.

Σχετικά με την επιρροή του αριθμού των αντιγράφων στην κανονικοποιημένη απόδοση, παρατηρούμε ότι αυτή διαφοροποιείται ανάλογα με το σενάριο. Σε σενάρια όπου η παρεμβολή δίνεται κυρίως από pods CPU, η αύξηση των αντιγράφων επιφέρει αύξηση της απόδοσης. Αντίθετα, σε σενάρια όπου συμμετέχουν pods L3, ο αριθμός αυτός αποτελεί λιγότερο αποτελεσματικό παράγοντα. Ο ρυθμός αιτημάτων έχει πιο περιορισμένη επίδραση, τροποποιώντας ελαφρά το γενικό μοτίβο.

Συνολικά, τα μικτά σενάρια δείχνουν ότι στις πραγματικές συνθήκες παρεμβολής οι επιπτώσεις δεν αποτελούν απλό άθροισμα των επιμέρους τύπων, αλλά προκύπτουν από τον συνδυασμό και την αλληλεπίδραση τους, γεγονός που καθιστά απαραίτητη τη συνεχή παρακολούθηση όλων των πόρων και την υιοθέτηση ολιστικών στρατηγικών τοποθέτησης.

4.3 Σύνολο Εκπαίδευσης και Μοντέλο Παλινδρόμησης

Με την χρήση της σουίτας iBench δόθηκε μια σαφής εικόνα για το πώς η συνύπαρξη εφαρμογών που ανταγωνίζονται για συγκεκριμένους πόρους επηρεάζει την απόδοση. Ωστόσο, για να αποκτηθεί μια πιο ολοκληρωμένη κατανόηση της κατάστασης του κόμβου, χρειάζεται η παρακολούθηση μετρικών χαμηλού επιπέδου που αποτυπώνουν τη λειτουργία του υλικού στο σύνολό του. Για τον σκοπό αυτό, χρησιμοποιήθηκε το εργαλείο **Intel PCM**.

Κατά την εκτέλεση κάθε πειράματος, καταγράφηκαν οι εξής επτά βασικές μετρικές ανά πυρήνα:

- `IPC` — Instructions Per Cycle
- `L3MISS` — Ποσοστό αστοχιών στην cache L3
- `L2MISS` — Ποσοστό αστοχιών στην cache L2
- `C0res%`, `C1res%`, `C6res%`
- `PhysIPC` — Πραγματικό IPC σε φυσικό πυρήνα

Η δειγματοληψία των παραπάνω μετρικών πραγματοποιούνταν κάθε 2 δευτερόλεπτα, με αποτέλεσμα κάθε πείραμα διάρκειας 180 δευτερολέπτων να παράγει μια χρονική σειρά δεδομένων με περίπου 90 εγγραφές ανά μετρική. Ωστόσο, η απόδοση του Nginx καταγραφόταν συγκεντρωτικά από το εργαλείο vegeta, με την τελική έξοδο να συνοψίζεται σε μία γραμμή μετρικών (π.χ. p95 latency, p99 latency) ανά πείραμα.

Ετσι, καθίσταται αναγκαίος ο **μετασχηματισμός των χρονοσειρών** που προκύπτουν απο το Intel PCM σε συνοπτικά χαρακτηριστικά, ώστε να είναι δυνατή η ευθυγράμμισή τους με τις μετρικές απόδοσης.

4.3.1 Στατιστική Ανάλυση των PCM Μετρικών

Σκοπός της παρούσας υποενότητας είναι η στατιστική ανάλυση των ακατέργαστων χρονοσειρών των PCM μετρικών, όπως αυτές συλλέχθηκαν από τα πειράματα. Η ανάλυση αυτή αποσκοπεί στη διερεύνηση των κατανομών, της μεταβλητότητας, της συμπεριφοράς των ουρών, των συσχετίσεων μεταξύ των μετρικών, καθώς και της αντιπροσωπευτικής χρονικής τους εξέλιξης.

Μέσα από αυτή τη διαδικασία εξετάζεται εάν η χρήση συνοπτικών χαρακτηριστικών όπως **mean**, **std** και **p95** αρκεί ώστε να αποτυπωθεί με επάρκεια η πληροφορία των PCM εγγραφών για την εκπαίδευση του μοντέλου πρόβλεψης.

Κατανομές (Distribution Analysis). Τα boxplots του διαγράμματος 4.6 δείχνουν ότι οι κατανομές διαφέρουν σημαντικά ανά τύπο παρέμβασης: το IPC μειώνεται και παρουσιάζει μεγαλύτερη διασπορά σε CPU και L3 interference, ενώ οι L3MISS εμφανίζουν βαριές ουρές υπό L3 stress. Αυτό που πρέπει να κρατήσουμε είναι ότι οι μετρικές δεν ακολουθούν συμμετρικές ή στενές κατανομές, άρα χρειάζονται περισσότερα από τον μέσο όρο για να περιγραφούν.

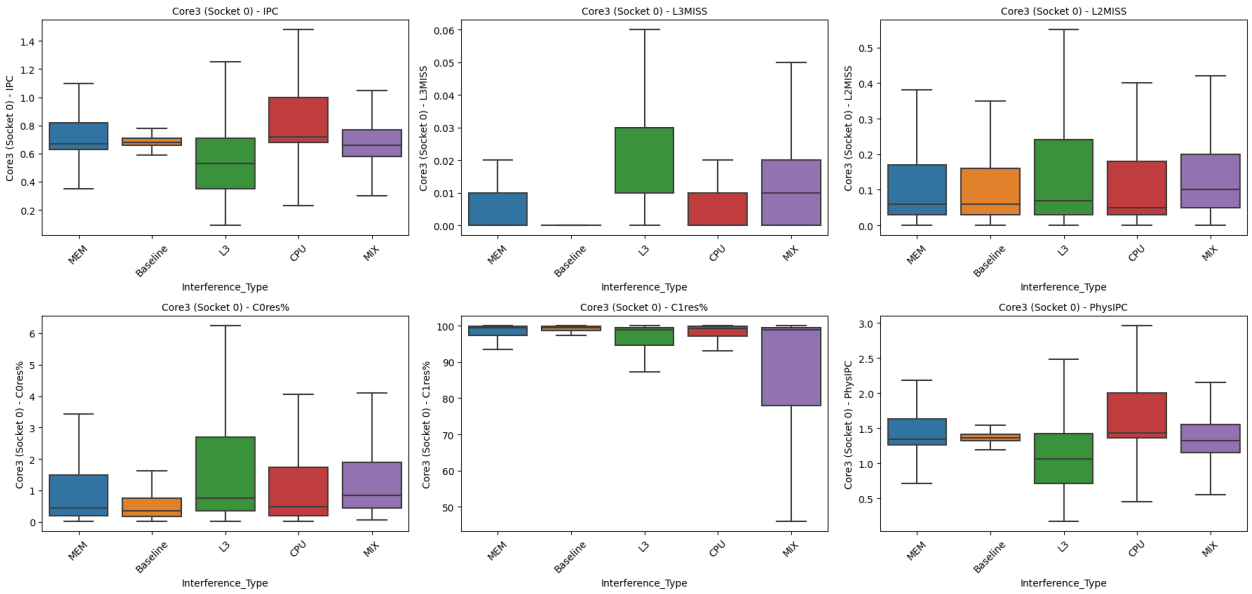

Figure 4.6: Κατανομές επιλεγμένων PCM μετρικών ανά τύπο παρεμβολής

Μεταβλητότητα (Variability Analysis). Τα barplots της τυπικής απόκλισης (STD) του διαγράμματος 4.7 και του συντελεστή μεταβλητότητας (CV) του διαγράμματος 4.8 αναδεικνύουν ότι διαφορετικές μετρικές γίνονται ασταθείς υπό διαφορετικές παρεμβολές: το IPC υπό CPU interference, οι L3MISS υπό L3 interference, και οι C-state residencies υπό Mixed interference. Συμπέρασμα: η std είναι απαραίτητη για να καταγράψει την αστάθεια που δεν φαίνεται στον μέσο όρο, ενώ ο **CV** προσθέτει συμπληρωματική οπτική, αποκαλύπτοντας bursty συμπεριφορές που αλλιώς θα υποεκτιμούνταν.

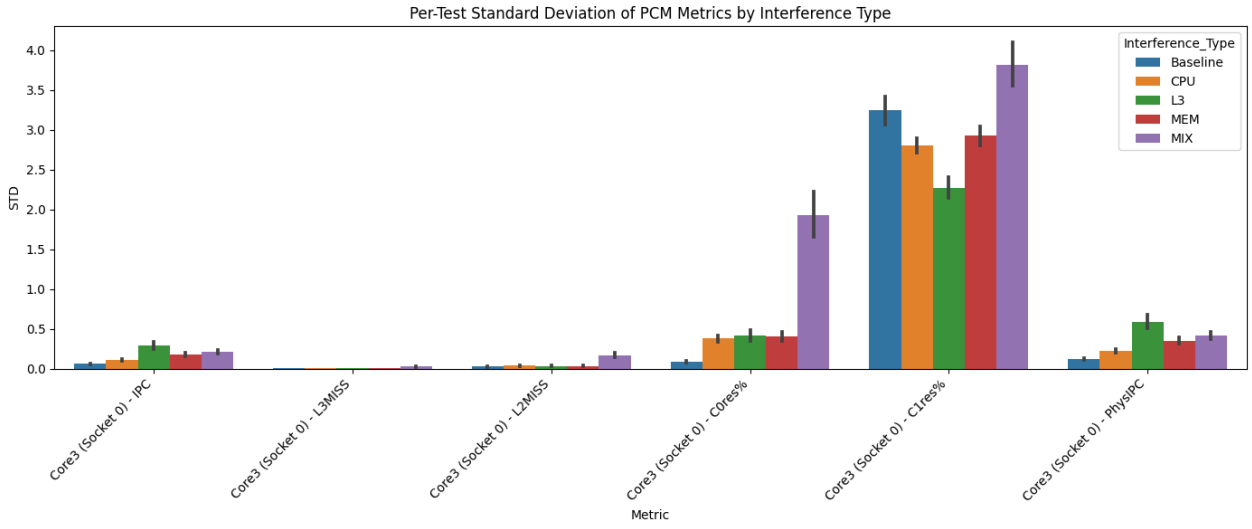

Figure 4.7: Τυπική απόκλιση (STD) των PCM μετρικών ανά τύπο παρεμβολής.

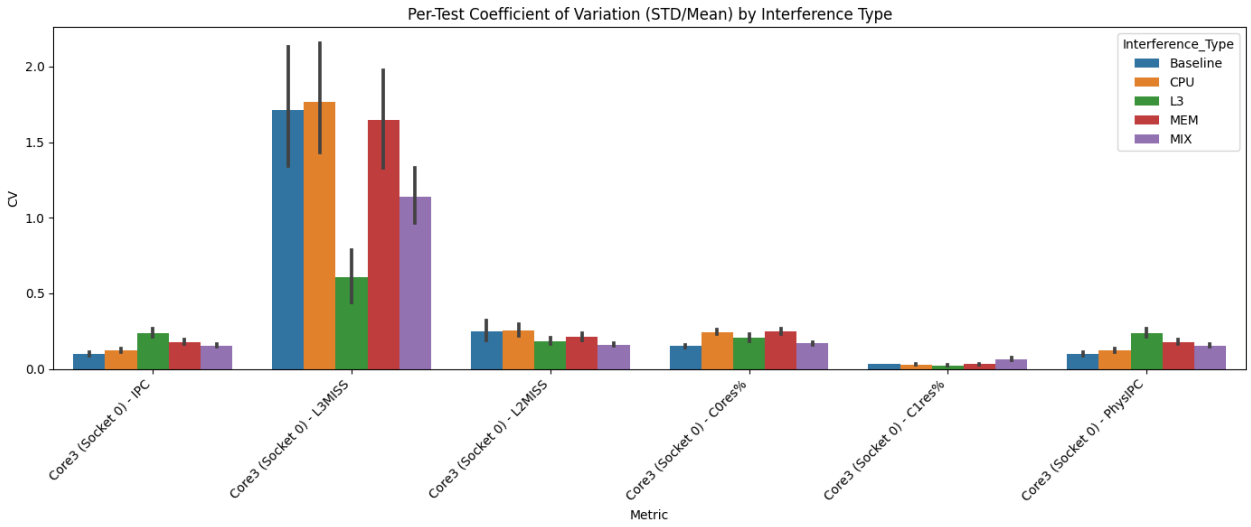

Figure 4.8: Συντελεστής μεταβλητότητας (CV = STD/Mean) των PCM μετρικών ανά τύπο παρεμβολής.

Ουρές/Ακραίες Τιμές (Tails / Extremes). Στην γραφική του IPC του διαγράμματος 4.9 παρατηρείται ότι πειράματα με παρόμοιο μέσο όρο μπορούν να έχουν σημαντικές αποκλίσεις στο p95, γεγονός που δείχνει ότι η ουρά προσθέτει κρίσιμη πληροφορία. Αντίθετα, στις L3MISS το p95 ακολουθεί στενά τον μέσο, κάτι που δείχνει ότι σε ορισμένες μετρικές η προστιθέμενη αξία είναι περιορισμένη. Τα παραδείγματα αυτά είναι αντιπροσωπευτικά και η ίδια τάση παρατηρείται και σε άλλους πυρήνες.

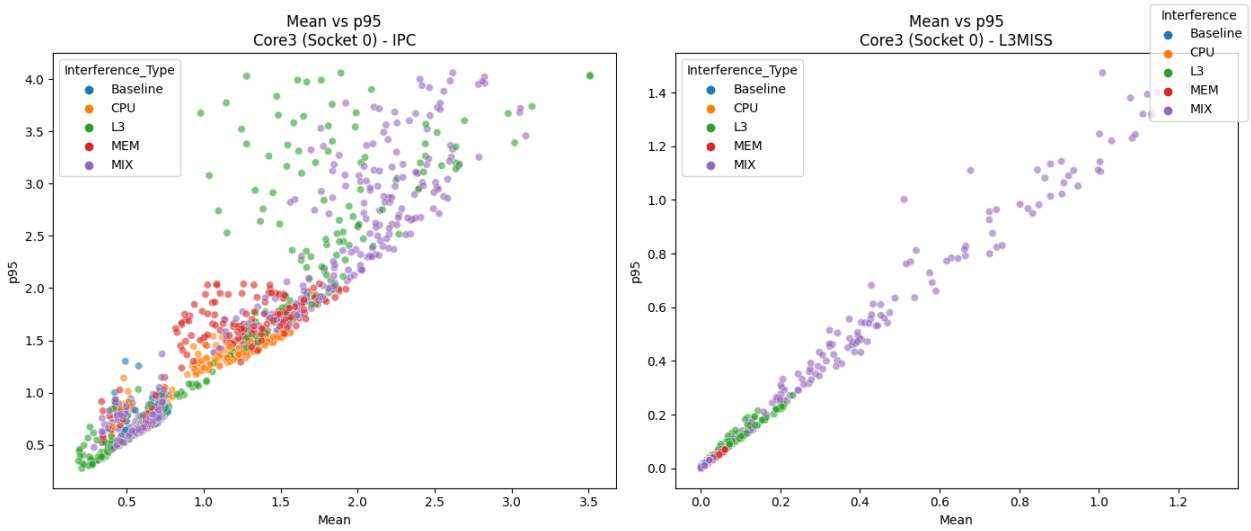

Figure 4.9: Διάγραμμα Mean–p95 για δύο χαρακτηριστικές PCM μετρικές.

Συσχέτιση Μετρικών (Correlation Study). Το heatmap του διαγράμματος 4.10 αναδεικνύει την ύπαρξη πλεονασμών και συμπληρωματικότητας: τα IPC και PhysIPC μπορούν να θεωρηθούν ισοδύναμα (προτείνεται διατήρηση μόνο των IPC για ερμηνευσιμότητα), τα C-states είναι στενά (αντι)συσχετισμένα μεταξύ τους, ενώ τα miss counters σχηματίζουν ξεχωριστό block. Παράλληλα, παρότι τα *mean*, *std* και *p95* εμφανίζουν υψηλές συσχετίσεις εντός κάθε μετρικής, δεν είναι πλήρως πλεονάζοντα (ιδίως στο IPC), κάτι που δικαιολογεί τη χρήση και των τριών συνοπτικών μεγεθών στα τελικά χαρακτηριστικά.

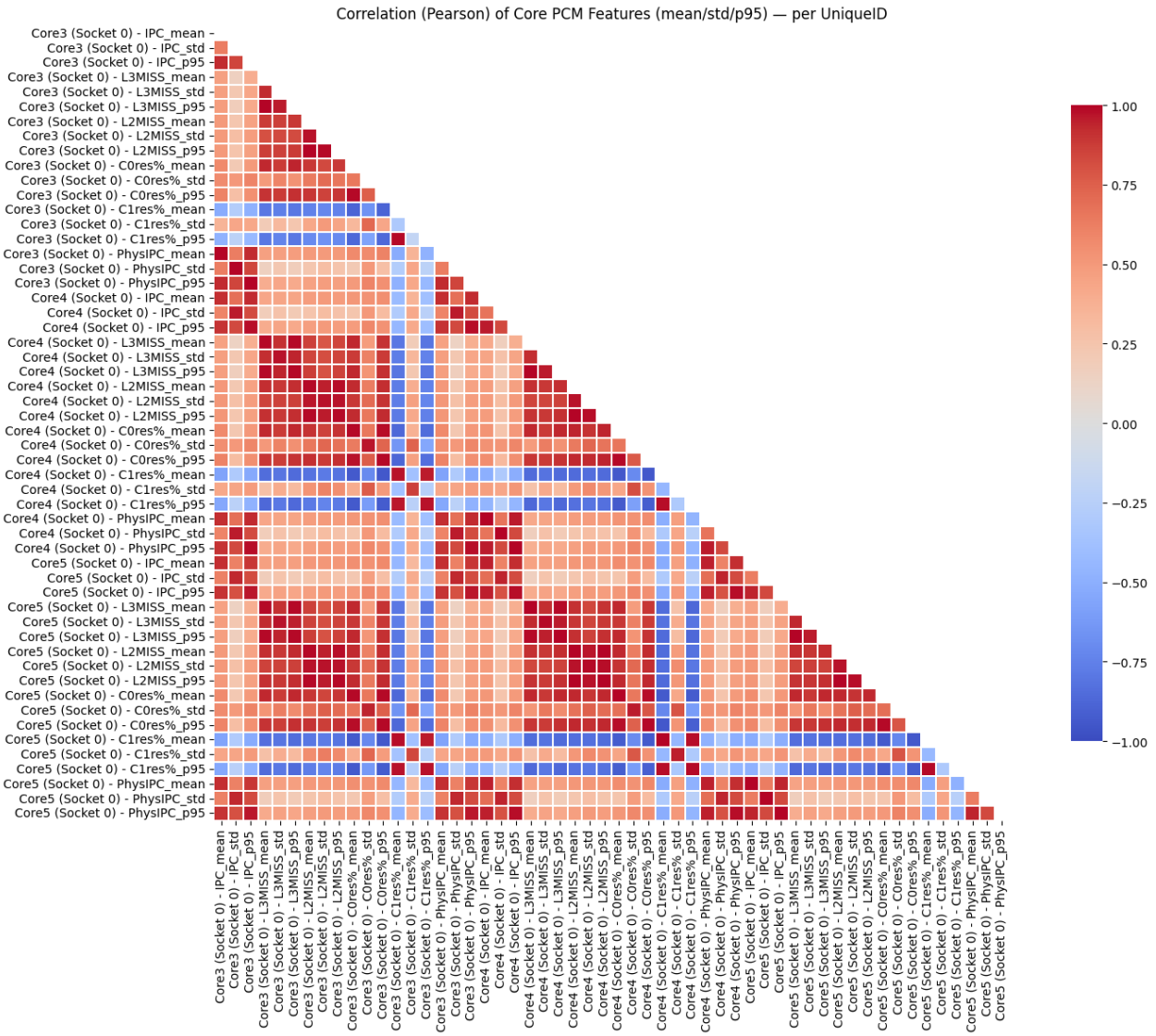

Figure 4.10: Heatmap Pearson συσχετίσεων των συνοπτικών χαρακτηριστικών *(mean, std, p95)* για τις per-core PCM μετρικές

Αντιπροσωπευτική Χρονοσειρά (Representative Time Series). Η ανάλυση των χρονοσειρών που φαίνονται στο διάγραμμα 4.11, καταδεικνύει ότι οι PCM μετρικές παρουσιάζουν σταθερές κατανομές ανά πείραμα, με περιορισμένες διακυμάνσεις. Οι παρεμβολές εκδηλώνονται είτε ως αυξημένη αστάθεια (CPU, MEM), είτε ως χαρακτηριστικά bursts (L3).

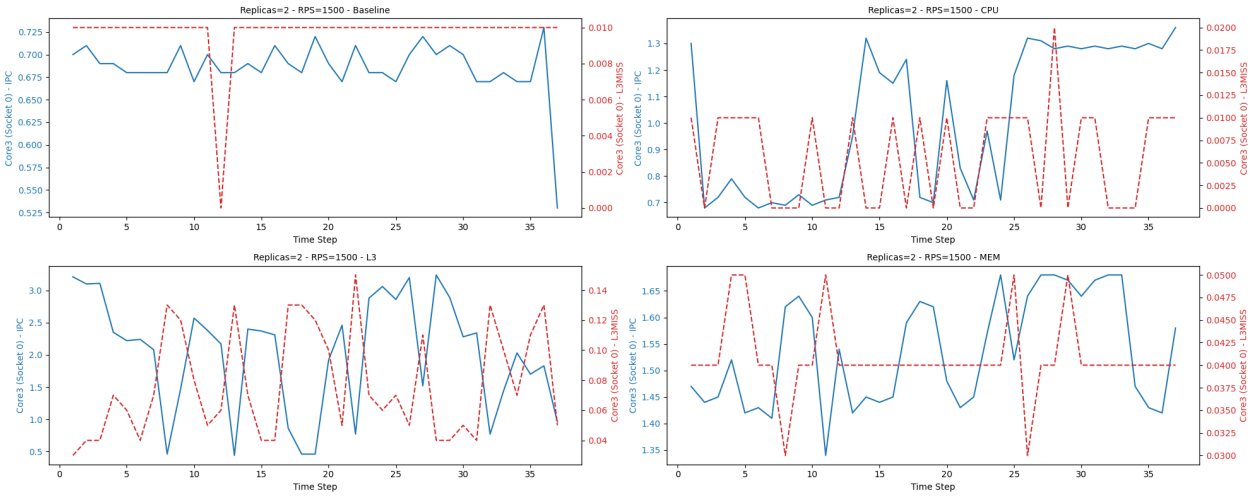

Figure 4.11: Αντιπροσωπευτικές χρονοσειρές για τις μετρικές IPC και L3MISS

Η εικόνα αυτή δικαιολογεί τη χρήση συνοπτικών χαρακτηριστικών για τον επιθυμητό μετασχηματισμό των χρονοσειρών. Ο **μέσος όρος (mean)** αποτυπώνει το βασικό επίπεδο της μετρικής/φόρτου, η **τυπική απόκλιση (std)** καταγράφει την ενδοπειραματική αστάθεια/διακυμάνσεις, και το **p95** συλλαμβάνει τα ακραία επεισόδια (bursts) που επηρεάζουν την απόδοση.

Ως αποτέλεσμα, κατασκευάστηκε ένα **ενιαίο και συνεκτικό σύνολο δεδομένων**, στο οποίο για κάθε πείραμα τα στατιστικά mean, std και p95 των μετρικών που προκυπτουν απο το Intel PCM, συσχετίστηκαν με μία μόνο εγγραφή μετριω απόδοσης (όπως το p99 latency) που προκύπτει από το εργαλείο vegeta. Στην επόμενη ενότητα εξετάζεται πώς το σύνολο αυτό αξιοποιείται για την εκπαίδευση του καταλληλότερου μοντέλου πρόβλεψης.

4.3.2 Επιλογή και Εκπαίδευση Μοντέλου

Απο την στατιστική ανάλυση των χρονοσειρών που κατέγραψε το Intel PCM, είναι σαφές οτι η χρήση όλων των συνοπτικών χαρακτηριστικών θα δημιουργούσε πλεονασμούς, κυριώς λόγω συσχετίσεων, αυξάνοντας τον κίνδυνο υπερπροσαρμογής του μοντέλου. Για τον λόγο αυτό, επιλέχθηκε ενα αντιπροσωπευτικό υποσύνολο, όπου από τα αρχικά 108 χαρακτηριστικά διατηρήθηκαν 25. Το υποσύνολο αυτό, σε συνδυασμό με τις μεταβλητές που περιγράφουν το εκάστοτε πείραμα (αριθμός αντιγράφων nginx και ρυθμός αιτημάτων) κατασκευάζει το σύνολο δεδομένων το οποίο θα χρησιμοποιηθεί για την εκπαίδευση.

Επιπλέον, τα συνοπτικά χαρακτηριστικά (mean, std, p95) δεν υπολογίστηκαν μόνο σε επίπεδο ολόκληρου πειράματος αλλά μέσω **κυλιόμενων παραθύρων χρόνου**. Το μέγεθος παραθύρου θεωρήθηκε υπερπαράμετρος (hyperparameter) και η τελική επιλογή του (8) προέκυψε μέσω αναζήτησης υπερπαραμέτρων (hyperparameter optimization, HPO).

Το τελικό σύνολο δεδομένων έχει διαστάσεις **(1898, 27)**.

Ιδιότητα	**Τιμή / Περιγραφή**
Αριθμός δειγμάτων (rows)	1898
Αριθμός χαρακτηριστικών (columns)	27
PCM χαρακτηριστικά	25 (επιλεγμένα από τα αρχικά 108)
Μέγεθος παραθύρου (hyperparameter)	8
Μεταβλητές πειράματος	Αριθμός αντιγράφων Nginx, Ρυθμός αιτημάτων (RPS)
Στόχος (target)	Κανονικοποιημένη απόδοση (normalized performance) (0,1)

Table 4.3: Σύνοψη χαρακτηριστικών του τελικού συνόλου δεδομένων

Η φύση του προβλήματος καθώς και η μορφή του συνόλου δεδομένων, καθιστούν τη χρήση μοντέλων παλινδρόμησης ως την πιο κατάλληλη επιλογή για τις ανάγκες της παρούσας εργασίας. Καταρχάς, στόχος είναι η πρόβλεψη μιας συνεχούς μεταβλητής, δηλαδή της κανονικοποιημένης απόδοσης. Επιπλέον, το σύνολο των δεδομένων (PCM counters, RPS, replicas) βρίσκεται σε μορφή πίνακα, κάτι που ευνοεί τέτοιου είδους μοντέλα.

Η ύπαρξη μίας μόνο γραμμής μετρήσεων απόδοσης ανά πείραμα (π.χ. p99 latency από το Vegeta) καθιστά ακατάλληλη τη χρήση ακολουθιακών μοντέλων τύπου RNN, όπως τα LSTM, τα οποία προϋποθέτουν πλούσια και εκτενή χρονικά δεδομένα για να αποδώσουν. Ακόμη κι αν διατηρούνταν οι πλήρεις χρονοσειρές, η χρήση RNNs θα εισήγαγε υπερβολικό υπολογιστικό κόστος, καθώς είναι γενικά δύσκολο να επιτευχθεί χαμηλό inference latency με τέτοια μοντέλα [62, 63, 64].

Αντίθετα, οι σχέσεις μεταξύ των χαρακτηριστικών εισόδου (PCM counters, RPS, replicas) και της κανονικοποιημένης απόδοσης δεν ακολουθούν γραμμικά πρότυπα. Περιλαμβάνουν αλληλεπιδράσεις και μη γραμμικές σχέσεις, οι οποίες μπορούν να μοντελοποιηθούν με ακρίβεια από αλγορίθμους δέντρων και ensemble learning. Τέλος, οι απαιτήσεις πραγματικού χρόνου (λήψη αποφάσεων ανά λεπτό) επιβάλλουν την επιλογή μοντέλων που προσφέρουν ταχύτητα πρόβλεψης χωρίς να θυσιάζουν σε ακρίβεια.

Σε αυτό το πλαίσιο δοκιμάστηκαν τρία διαφορετικά μοντέλα παλινδρόμησης:

- **Ridge Regression** (γραμμικό μοντέλο με κανονικοποίηση),
- **Random Forest Regressor** (ensemble δέντρων),
- **XGBoost Regressor** (gradient boosting σε δέντρα).

Τα μοντέλα αξιολογήθηκαν με βάση τις εξής μετρικές:

- **MAE (Mean Absolute Error)**: μέσο απόλυτο σφάλμα πρόβλεψης, με χαμηλότερες τιμές να υποδηλώνουν καλύτερη ακρίβεια.
- **R^2 (Coefficient of Determination)**: ποσοστό διακύμανσης που εξηγείται από το μοντέλο, με τιμές κοντά στο 1 να δείχνουν εξαιρετική προσαρμογή.
- **CV MAE (Cross-Validated MAE)**: μέτρο της γενικευσιμότητας του μοντέλου, μέσω πενταπλής διασταυρούμενης επικύρωσης.

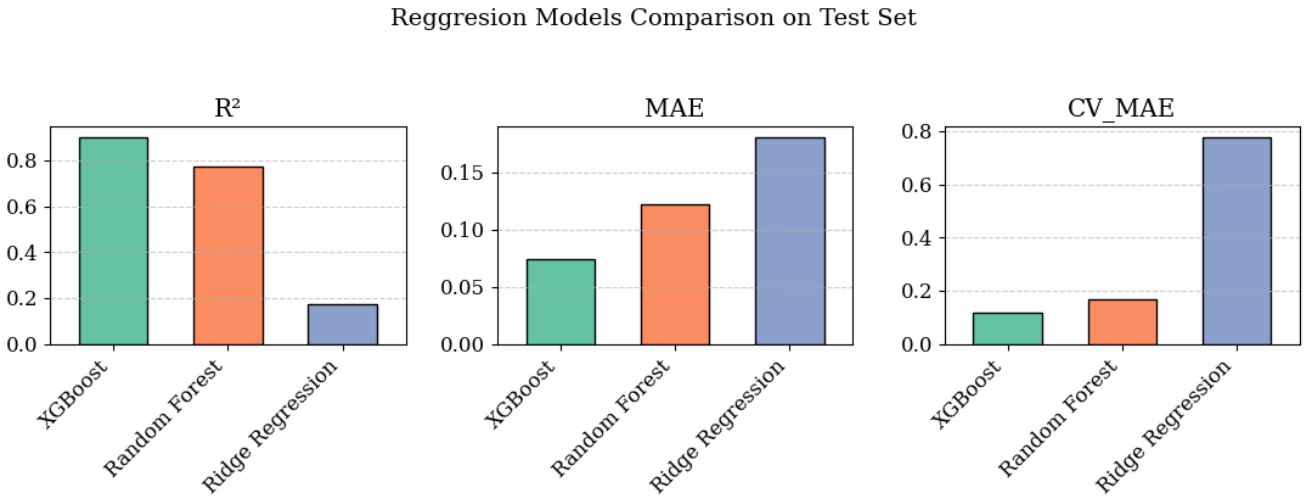

Figure 4.12: Σύγκριση απόδοσης των μοντέλων παλινδρόμησης με βάση τις μετρικές R^2, MAE και CV MAE.

Model	MAE	CV_MAE	R^2
Random Forest	0.122	0.170	0.772
Ridge Regression	0.181	0.778	0.173
XGBoost	0.074	0.117	0.901

Table 4.4: Αποτελέσματα Σύγκρισης Μοντέλων Παλινδρόμησης

Από τη σύγκριση των τριών μοντέλων (Σχ. 4.12 και Πίνακας 4.4) προκύπτει ότι το **XGBoost Regressor** υπερτερεί. Συγκεκριμένα, καταγράφει το μικρότερο CV MAE με τιμή 0.117, κάτι που υποδηλώνει δεν περιορίζεται στο να αποδίδει καλά μόνο στα δεδομένα εκπαίδευσης, αλλά διατηρεί την ακρίβειά του και σε καινούργια δεδομένα, αποδεικνύοντας ισχυρή ικανότητα γενίκευσης χωρίς υπερπροσαρμογη. Παράλληλα, εμφανίζει τον υψηλότερο δείκτη R^2 με τιμή 0.90, γεγονός που δείχνει ότι οι προβλέψεις του μοντέλου εξηγούν την πραγματική μεταβλητότητα των δεδομένων καλύτερα απο τα άλλα μοντέλα. Τέλος, επιτυγχάνει το χαμηλότερο MAE, άρα οι εκτιμήσεις του απέχουν λιγότερο από τις πραγματικές παρατηρήσεις σε σχέση με τα άλλα μοντέλα, παρέχοντας έτσι τις ακριβέστερες προβλέψεις.

Στη συνέχεια, ακολούθησε η ενσωμάτωσή του XGBoost Regressor ως το βέλτιστο μοντέλο, στον προτεινόμενο ελεγκτή **Marla**. Το μοντέλο εκτέθηκε μέσω REST API, ώστε να μπορεί να δέχεται αιτήματα με χαρακτηριστικά ανά node:

- PCM στατιστικά
- αριθμό replicas
- προβλεπόμενο RPS

και να επιστρέφει την εκτιμώμενη κανονικοποιημένη απόδοση.

Ετσι, ο controller αξιολογεί όλα τα εφικτά σενάρια κατανομής αντιγράφων nginx και επιλέγει αυτό με το μέγιστο **Aggregate Score**, σύμφωνα με τη μεθοδολογία που έχει παρουσιαστεί αναλυτικά στο Κεφ. 3.

4.4 Αξιολόγηση του Συστήματος

Σε αυτή την ενότητα παρουσιάζουμε την τελική αξιολόγηση της υλοποίησης **Marla** σε συνθήκες που προσομοιώνουν ένα δυναμικό και ρεαλιστικό περιβάλλον λειτουργίας. Στόχος είναι να εξεταστεί η αποτελεσματικότητα του συστήματος σε πραγματικό χρόνο. Για τον σκοπό αυτό, η υλοποίηση Marla θα συγκριθεί με τον **Default Kubernetes Scheduler** [40].

Ο προεπιλεγμένος scheduler του Kubernetes βασίζει τις αποφάσεις του κυρίως στα resource requests και limits που δηλώνει κάθε pod, φιλτράροντας τους διαθέσιμους κόμβους και στη συνέχεια βαθμολογώντας τους σύμφωνα με κριτήρια όπως η επάρκεια CPU και μνήμης. Ο scheduler αυτός δεν διαθέτει καμία γνώση ή μηχανισμό για την αντιμετώπιση των παρεμβολών. Στο υπόλοιπο κεφάλαιο θα αναφερόμαστε σε αυτόν με την ονομασία **Naive** Scheduler.

Επιπλέον, καθώς ο Naive Scheduler ακολουθεί τον προεπιλεγμένο αλγόριθμο του Kubernetes και δεν λαμβάνει υπόψη παρεμβολές, προσαρμόζει τον αριθμό των αντιγράφων nginx αποκλειστικά με βάση το προβλεπόμενο RPS. Έτσι, μπορεί να αυξήσει τα συνολικά αντίγραφα, αλλά δεν έχει τη δυνατότητα να τα μειώσει στρατηγικά όταν οι παρεμβολές καθιστούν μη αποδοτική την κλιμάκωση τους. Αντίθετα, η Marla αποτελεί έναν interference-aware μηχανισμό. Επομένως, λαμβάνει αποφάσεις που μπορεί να περιλαμβάνουν ακόμη και την τοποθέτηση λιγότερων αντιγράφων, αναγνωρίζοντας ότι σε πολλές περιπτώσεις το κόστος των παρεμβολών είναι μεγαλύτερο από το όφελος της επιπλέον κλιμάκωσης, όπως εξηγήθηκε στην ενότητα 4.2 (Επίδραση Παρεμβολής στην Απόδοση Εφαρμογών).

Για την σύγκριση αυτή, σχεδιάστηκε ένα πείραμα διάρκειας 30 λεπτών, στο οποίο τόσο ο ρυθμός αιτημάτων όσο και τα επίπεδα παρεμβολής μεταβάλλονται. Ο ρυθμός αιτημάτων ακολουθεί ένα τριγωνικό μοτίβο αύξησης και μείωσης (βλ. Σχ. 4.13), ενώ η παρεμβολή εναλλάσσεται δυναμικά μέσω pods της σουίτας **iBench**. Συνολικά δημιουργήθηκαν 4 σενάρια παρεμβολής: παρεμβολή τύπου **CPU**, παρεμβολή στη **L3 Cache**, παρεμβολή στο **Memory Bandwidth**, και μικτά σενάρια **συνδυασμένης παρεμβολής** (CPU + L3 + MemBw).

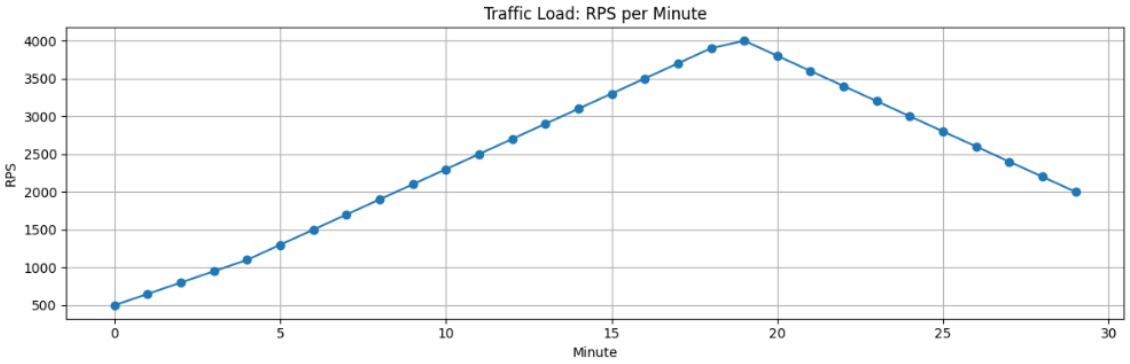

Figure 4.13: Χρονική μεταβολή ρυθμού αιτημάτων

Η αξιολόγηση και του επιπέδου ελέγχου Marla και του Naive Scheduler βασίζεται στις εξής μετρικές:

- **Καθυστέρηση P99 ανά λεπτό**: μέτρο ποιότητας εξυπηρέτησης (QoS) της εφαρμογής.
- **Μέση καθυστέρηση P99**: συνολική απόδοση του controller σε ολόκληρη την διάρκεια του πειράματος.
- **Κατανομή αντιγράφων (replicas)**: τρόπος που το σύστημα διαχειρίζεται την κλιμάκωση.
- **Συνολικοί πόροι που καταναλώθηκαν**: οι συνολικές απαιτήσεις πόρων για κάθε αντίγραφο.

Ετσι, η αρχιτεκτονική του τελικού πειράματος (βλ. 4.14) αποτελείται απο το Minikube Cluster με δύο nodes, τα pods του Nginx, τα παρεμβαλλόμενα pods της σουίτας iBench, το επίπεδο ελέγχου Marla, καθώς και τα εξωτερικά εργαλεία: Vegeta για την παραγωγή αιτημάτων και Interference Injector για την εισαγωγή παρεμβολών.

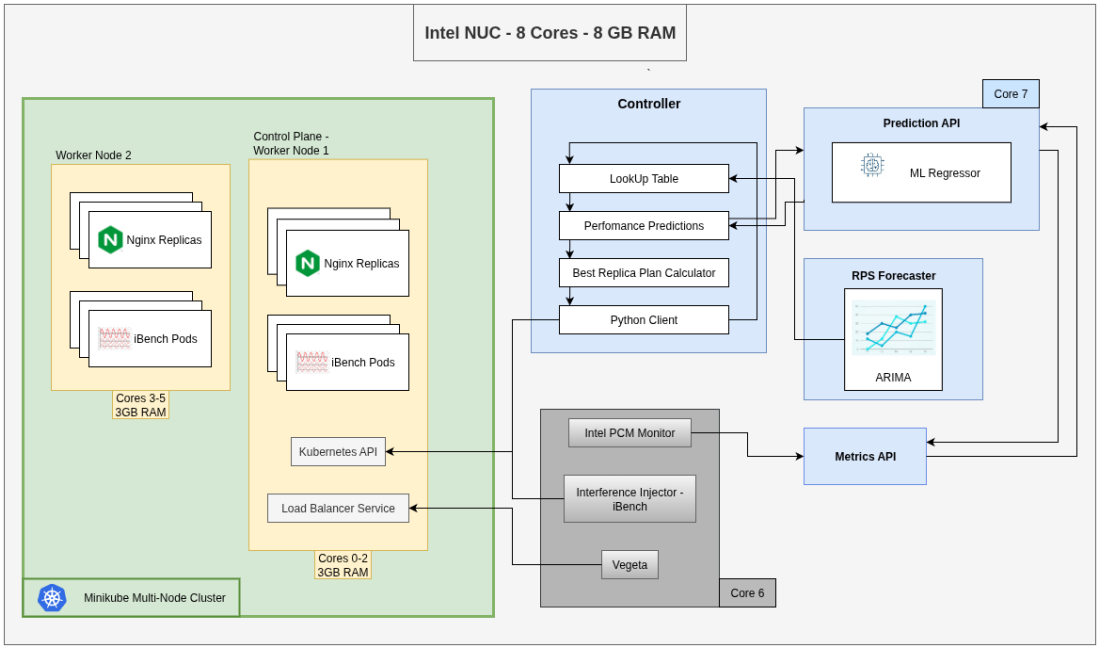

Figure 4.14: Διάταξη της Πειραματικής Αξιολόγησης

4.4.1 Παρεμβολή τύπου CPU

Αρχικά, το επίπεδο ελέγχου Marla δοκιμάστικε σε ένα σενάριο παρεμβολής με την εκτέλεση pods τύπου cpu απο την σουίτα iBench (βλ. 4.15) Κατά τα πρώτα 5 λεπτά δεν δημιουργείται καποια παρεμβολή, ώστε το σύστημα να

σταθεροποιηθεί και ο μηχανισμός πρόβλεψης να μπορεί να εκτιμήσει με ακρίβεια τον επόμενο ρυθμό αιτημάτων. Η ιδια τεχνική θα ακολουθηθεί και στα επόμενα σενάρια παρεμβολής. Στη συνέχεια, τα pods εκκινούν σε τυχαίες χρονικές στιγμές, με τυχαία διάρκεια εκτέλεσης και σε διαφορετικό πλήθος αντιγράφων, δημιουργώντας δυναμικά επίπεδα παρεμβολής σε επίπεδο πυρήνων. Η διαδικασία αυτή υλοποιείται αυτόματα μέσω της συνιστώσας Interference Injector, όπως περιγράφηκε στην αρχιτεκτονική του πειράματος.

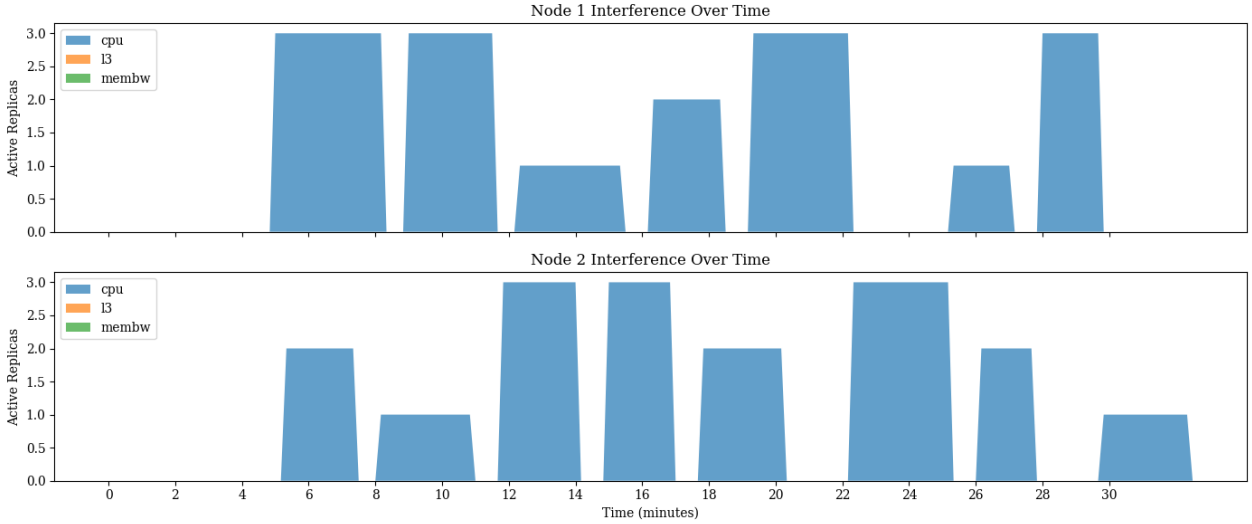

Figure 4.15: Χρονική μεταβολή παρεμβολών CPU ανά node (ενεργά `ibench-cpu` replicas).

Στο Σχ. 4.16 παρουσιάζεται η καθυστέρηση P99 ανά λεπτό, καθώς και η μέση τιμή P99 για κάθε controller. Συνολικά, το επίπεδο ελέγχου `Marla` φαίνεται να διαχειρίζεται καλύτερα την τοποθέτηση των αντιγράφων των nginx pods, ιδίως μετά τα πρώτα 15 λεπτά του πειράματος, παρότι σε ορισμένες περιπτώσεις οι αποφάσεις του δεν οδήγησαν στο βέλτιστο αποτέλεσμα. Η συνολική εικόνα αποτυπώνεται στις μέσες τιμές p99 καθυστέρησης:

- **Naive**: `1.41` ms
- **Marla**: `1.06` ms

που αντιστοιχούν σε **βελτίωση 28.7%** υπέρ του επιπέδου ελέγχου Marla.

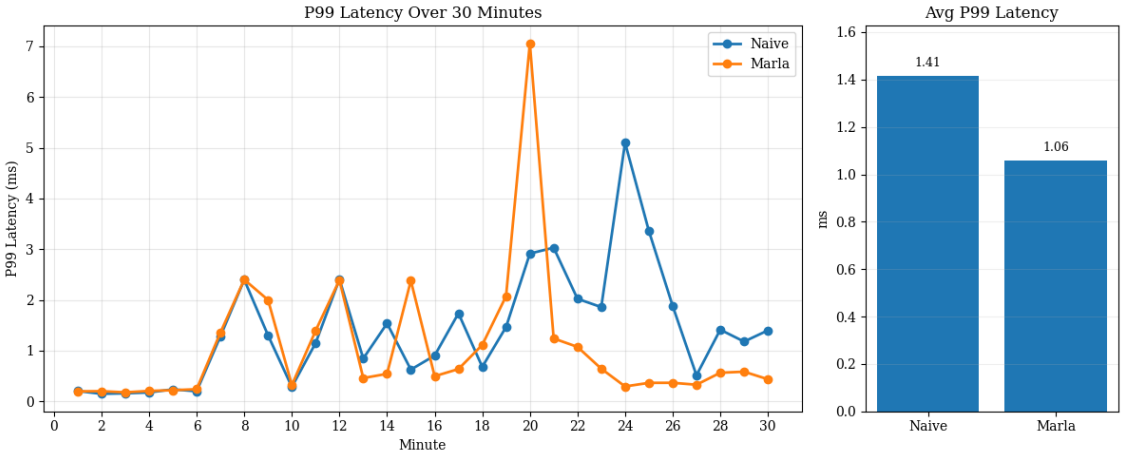

Figure 4.16: Καθυστέρηση P99 ανά λεπτό και μέση τιμή, υπό παρεμβολή CPU.

Στα Σχ. 4.17 και 4.18 φαίνονται αναλυτικά οι αποφάσεις κατανομής αντιγράφων και το ποσοστό της συνολικής χωρητικότητας του Cluster που αυτά δεσμεύουν. Το επίπεδο ελέγχου Marla λαμβάνει συχνότερα αποφάσεις

που συγκεντρώνουν όλα τα replicas σε έναν μόνο κόμβο, ενώ ο Naive Scheduler ακολουθεί μια πιο στατική στρατηγική. Η τελευταία οδηγεί σε έντονη παρεμβολή με τις ibench-cpu εφαρμογές και συνεπώς σε μικρότερη απόδοση, αναφορικά με την p99 καθυστέρηση. Επιπλέον, το Marla λαμβάνει υπόψη και την παρεμβολή που προκαλούν τα ίδια τα αντίγραφα Nginx μεταξύ τους, επιλέγοντας συχνά ως βέλτιστη λύση τη χρήση μικρότερου αριθμού αντιγράφων. Αυτή η προσαρμοστική προσέγγιση εξηγεί και το πλεονέκτημα στην απόδοση που καταγράφηκε στο προηγούμενο γράφημα p99.

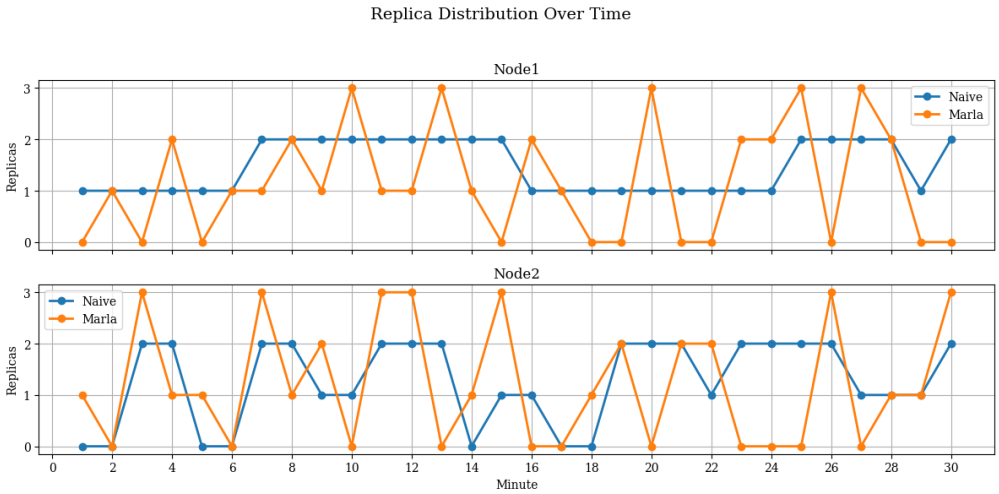

Figure 4.17: Κατανομή αντιγράφων ανά node (χρονική εξέλιξη) υπό CPU παρεμβολή.

Ένα χαρακτηριστικό παράδειγμα φαίνεται στο χρονικό διάστημα 20–22, όταν η παρεμβολή μειώνεται στον Node 1 και αυξάνεται στον Node2. Σε αυτό το διάστημα το επίπεδο ελέγχου Marla προσαρμόζεται άμεσα, τοποθετώντας λιγότερα αντίγραφα στον πιο «καθαρό απο παρεμβολές» κόμβο, ενώ ο Naive επιμένει στην στατική στρατηγική, με αποτέλεσμα να προκαλείται μεγαλύτερη p99 καθυστέρηση.

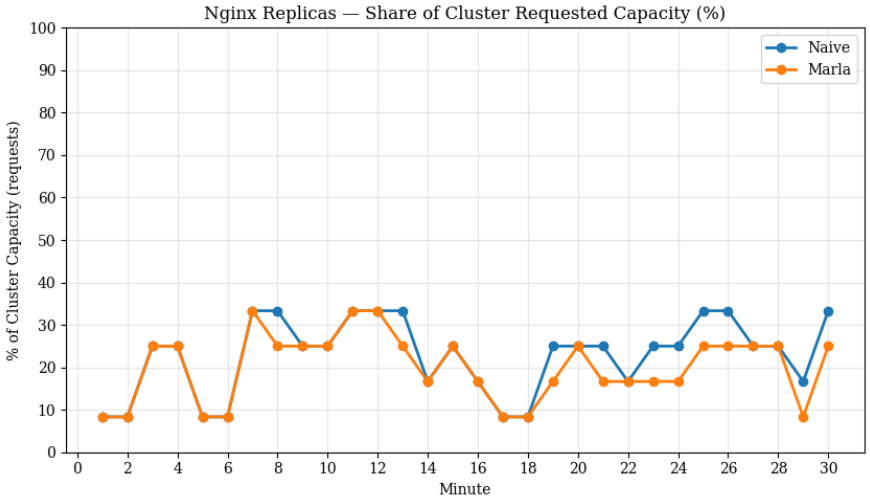

Figure 4.18: Nginx Replicas - Ποσοστό αιτούμενης χωρητικότητας του cluster (%) υπό παρεμβολή CPU.

4.4.2 Παρεμβολή τύπου L3 Cache

Στο επόμενο σενάριο, η παρεμβολή προέρχεται από pods τύπου `ibench-l3`, τα οποία στοχεύουν τη κοινή κρυφή μνήμη τελευταίου επιπέδου. Η χρονική εξέλιξη των παρεμβολών παρουσιάζεται στο Σχ. 4.19.

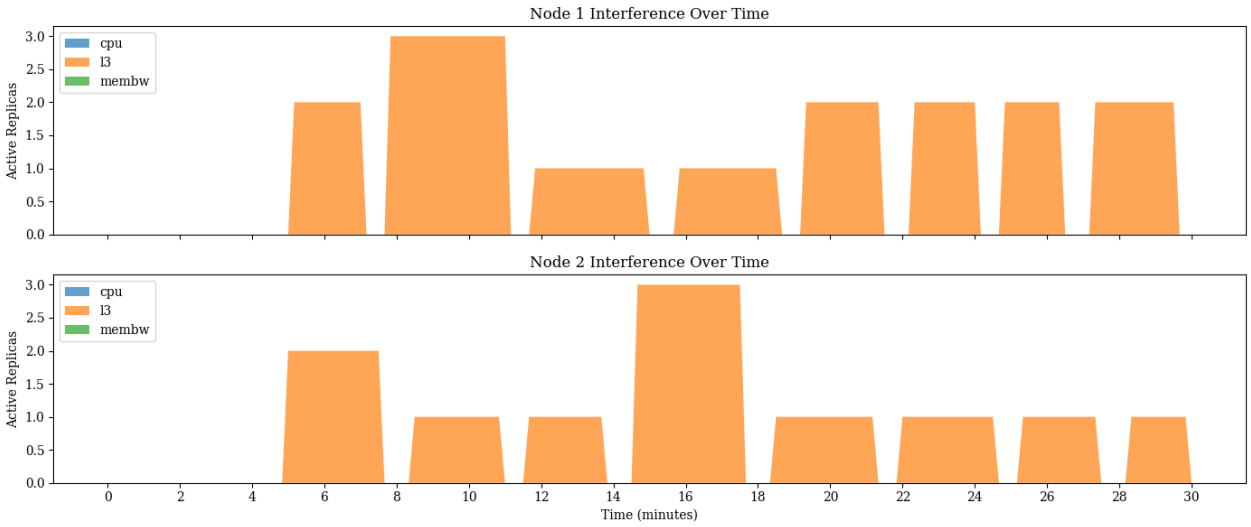

Figure 4.19: Χρονική μεταβολή παρεμβολών L3 ανά node (ενεργά `ibench-l3` replicas).

Η ανάλυση των αποτελεσμάτων σε αυτό το σενάριο παρεμβολής δείχνει ότι, όπως και στο σενάριο CPU, μετά το 15ο λεπτό το επίπεδο ελέγχου Marla επιτυγχάνει καλύτερη απόδοση σε σχέση με τον Naive Scheduler. Μια ενδιαφέρουσα παρατήρηση είναι ότι οι γραφικές των δύο controllers στο γραφημα 4.20 παρουσιάζουν παρόμοια κλίση σε αρκετά σημεία του πειράματος, παρότι η τοποθέτηση των αντιγράφων nginx διαφέρει σημαντικά. Αυτό φανερώνει ότι η συμπεριφορά του συστήματος επηρεάζεται έντονα από τη δυναμική της παρεμβολής L3, η στρατηγική τοποθέτησης ωστόσο εξακολουθεί να παίζει κρίσιμο ρόλο, όπως φανερώνεται στα επόμενα γραφήματα.

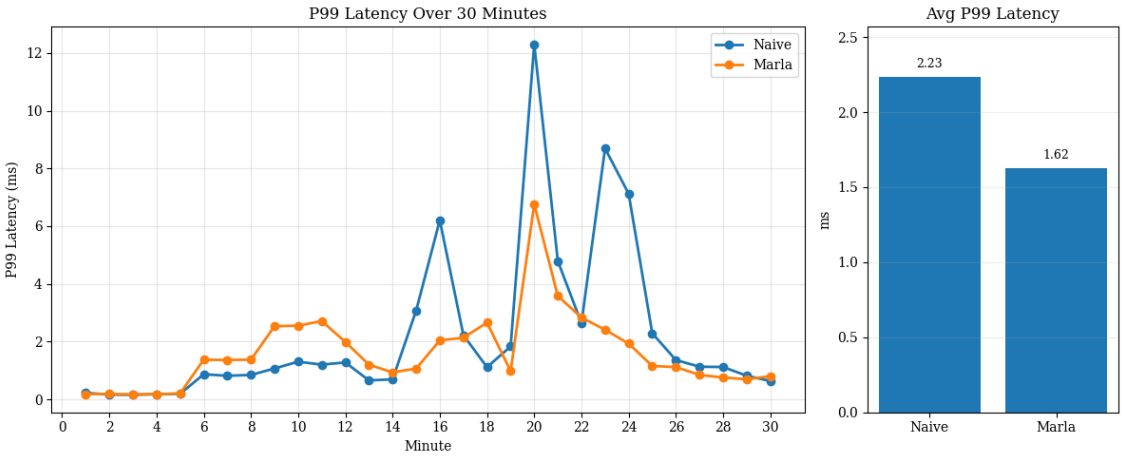

Figure 4.20: Καθυστέρηση P99 ανά λεπτό και μέση τιμή, υπό παρεμβολή L3.

Ιδιαίτερη σημασία έχουν τα λεπτά 16, 20 και 23–24, όπου το επίπεδο ελέγχου Marla αποδίδει εμφανώς καλύτερα. Το πλεονέκτημα αυτό οφείλεται σε δύο παράγοντες. Πρώτον, το Marla αναγνωρίζει και την παρεμβολή που προκαλούν μεταξύ τους τα ίδια τα αντίγραφα Nginx και συχνά αποφασίζει τη χρήση μικρότερου αριθμού αντιγράφων. Δεύτερον, αντιδρά πιο ευέλικτα στη μετακίνηση της παρεμβολής. Συγκεκριμένα, στο διάστημα 14-16, καθώς η L3 παρεμβολή «μεταφέρεται» από τον Node 1 στον Node 2, το επίπεδο ελέγχου Marla επιλέγει περισσότερο τον Node1 για την τοποθέτηση των αντιγράφων, ενώ ο Naive διατηρεί αντίγραφα και στους δύο κόμβους (με έμφαση φυσικά στον Node 1 λόγω των resource requests). Αυτή η διαφορά στρατηγικής εξηγεί και την υπεροχή του Marla στην τελική μέση τιμή της p99 καθυστέρησης, στο σύνολο του πειράματος:

- **Naive**: `2.23` ms

- **Marla**: 1.62 ms

που αντιστοιχεί σε **βελτίωση 27.4%**.

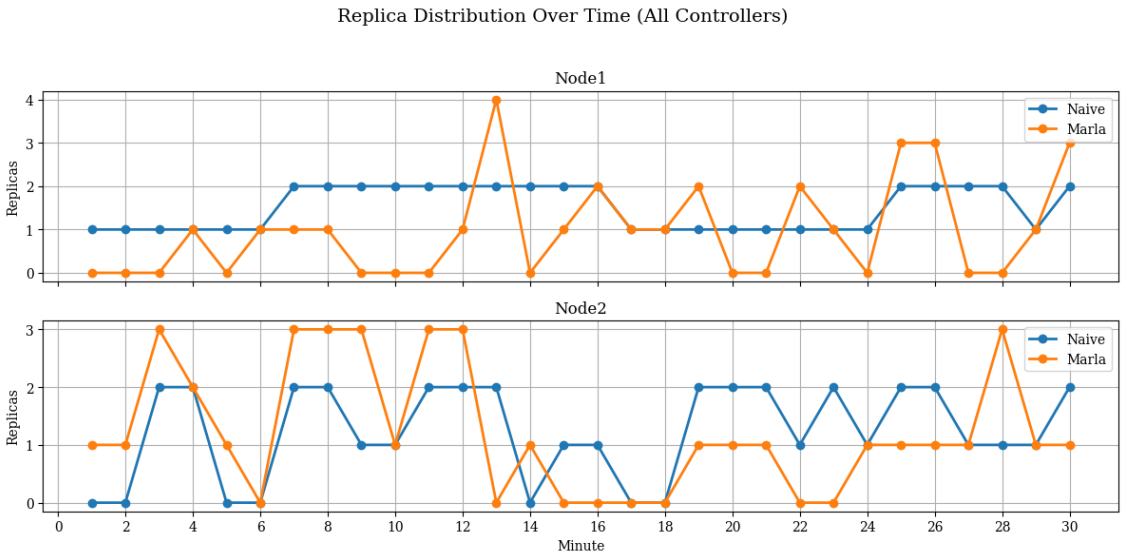

Figure 4.21: Κατανομή αντιγράφων ανά node (χρονική εξέλιξη) υπό L3 παρεμβολή.

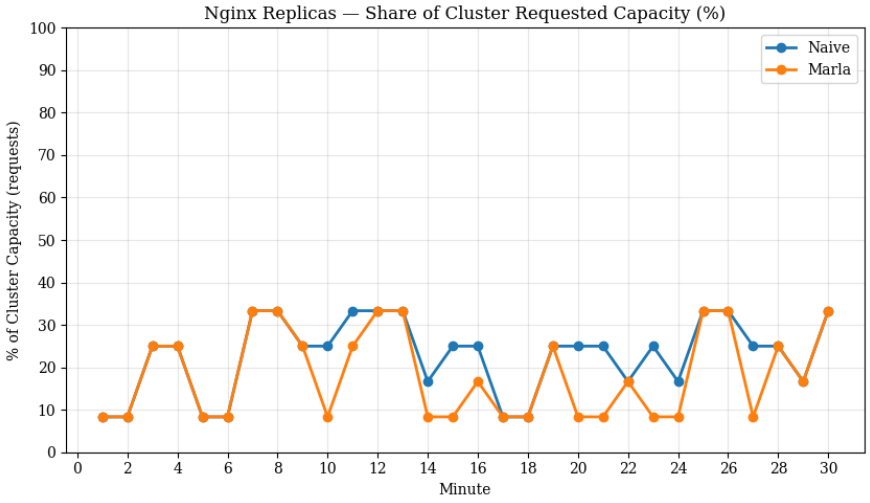

Figure 4.22: Nginx Replicas - Ποσοστό αιτούμενης χωρητικότητας του cluster (%) υπό παρεμβολή L3.

4.4.3 Παρεμβολή τύπου Memory Bandwidth

Στο τελευταίο σενάριο με έναν τύπο παρεμβολής, ο Cluster υποβάλλεται σε παρεμβολή που στοχεύει το **Memory Bandwidth**. Τα ibench-membw pods ενεργοποιούνται και τερματίζονται τυχαία σε κάθε node, προκαλώντας ανταγωνισμό στον κοινό δίαυλο επικοινωνίας από και προς τη μνήμη, επηρεάζοντας όλους τους πυρήνες. Στα πειράματα της Ενότητας 4.2 φανερώθηκε ότι αυτή η μορφή παρεμβολής έχει τη μικρότερη επίδραση στα αντίγραφα του Nginx σε σχέση με τις υπόλοιπες (CPU, L3).

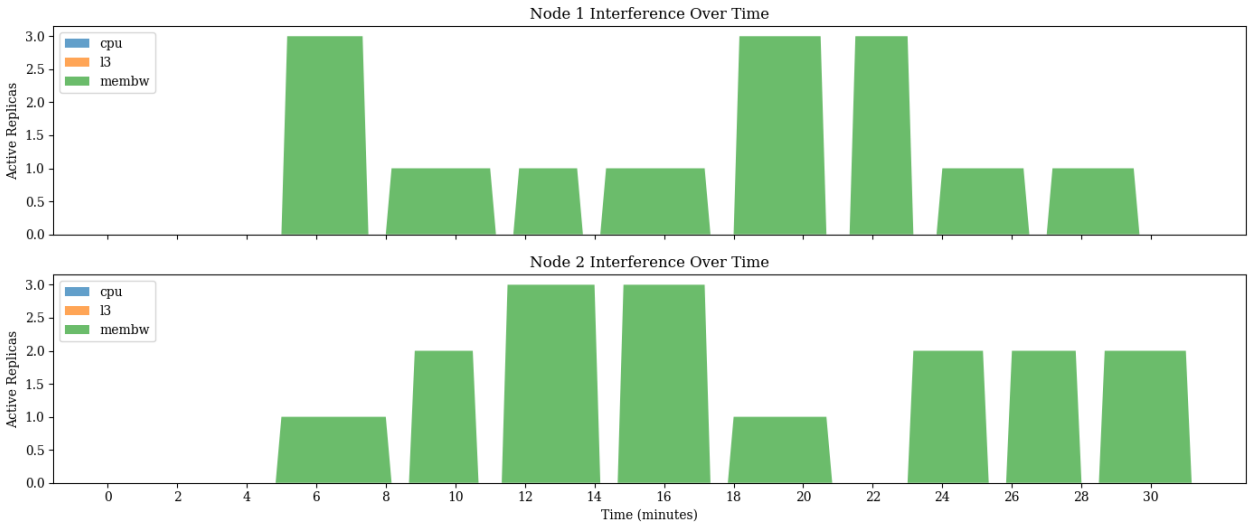

Figure 4.23: Χρονική μεταβολή παρεμβολών Memory Bandwidth ανά node (ενεργά `ibench-membw` replicas)

Στο Σχ. 4.24 παρουσιάζεται η καθυστέρηση p99 ανά λεπτό και η μέση τιμή της. Η υλοποίηση Marla πέτυχε τη μεγαλύτερη συνολική βελτίωση έναντι του Naive Scheduler, με μείωση της μέσης p99 καθυστέρησης κατά **34.9%**. Συγκεκριμένα οι μέσες τιμές p99 καθυστέρησης ήταν:

- **Naive**: `1.04` ms
- **Marla**: `0.73` ms

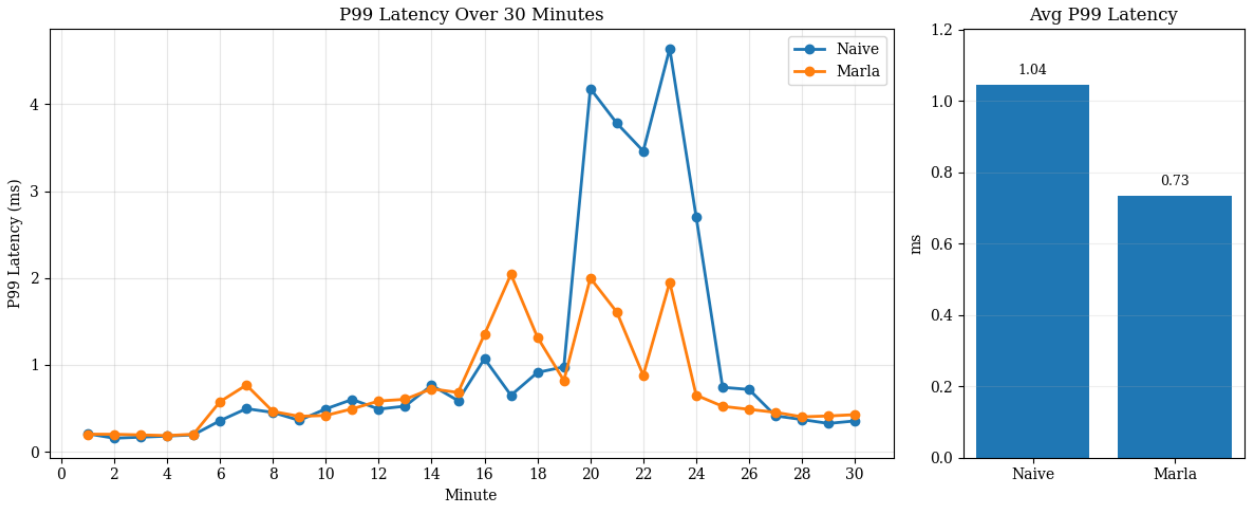

Figure 4.24: Καθυστέρηση P99 ανά λεπτό και μέση τιμή, υπό παρεμβολή Memory Bandwidth

Σε αυτό το πείραμα, αξίζει να αναλυθεί το διάστημα είναι μεταξύ των λεπτών **19**–**26**, όπου οι καμπύλες των δύο schedulers έχουν μεν παρόμοια κλίση, αλλά με τεράστια διαφορά στις τιμές της p99 καθυστέρησης. Από το χρονοδιάγραμμα παρεμβολής φαίνεται ότι σε αυτό το διάστημα η παρεμβολή στον Node 2 μειώνεται ενω ταυτόγχρονα αυξάνεται στον Node 1. Η υλοποίηση Marla ακολουθεί αυτή τη μεταβολή, προσαρμόζοντας την τοποθέτηση των αντιγράφων στον κόμβο με την μικρότερη επιβάρυνση, ενώ ο Naive Scheduler αδυνατεί να προσαρμοστεί σε πραγματικό χρόνο. Επιπλέον, απο το διαγραμμα 4.26 παρατηρείται οτι μόνο στο 50% του διαστήματος αυτού το επίπεδο ελέγχου Marla χρησιμοποιεί λιγότερους συνολικούς πόρους ενώ η βελτιστοποίηση της απόδοσης προκύπτει από την έξυπνη τοποθέτηση.

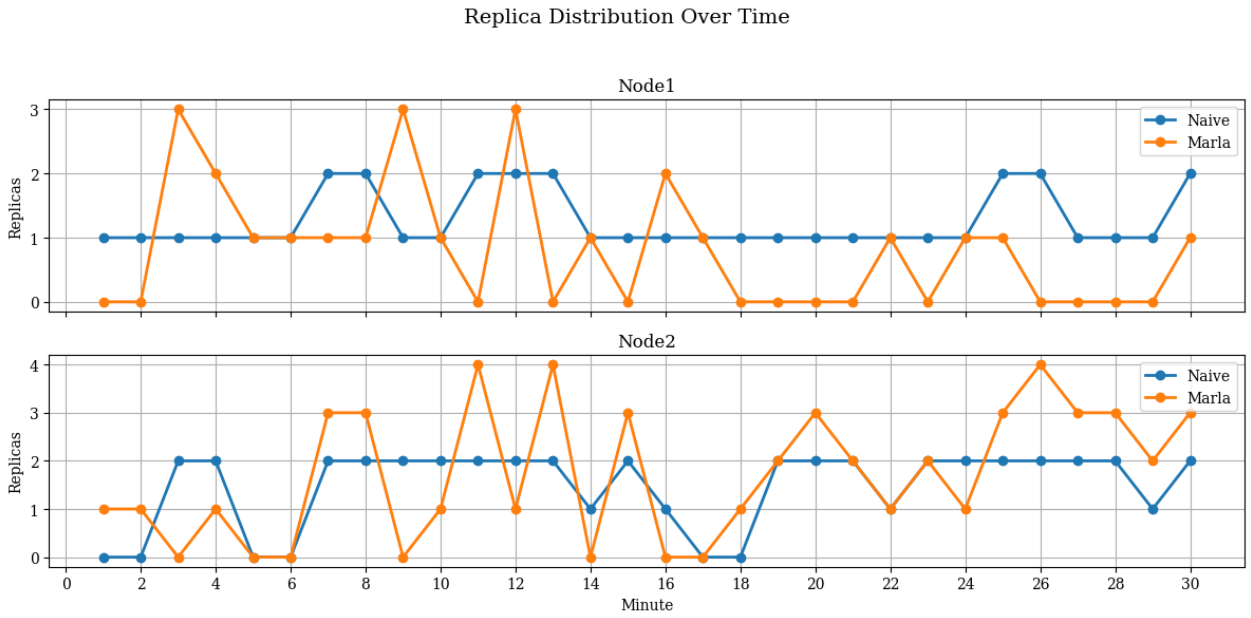

Figure 4.25: Κατανομή αντιγράφων ανά node (χρονική εξέλιξη) υπό Memory Bandwidth παρεμβολή

Ακόμα, στο διάστημα **8–14 λεπτών**, ο ελεγκτής Marla πραγματοποιεί πολλές μαζικές επανατοποθετήσεις των αντιγράφω, όπως φαίνεται απο το Σχ. 4.25. Παρά αυτή την φαινομενικά ασταθή τοποθέτηση, η καθυστέρηση p99 εμφανίζει χαμηλότερες τιμές, σε σχέση με τον Naive Scheduler, αποδεικνύοντας οτι η αρχιτεκτονική αυτή μπορεί να προσαρμοστεί ακόμα και σε περιπτώσεις απότομης αλλαγής της παρεμβολής.

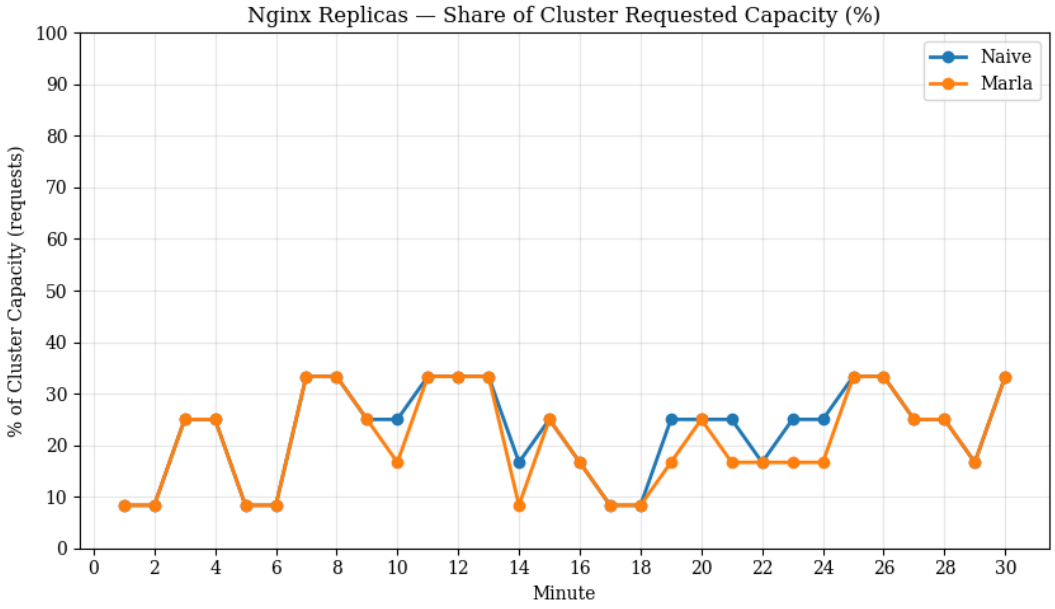

Figure 4.26: Nginx Replicas — Ποσοστό αιτούμενης χωρητικότητας του cluster (%) υπο Memory Bandwidth παρεμβολή

4.4.4 Συνδυαστική Παρεμβολή

Στο τελευταίο πείραμα χρησιμοποιήθηκαν Μικτά Σενάρια Παρεμβολής, τα οποία αποτελούν και την πιο ρεαλιστική κατηγορία σεναρίων. Σε αυτή την περίπτωση, αξιοποιούνται εφαρμογές της σουίτας iBench που συνδυάζουν

όλους τους τύπους παρεμβολής, δηλαδή CPU, L3 Cache και Memory Bandwidth. Η ανάπτυξη των iBench εφαρμογών γίνεται σε τυχαίες χρονικές στιγμές και σε τυχαίο αριθμό αντιγράφων, με αποτέλεσμα η συνολική παρεμβολή να μεταβάλλεται συνεχώς απρόβλεπτα, όπως φαίνεται στο Σχ. 4.27.

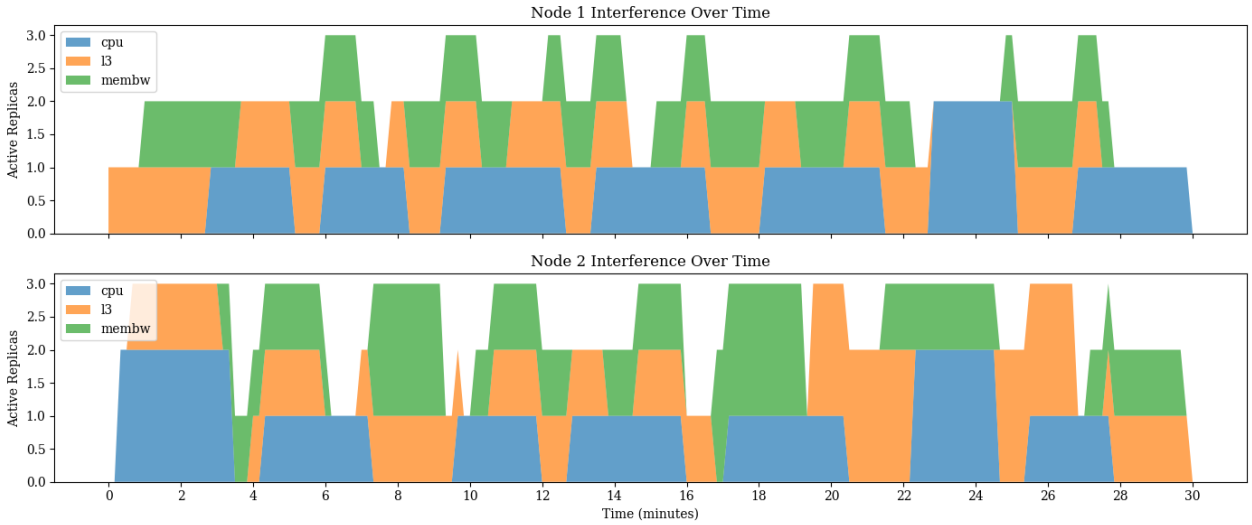

Figure 4.27: Χρονική μεταβολή μικτών παρεμβολών (CPU, L3, MemBw) ανά node.

Στο Σχ. 4.28 παρουσιάζεται η καθυστέρηση P99 ανά λεπτό, καθώς και η μέση τιμή της. Σε ορισμένα χρονικά διαστήματα το Επίπεδο Ελέγχου Marla αποδίδει σαφώς καλύτερα από τον Naive Scheduler, ενώ σε άλλα οι δύο μηχανισμοί έχουν παρόμοια συμπεριφορά, με τις καμπύλες απόδοσης τους να εμφανίζουν ίδια κλίση και παραπλήσιες τιμές. Για να εξηγηθεί υπεροχή του Marla θα πρέπει να γίνει βαθύτερη αναλυση σε συγκεκριμένα τμήματα του πειράματος.

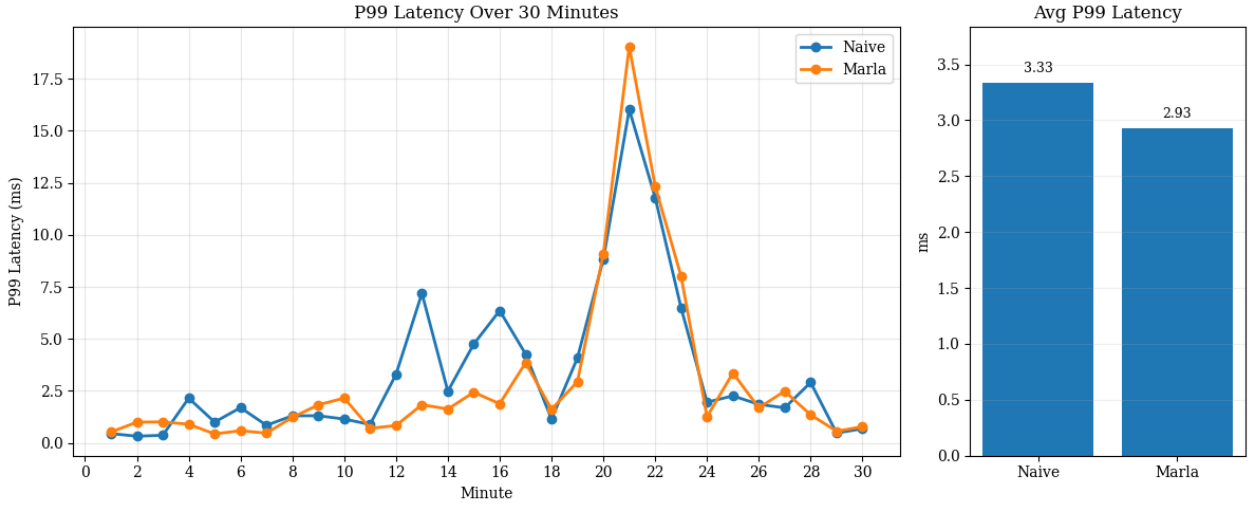

Figure 4.28: Καθυστέρηση P99 ανά λεπτό και μέση τιμή, υπό μικτή παρεμβολή.

Ένα ιδιαίτερα κρίσιμο διάστημα είναι τα λεπτά 12 έως 17. Σε αυτό το διάστημα, ο Naive Scheduler ακολουθεί τη στατική στρατηγική του, μοιράζοντας τα αντίγραφα με ίσο τρόπο στους δύο κόμβους, καθώς το σύνολο των αντιγράφων των εφαρμογών παρεμβολής είναι ίδιο. Ωστόσο, οι παρεμβολές δεν έχουν την ίδια βαρύτητα, καθώς στον Node1 παρατηρούνται περισσότερες L3 παρεμβολές, οι οποίες, όπως είδαμε στην ενότητα 4.2, επηρεάζουν έντονα την απόδοση των Nginx pods και στον Node 2 οι περισσότερες παρεμβολές είναι τύπου Memory Bandwidth, με μικρότερη επίδραση.

Το Επίπεδο Ελέγχου Marla μπορεί να αναγνωρίσει αυτή τη διαφορά και είτε κατανέμει ισομερώς τα αντίγραφα, είτε τα τοποθετεί εξ ολοκλήρου στον Node 2, αποφεύγοντας τον «επικίνδυνο» κόμβο. Αξίζει επίσης να σημειωθεί ότι μόνο για περίπου το μισό αυτής της περιόδου το Marla χρησιμοποιεί λιγότερους συνολικούς πόρους. Το μεγαλύτερο όφελος προκύπτει από την έξυπνη τοποθέτηση, αποδεικνύοντας ότι η τοποθέτηση είναι εξίσου σημαντική με τον απόφαση του συνολικό αριθμό αντιγράφων.

Το αποτέλεσμα αυτού του χαρακτηριστικού αποτυπώνεται καλύτερα στην τελική μέση τιμή της P99 καθυστέρησης:

- **Naive**: 3.33 ms
- **Marla**: 2.93 ms

που αντιστοιχεί σε **βελτίωση 12.0%**.

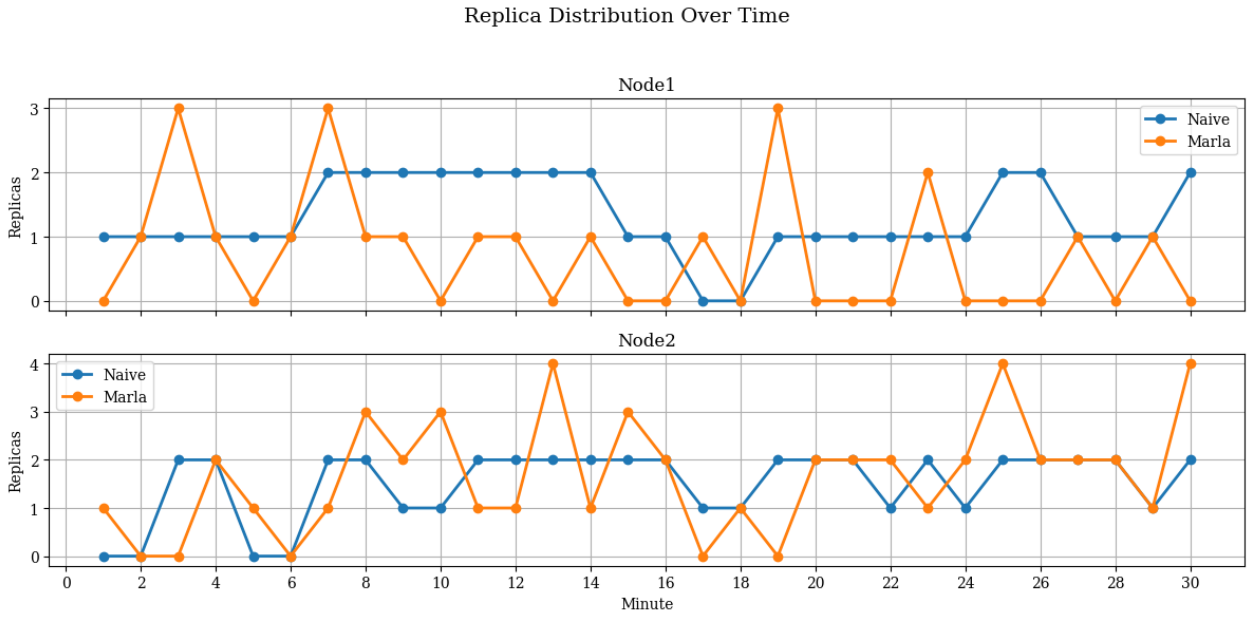

Figure 4.29: Κατανομή αντιγράφων ανά node (χρονική εξέλιξη) υπό μικτή παρεμβολή.

Εξίσου ενδιαφέρον είναι το διάστημα 20 εως 22. Εκεί, το Marla χρησιμοποιεί μικρότερο αριθμο αντιγράφων σε σχέση με τον Naive, όμως η απόδοση παραμένει σχεδόν ίδια. Όπως αναφέρθηκε στην ενότητα 4.2, σε συνθήκες έντονης παρεμβολής, υπάρχει ένα κατώφλι απόδοσης που δεν μπορεί να ξεπεραστεί μόνο με την αύξηση των replicas. Σε τέτοιες περιπτώσεις, το Marla επιλέγει τη βέλτιστη λύση αναφορικά με την χρήση των πόρων του cluster, πετυχαίνοντας ετσι παρόμοιο QoS με μικρότερο υπολογιστικό κόστος.

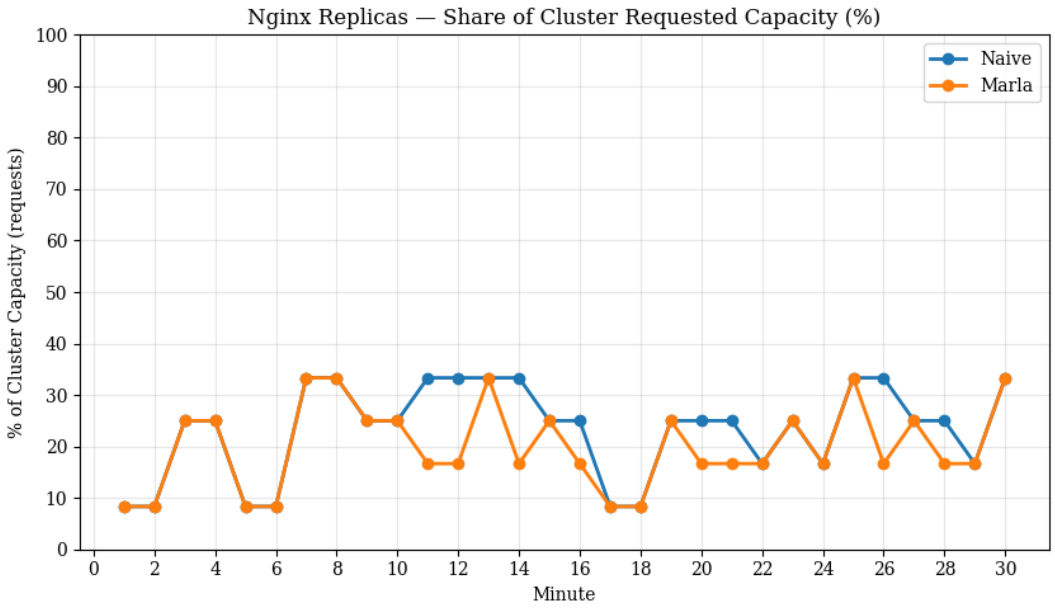

Figure 4.30: Nginx Replicas — Ποσοστό αιτούμενης χωρητικότητας του cluster (%).

Επιπρόσθετα μικτά σενάρια. Καθώς τα σενάρια μικτής παρεμβολής αποτελούν και την πιο ρεαλιστική κατηγορία σεναρίων, εκτελέστηκαν και πρόσθετα πειράματα (AS1/AS2/AS4/AS5/AS3). Ο Πίν. 4.5 συνοψίζει τα αποτελέσματα (μέση p99 σε 30 min) και δείχνει συνεπή υπεροχή του Marla.

Table 4.5: Πειραματική Αξιολόγηση σε Μικτά σενάρια παρεμβολής

Σενάριο (run)	p_{99} **Naive [ms]**	p_{99} **Marla [ms]**	**Μείωση [%]**
Mixed-AS0	3.33	2.93	**12.00**
Mixed-AS1	3.333	2.812	**16.95**
Mixed-AS2	3.160	2.406	**27.11**
Mixed-AS3	2.903	2.430	**17.76**
Mixed-AS4	3.262	2.724	**17.97**
Mixed-AS5	2.128	1.829	**15.15**
Μέσος όρος	*3.02*	*2.52*	***17.8***

Συνοψίζοντας, τα αποτελέσματα όλων των σεναρίων παρεμβολής συνοψίζονται στον Πίνακα 4.6. Παρατηρείται ότι ο ελεγκτής Marla υπερτερεί σταθερά έναντι του Naive Scheduler. Μια πιο αναλυτική ερμηνεία των αποτελεσμάτων αυτών παρουσιάζεται στο Κεφάλαιο 5.

Σενάριο	**Μέση** p_{99} **(Naive) [ms]**	**Μέση** p_{99} **(Marla) [ms]**	**Μείωση μέσης** p_{99} **[%]**
CPU	1.41	1.06	**28.7**
L3 Cache	2.23	1.62	**27.4**
Memory Bandwidth	1.04	0.73	**34.9**
Mixed	3.02	2.43	**17.8**

Table 4.6: Σύνοψη αποτελεσμάτων ανά σενάριο παρεμβολής

Για λόγους διαφάνειας και επαναληψιμότητας, όλη η υλοποίηση, τα σενάρια πειραμάτων καθώς και το σύνολο δεδομένων που χρησιμοποιήθηκε είναι διαθέσιμα στο GitHub repository: https://github.com/georgakopoulosgeo/Interference.

Chapter 5

Συμπεράσματα και Μελλοντικές Προεκτάσεις

5.1 Συμπεράσματα

Η πειραματική αξιολόγηση επιβεβαιώνει την βασική υπόθση αυτής της εργασίας, δηλαδή οτι οι αποφάσεις τοποθέτησεις των Pods σε ένα Kubernetes Cluster δεν πρεπει να βασίζονται αποκλειστικά σε στατικά ορια αλλά οφείλουν να λαμβάνουν υπόψην τους δυναμικούς παράγοντες της παρεμβολής και της κυκλοφορίας. Συγκεκριμένα, για το φαινόμενο της παρεμβολής απόδοσης (Performance Interference), μέσα απο την πρώτη φάση πειραμάτων αποτυπώθηκε η μη γραμμική και πολύπλοκη του φύση. Επιπλέον κατάφέραμε να ποσοτικοποιήσουεμ το φαινόμενο, με βάση τις μετρικές υλικού, και να εκπαιδεύσουμε ένα μοντέλο παλινδρόμησης XGBoost με συντελεστή προσαρμογής $R^2 = 0.90$, ικανό να προβλέψει με ικανοποιητική ακρίβεια την υποβάθμιση της απόδοσης της εφαρμογής υπό διαφορετικές συνθήκες παρεμβολής και κυκλοφορίας.

Εισάγοντας αυτό το μοντέλο στην υλοποίηση της προτεινόμενης αρχιτεκτονικής, τα αποτελέσματα δείχνουν οτι ο ελεγκτής Marla, πετυχαίνει τον διπλό στόχο της βελτιστοποίησης τόσο της απόδοσης όσο και της αξιοποίησης των πόρων. Σε όλα τα σενάρια παρεμβολής πέτυχε καλύτερη μέση καθυστέρηση p99, σημειώνοντας βελτίωση μεταξύ 17.8% και 34.9% έναντι στο προκαθορισμένο Scheduler του Kubernetes. Ταυτόγχρονα, ανιχνεύοντας την παρεμβολή που τα ιδια τα αντιγραφα της εφαρμογής δημιουργούν, προσάρμοσε τον συνολικό αριθμό τους πετυχαίνοντας διατήρηση του QoS με λιγότερη επιβάρυνση των πόρων. Συνεπώς, η παρούσα εργασία αποδεικνύει ότι η ενσωμάτωση προγνωστικών μοντέλων σε συστήματα ελέγχου είναι ιδιαίτερα κρίσιμη για πιο έξυπνη και αποδοτική ενορχήστρωση μικροϋπηρεσιών.

5.2 Μελλοντικές Προεκτάσεις

Η ανάλυση, οι παρατηρήσεις και η μεθοδολογία που προτείνεται σε αυτή την εργασία αποτελούν μια πρώτη προσπάθεια να περιγραφεί, να ποσοτικοποιηθεί και να εντοπιστεί το φαινόμενο της παρεμβολής σε περιβάλλον Kubernetes. Παρότι τα πειραματικά αποτελέσματα δείχνουν ότι η προτεινόμενη αρχιτεκτονική είναι λειτουργική και αποδοτική, κρίνονται αναγκαίες περαιτέρω δοκιμές σε πραγματικό cluster μεγαλύτερης κλίμακας από το Minikube. Μόνο με αυτό τον τρόπο μπορεί να αξιολογηθεί η επεκτασιμότητα της αρχιτεκτονικής σε συνθήκες κοντινότερες προς την παραγωγή.

Επιπλέον, αν επιθυμούσαμε να αναπτύξουμε περισσότερο την παραπάνω προτεινόμενη αρχιτεκτονική, θα μπορούσαμε ενσωματώσουμε τεχνικές ενισχυτικής μάθησης (Reinforcement Learning). Με την βοήθεια ενός RL agent θα μπορούσαμε να πετύχουμε: (α) online learning, ώστε το σύστημα να προσαρμόζεται διαρκώς σε νέες συνθήκες χωρίς ανάγκη εκ νέου εκπαίδευσης του μοντέλου, καθώς και (β) scheduling σε multi-tenant περιβάλλοντα, όπου πολλαπλές εφαρμογές συνυπάρχουν και ανταγωνίζονται για κοινούς πόρους. Με τον τρόπο αυτό, η λογική που εφαρμόστηκε εδώ για μία μόνο latency-critical εφαρμογή μπορεί να επεκταθεί σε πιο σύνθετα σενάρια, εδραιώνοντας τον ρόλο των interference aware μηχανισμών σε όλα τα συστήματα ορχήστρωσης containerized εφαρμογών.

Chapter 6

Βιβλιογραφία

[1] Eurostat. *Cloud computing - statistics on the use by enterprises*. Accessed: 20 September 2025. 2023.

[2] Koh, Y. et al. "An analysis of performance interference effects in virtual environments". In: *IEEE International Symposium on Performance Analysis of Systems & Software*. IEEE. 2007, pp. 200–209.

[3] Lin, W. et al. "Performance Interference of Virtual Machines: A Survey". In: *ACM Computing Surveys* 55.12 (Mar. 2023), 254:1–254:37. DOI: 10.1145/3573009.

[4] Intel Corporation. *What is Hyper-Threading?* Accessed: 2025-09-20. 2023. URL:

[5] Pu, X. et al. "Understanding performance interference of I/O workload in virtualized cloud environments". In: *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*. Los Alamitos, CA, USA: IEEE, 2010, pp. 51–58. DOI: 10.1109/CLOUD.2010.41.

[6] Blagodurov, S. et al. "A case for NUMA-aware contention management on multicore systems". In: *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. ACM, 2010, pp. 557–558.

[7] Aghilinasab, H. et al. "Dynamic memory bandwidth allocation for real-time GPU-based SoC platforms". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.11 (2020), pp. 3348–3360. DOI: 10.1109/TCAD.2020.2997768.

[8] Garrido, L. A. and Carpenter, P. "vMCA: Memory capacity aggregation and management in cloud environments". In: *Proceedings of the 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*. Los Alamitos, CA, USA: IEEE, 2017, pp. 674–683. DOI: 10.1109/ICPADS.2017.00092.

[9] Huang, D. "Managing IO Resource for Co-Running Data Intensive Applications in Virtual Clusters". PhD thesis. College of Engineering and Computer Science, University of Central Florida, 2012.

[10] Yuan, Y. et al. "On interference-aware provisioning for cloud-based big data processing". In: *Proceedings of the 2013 IEEE/ACM 21st International Symposium on Quality of Service (IWQoS)*. Los Alamitos, CA, USA: IEEE, 2013, pp. 1–6. DOI: 10.1109/IWQoS.2013.6550276.

[11] Barker, S. K. and Shenoy, P. "Empirical evaluation of latency-sensitive application performance in the cloud". In: *Proceedings of the 1st Annual ACM SIGMM Conference on Multimedia Systems*. ACM, 2010, pp. 35–46. DOI: 10.1145/1730836.1730843.

[12] Pu, X. et al. "Who is your neighbor: Net I/O performance interference in virtualized clouds". In: *IEEE Transactions on Services Computing* 6.3 (2012), pp. 314–329. DOI: 10.1109/TSC.2011.31.

[13] Delimitrou, C. and Kozyrakis, C. "Quasar: resource-efficient and QoS-aware cluster management". In: *SIGPLAN Not.* 49.4 (Feb. 2014), pp. 127–144. ISSN: 0362-1340. DOI: 10.1145/2644865.2541941. URL:

[14] Qiu, H. et al. "FIRM: An Intelligent Fine-grained Resource Management Framework for SLO-Oriented Microservices". In: *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 805–825. ISBN: 978-1-939133-19-9. URL:

[15] Lin, W. et al. "Performance Interference of Virtual Machines: A Survey". In: *ACM Comput. Surv.* 55.12 (Mar. 2023). ISSN: 0360-0300. DOI: 10.1145/3573009. URL:

[16] Sun, X. et al. "MVEI: An Interference Prediction Model for CPU-intensive Application in Cloud Environment". In: *2014 13th International Symposium on Distributed Computing and Applications to Business, Engineering and Science*. 2014, pp. 83–87. DOI: 10.1109/DCABES.2014.21.

[17] Tzenetopoulos, A. et al. "Interference-Aware Orchestration in Kubernetes". In: *High Performance Computing*. Ed. by H. Jagode et al. Cham: Springer International Publishing, 2020, pp. 321–330. ISBN: 978-3-030-59851-8.

[18] Cheng, Y. et al. "Precise contention-aware performance prediction on virtualized multicore system". In: *Journal of Systems Architecture* 72 (2017), pp. 42–50.

[19] Zhang, X. et al. "CPI2: CPU performance isolation for shared compute clusters". In: *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys)*. ACM, 2013, pp. 379–392.

[20] Delimitrou, C. and Kozyrakis, C. "iBench: Quantifying Interference for Datacenter Workloads". In: *Proceedings of the 2013 IEEE International Symposium on Workload Characterization (IISWC)*. Benchmark suite for interference injection across shared resources. Portland, OR, USA, Sept. 2013.

[21] Mars, J., Tang, L., and Soffa, M. L. "Directly characterizing cross core interference through contention synthesis". In: *6th International Conference on High Performance and Embedded Architectures and Compilers (HiPEAC)*. ACM, 2011, pp. 167–176.

[22] Govindan, S. et al. "Cuanta: Quantifying effects of shared on-chip resource interference for consolidated virtual machines". In: *Proceedings of the 2nd ACM Symposium on Cloud Computing (SoCC)*. ACM, 2011, 22:1–22:14.

[23] Barve, Y. D. et al. "FECBench: A holistic interference-aware approach for application performance modeling". In: *Proceedings of the 2019 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2019, pp. 211–221.

[24] Taheri, J., Latif, S., and Malik, S. "vmBBProfiler: A Black-Box Profiling Approach to Quantify Sensitivity of Virtual Machines to Shared Cloud Resources". In: *Computing*. Vol. 99. 12. Black-box profiling, historical logging of VM behavior. 2017, pp. 1149–1177.

[25] Intel Corporation. *Intel Performance Counter Monitor (Intel PCM)*. Accessed: 2025-09-23. 2025.

[26] Linux Kernel Organization. *perf: Linux profiling with performance counters*. Accessed: 2025-09-23.

[27] Google Open Source. *cAdvisor (Container Advisor)*. Accessed: 2025-09-23.

[28] Prometheus Authors. *Prometheus: Monitoring system and time series database*. Accessed: 2025-09-23.

[29] Buchaca, D. et al. "Sequence-to-sequence models for workload interference prediction on batch processing datacenters". In: *Future Generation Computer Systems* 110 (2020), pp. 155–166.

[30] Kim, S.-G., Eom, H., and Yeom, H. Y. "Virtual machine consolidation based on interference modeling". In: *The Journal of Supercomputing* 66.3 (2013), pp. 1489–1506. DOI: 10.1007/s11227-013-0939-2. URL:

[31] Chen, L., Shen, H., and Platt, S. "Cache contention aware Virtual Machine placement and migration in cloud datacenters". In: *2016 IEEE 24th International Conference on Network Protocols (ICNP)*. 2016, pp. 1–10. DOI: 10.1109/ICNP.2016.7784447.

[32] Ludwig, U. L. et al. "Optimizing multi-tier application performance with interference and affinity-aware placement algorithms". In: *Concurrency and Computation: Practice and Experience* 31.18 (2019). e5098 cpe.5098, e5098. DOI: https://doi.org/10.1002/cpe.5098. eprint: URL:

[33] Meyer, V. et al. "An Interference-Aware Application Classifier Based on Machine Learning to Improve Scheduling in Clouds". In: *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. 2020, pp. 80–87. DOI: 10.1109/PDP50117.2020.00019.

[34] Horchulhack, P. et al. "Detection of quality of service degradation on multi-tenant containerized services". In: *Journal of Network and Computer Applications* 224 (2024), p. 103839. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2024.103839. URL:

[35] Baluta, A., Mukherjee, J., and Litoiu, M. "Machine Learning based Interference Modelling in Cloud-Native Applications". In: *Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering*. ICPE '22. Beijing, China: Association for Computing Machinery, 2022, pp. 125–132. ISBN: 9781450391436. DOI: 10.1145/3489525.3511677. URL:

[36] Masouros, D., Xydis, S., and Soudris, D. "Rusty: Runtime Interference-Aware Predictive Monitoring for Modern Multi-Tenant Systems". In: *IEEE Transactions on Parallel and Distributed Systems* 32.1 (2021), pp. 184–198. DOI: 10.1109/TPDS.2020.3013948.

[37] Buchaca, D. et al. "Sequence-to-sequence models for workload interference prediction on batch processing datacenters". In: *Future Generation Computer Systems* 110 (2020), pp. 155–166. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2020.03.058. URL:

[38] Li, J. et al. "dCCPI-predictor: A state-aware approach for effectively predicting cross-core performance interference". In: *Future Generation Computer Systems* 105 (2020), pp. 184–195. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2019.11.016. URL:
[39] The Kubernetes Authors. *Kubernetes Documentation*. Accessed: 2025-09-23. 2024.
[40] Kubernetes Authors. *Kubernetes Scheduler*. Accessed: 2025-09-14. 2025.
[41] Burns, B. et al. "Borg, omega, and kubernetes". In: *Communications of the ACM* 59.5 (2016), pp. 50–57.
[42] Delimitrou, C. and Kozyrakis, C. "Paragon: QoS-aware scheduling for heterogeneous datacenters". In: *SIGPLAN Not.* 48.4 (Mar. 2013), pp. 77–88. ISSN: 0362-1340. DOI: 10.1145/2499368.2451125. URL:
[43] Yang, H. et al. "Bubble-Flux: Precise online QoS management for increased utilization in warehouse scale computers". In: *ACM SIGARCH Computer Architecture News*. Vol. 41. 3. ACM, 2013, pp. 607–618.
[44] Romero, F. and Delimitrou, C. "Mage: online and interference-aware scheduling for multi-scale heterogeneous systems". In: *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*. PACT '18. Limassol, Cyprus: Association for Computing Machinery, 2018. ISBN: 9781450359863. DOI: 10.1145/3243176.3243183. URL:
[45] Luo, S. et al. "Erms: Efficient Resource Management for Shared Microservices with SLA Guarantees". In: *Association for Computing Machinery*. ASPLOS 2023. Vancouver, BC, Canada, 2022, pp. 62–77. ISBN: 9781450399159. DOI: 10.1145/3567955.3567964. URL:
[46] Chen, M. et al. "TraDE: Network and Traffic-aware Adaptive Scheduling for Microservices Under Dynamics". In: *arXiv preprint arXiv:2411.05323* (2024).
[47] Chen, S. et al. "Workload characterization of interactive cloud services on big and small server platforms". In: *2017 IEEE International Symposium on Workload Characterization (IISWC)*. 2017, pp. 125–134. DOI: 10.1109/IISWC.2017.8167770.
[48] Gan, Y. et al. "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems". In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '19. Providence, RI, USA: Association for Computing Machinery, 2019, pp. 3–18. ISBN: 9781450362405. DOI: 10.1145/3297858.3304013. URL:
[49] Dean, J. and Barroso, L. A. "The tail at scale". In: *Communications of the ACM* 56.2 (2013), pp. 74–80.
[50] Chandra, A., Gong, W., and Shenoy, P. "Dynamic Resource Allocation for Shared Data Centers Using Online Measurements". In: *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. San Diego, CA, USA: ACM, 2003, pp. 300–301. DOI: 10.1145/781027.781067.
[51] Shen, Z. et al. "CloudScale: elastic resource scaling for multi-tenant cloud systems". In: *Proceedings of the 2nd ACM Symposium on Cloud Computing (SoCC)*. ACM, 2011, 5:1–5:14. DOI: 10.1145/2038916.2038921.
[52] Xu, C. et al. "Performance Interference of Virtual Machines: A Survey". In: *ACM SIGMETRICS Performance Evaluation Review* 41.3 (2013), pp. 6–18. DOI: 10.1145/2556647.2556650.
[53] Tzenetopoulos, A. et al. "Interference-aware workload placement for improving latency distribution of converged HPC/Big Data cloud infrastructures". In: *Proceedings of the 2021 IEEE/ACM Workshop on Interference-Aware Computing (IAC '21)*. St. Louis, MO, USA: IEEE, 2021, pp. 1–9. DOI: 10.1109/IAC52557.2021.00007.
[54] Kubernetes Authors. *Kubernetes Scheduler Extender*. Accessed: 2025-09-21. 2025.
[55] Kubernetes Authors. *Kubernetes API Overview*. Accessed: 2025-09-21. 2025.
[56] Kubernetes Authors. *Kubernetes Operators*. Accessed: 2025-09-21. 2025.
[57] Kubernetes Authors. *Kubernetes Custom Resources*. Accessed: 2025-09-21. 2025.
[58] Box, G. E. P. and Jenkins, G. M. *Time Series Analysis: Forecasting and Control*. San Francisco: Holden-Day, 1970.
[59] Kubernetes Authors. *Kubernetes Python Client*. Accessed: 2025-09-19. 2025.
[60] Kubernetes SIGs. *minikube Documentation*. Accessed: 2025-09-29.
[61] Docker Inc. *Docker: Accelerated Container Application Development*. Accessed: 2025-09-29.
[62] Winterstein, F. *Ultra-Low Latency XGBoost with Xelera Silva*. 2025.
[63] Gao, P. et al. "Low latency rnn inference with cellular batching". In: *Proceedings of the Thirteenth EuroSys Conference*. 2018, pp. 1–15.

[64] Kouris, A. et al. *Approximate LSTMs for Time-Constrained Inference: Enabling Fast Reaction in Self-Driving Cars*. 2019. arXiv: 1905.00689 [eess.SP]. URL: