ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

# Σχεδιασμός και Υλοποίηση Μηχανισμών για Συνεργατική και Αυτόνομη Διαχείριση IoT-Edge-Cloud Υποδομών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ανδρέας Ι. Ψύχας

**Επιβλέπων :** Εμμανουήλ Βαρβαρίγος
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2025

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

# Σχεδιασμός και Υλοποίηση Μηχανισμών για Συνεργατική και Αυτόνομη Διαχείριση IoT-Edge-Cloud Υποδομών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ανδρέας Ι. Ψύχας

**Επιβλέπων :** Εμμανουήλ Βαρβαρίγος
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 21η Οκτωβρίου 2025.

| ..... | ..... | ..... |
|---|---|---|
| Εμμανουήλ Βαρβαρίγος | Κωνσταντίνος Τσερπές | Θεοδώρα Βαρβαρίγου |
| Καθηγητής Ε.Μ.Π. | Επ. Καθηγητής Ε.Μ.Π. | Καθηγήτρια Ε.Μ.Π. |

Αθήνα, Οκτώβριος 2025

.....

Ανδρέας Ι. Ψύχας

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

# Περίληψη

Η ραγδαία εξέλιξη κατανεμημένων υπολογιστικών υποδομών έχει οδηγήσει στην ανάπτυξη cloud-native εφαρμογών, οι οποίες βασίζονται στην διάσπαση μονολιθικών εφαρμογών σε μικρότερες αλληλοσυνδεόμενες εργασίες, που αναφέρονται ως μικροϋπηρεσίες. Η προσέγγιση αυτή ενισχύει την απόδοση, την ελαστικότητα και την ανεκτικότητα των εφαρμογών, ενώ βελτιώνει τη χρησιμοποίηση των υπολογιστικών πόρων. Η βελτιστοποίηση της απόδοσης των cloud-native εφαρμογών, απαιτεί επιπλέον παραμέτρους για την συνολική καθυστέρηση των εφαρμογών και την καθυστέρηση μεταξύ των μικροϋπηρεσιών, οι οποίες δεν μπορούν να εξυπηρετηθούν αποκλειστικά από το cloud. Για τον λόγο αυτό, καθίσταται αναγκαία η αξιοποίηση υπολογιστικών πόρων κοντά στους χρήστες σε near και far edge επίπεδα, ώστε να διασφαλίζεται η καλύτερη ποιότητα εξυπηρέτησης. Επίσης, η ευρεία διάδοση τεχνολογιών, όπως το 5G, εντείνει την ανάγκη για αποτελεσματική κατανομή εφαρμογών που απαιτούν συνεχή απόκριση σε πραγματικό χρόνο, καθώς αυξάνεται ο αριθμός συσκευών, όπως αυτόνομα αυτοκίνητα, κινητά τηλέφωνα και IoT συσκευών, που μετακινούνται και προωθούν διαρκώς υπολογιστικές διεργασίες προς το edge-cloud.

Στην παρούσα εργασία προτείνονται μηχανισμοί αυτοματοποίησης της κατανομής cloud-native εφαρμογών στις ετερογενείς υπολογιστικές τοποθεσίες ενός edge-cloud περιβάλλοντος, με στόχο τη διατήρηση της ποιότητας εξυπηρέτησης καθ’ όλη τη διάρκεια εκτέλεσης τους. Το πρόβλημα αρχικά μοντελοποιείται ως Μεικτό Ακέραιο Πρόβλημα Γραμμικού Προγραμματισμού (MILP), το οποίο παρέχει τη βέλτιστη λύση για μικρής κλίμακας περιπτώσεις. Η εκτέλεση του MILP ενδέχεται να μην μπορεί να ολοκληρωθεί για προβλήματα με πραγματικό πλήθος εφαρμογών και edge-cloud τοποθεσιών σε ρεαλιστικό χρόνο. Για την αντιμετώπιση αυτού του περιορισμού, αναπτύσσεται ένας άπληστος αλγόριθμος που κατανέμει πόρους με την τεχνική καλύτερου ταιριάσματος (best-fit) προσεγγίζοντας τα αποτελέσματα του MILP σε σημαντικά λιγότερο χρόνο. Τέλος, αξιολογούνται τα αποτελέσματα των αλγορίθμων μέσω προσομοιώσεων σε διαφορετικά σενάρια και κριτήρια βελτιστοποίησης, αποδεικνύοντας την αποτελεσματικότητα και την επεκτασιμότητα τους σε μεγάλης κλίμακας προβλήματα.

**Λέξεις κλειδιά:** Cloud-native, Μικροϋπηρεσίες, Edge computing, Cloud Computing, Κατανομή πόρων, Κινητικότητα, Aggregators

# Abstract

The rapid evolution of distributed computing infrastructures has led to the development of cloud-native applications, which are based on the breakdown of a monolithic application into smaller, loosely connected components, often referred to as microservices. This approach enhances the performance, flexibility and resilience of applications while improving the utilization of computing resources. The optimization of cloud-native applications' performance requires additional parameters related to the overall application latency and communication delays between microservices, which cannot be fully addressed solely by the cloud. For this reason, the utilization of computing resources located closer to end users at the near and far edge layers becomes necessary to ensure optimal quality of service. Moreover, the widespread adoption of technologies such as 5G amplifies the need for efficient deployment of applications that demand continuous real-time responsiveness, as the number of devices, such as autonomous vehicles, mobile phones, and IoT devices, that move and continuously offload computational processes to the edge-cloud increases.

This study proposes mechanisms for automating the allocation of cloud-native applications across heterogeneous computing sites within an edge-cloud environment, aiming to maintain quality of service throughout their execution. The problem is initially modeled as a Mixed Integer Linear Programming (MILP) problem, which provides the optimal solution for small-scale cases. However, the execution of MILP may not be feasible for problems involving realistic numbers of applications and edge-cloud locations within realistic time limits. To address this limitation, a greedy algorithm is developed that allocates resources in a best-fit manner, approximating the MILP results in significantly less time. Finally, the algorithms' performance is evaluated through simulations under various scenarios and optimization criteria, demonstrating their effectiveness and scalability for large-scale problems.



**Keywords:** Cloud-native, Microservices, Edge computing, Cloud Computing, Resource allocation, Mobility, Aggregators

# Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον καθηγητή του ΕΜΠ κ. Εμμανουήλ Βαρβαρίγο για την εμπιστοσύνη και την δυνατότητα που μου έδωσε να εμβαθύνω σε ένα σύγχρονο θεματικό πεδίο με αυξανόμενο ερευνητικό ενδιαφέρον. Επίσης, ευχαριστώ τον Διδάκτορα Πολυζώη Σούμπλη για τη συνεχή καθοδήγησή του και τη συνεργασία σε όλα τα στάδια εκπόνησης της εργασίας. Τέλος, θα ήθελα να ευχαριστήσω τη οικογένεια μου και τους φίλους μου για τη στήριξη τους όλα αυτά τα χρόνια και την έμπνευση που μου έδωσαν καθ' όλη τη διάρκεια της ακαδημαϊκής μου πορείας.

Ανδρέας Ψύχας, Οκτώβριος 2025

# Πίνακας περιεχομένων

# Περιεχόμενα Εικόνων

# Περιεχόμενα Πινάκων

# 1 Εισαγωγή

Με την ραγδαία αύξηση του όγκου των παραγόμενων δεδομένων από IoT συσκευές και την εξέλιξη των cloud-native εφαρμογών, η αποκλειστική χρήση του cloud computing δεν είναι ικανή να εξυπηρετήσει τις ανάγκες των χρηστών. Πολλές φορές ο όγκος των δεδομένων καθιστά αδύνατη την μετάδοση σε ένα κεντρικό cloud για επεξεργασία και αποθήκευση λόγω του απαραίτητου χρόνου και κόστους μετάδοσης των δεδομένων και της χωρητικότητας των δικτύων πρόσβασης και μητροπολιτικών δικτύων που δεν επαρκεί και η καθυστέρηση που προστίθεται είναι απαγορευτική. Επίσης, πολλές cloud-native εφαρμογές απαιτούν real-time απόκριση (time-critical) και αδιάκοπη παροχή των υπηρεσιών (mission-critical), όπως συστήματα ασφαλείας, καθιστώντας δύσκολο να εγγυηθεί η εξυπηρέτηση τους από το cloud. Για τον λόγο αυτό προτάθηκε το edge computing που φέρνει τους πόρους σε διάφορες κατανεμημένες τοποθεσίες πιο κοντά στους τελικούς χρήστες επιτρέποντας καλύτερη εξυπηρέτηση τέτοιων εφαρμογών. Το edge αποτελείται από μικρότερης ισχύος υπολογιστικούς πόρους με αυξημένο κόστος λειτουργίας σε μικρά κέντρα δεδομένων ή τοπικούς εξυπηρετητές εντός της μητροπολιτικής περιοχής σε αντίθεση με το cloud που έχει πανίσχυρα κέντρα δεδομένων με εικονικά απεριόριστους υπολογιστικούς πόρους και μικρότερο κόστος λειτουργίας σε απομακρυσμένες τοποθεσίες.

Ωστόσο, η μεταφορά των δυνατοτήτων του cloud στο edge έρχεται με προκλήσεις στον συντονισμό (orchestration), στην ασφάλεια και στην συντήρηση των cloud-native εφαρμογών. Εξαιτίας των μικρών δυνατοτήτων και του μεγάλου πλήθους των edge τοποθεσιών δημιουργείται η ανάγκη για το συντονισμό και την κατάλληλη κατανομή των πόρων τους στις εφαρμογές.

Επίσης οι υπολογιστικοί πόροι στο edge είναι κατανεμημένοι σε διάφορα σημεία και συνήθως ανήκουν σε διαφορετικούς operators. Το edge είναι μια ετερογενής υποδομή με πόρους που διαφέρουν ως προς την υπολογιστική ισχύ, τον τύπο και ανήκουν σε διαφορετικούς διαχειριστές.

Πάνω από αυτήν την υποδομή έχει προταθεί η χρήση των aggregators οι οποίοι είναι υπεύθυνοι για την ενορχήστρωση και την κατανομή των εφαρμογών στους διάφορους κόμβους και τον διαρκή έλεγχο της κατάστασης της υποδομής.

Καθώς ένας aggregator δεν έχει τους διαθέσιμους υπολογιστικούς πόρους για να εξυπηρετήσει εφαρμογές οι οποίες απαιτούν τη συνεργατική χρήση υποδομών διαφορετικών χαρακτηριστικών και την εκτέλεση τους σε διαφορετικά σημεία, θα μπορούσαν οι aggregators να δουλεύουν συνεργατικά για την εξυπηρέτηση των εφαρμογών ανάλογα με τη διαθεσιμότητα των πόρων σε κάθε aggregator και τις καλύτερες τοποθεσίες για την εξυπηρέτηση μιας εφαρμογής. Παρόλο που αυτό δεν είναι πάντα εφικτό λόγω πχ διαφορετικών αναγκών ασφαλείας και συμφωνιών ανάμεσα στους διάφορους παρόχους των aggregators.

Από την εκτέλεση των εφαρμογών πάνω από μια τέτοια υποδομή, προκύπτουν προβλήματα λόγω της κινητικότητας των χρηστών, οι time-critical εφαρμογές απαιτούν σταθερά γρήγορο χρόνο απόκρισης και καθώς ο χρήστης μετακινείται απομακρύνεται από ορισμένους κόμβους της υποδομής και πλησιάζει άλλους με

αποτέλεσμα να απαιτείται επαναδρομολόγηση της εφαρμογής για να διατηρούνται οι περιορισμοί της. Στην περίπτωση που παραβιαστεί η ποιότητα εξυπηρέτησης (QoS), πχ ο επιτρεπτός χρόνος απόκρισης μιας εφαρμογής, θα πρέπει να γίνει νέα δέσμευση πόρων πάνω από μια τέτοια υποδομή. Για να εκτελεστεί σωστά η εφαρμογή θα πρέπει να γίνει μεταφορά ενός μέρους της εφαρμογής σε νέους κόμβους. Ανάλογα σε ποιον κόμβο γίνεται η μεταφορά της εφαρμογής ορίζουμε τις ακόλουθες δύο περιπτώσεις: (i) εντός της ίδιας συνεργασίας aggregators (soft handover) (ii) να μεταφερθεί ολόκληρη σε νέους κόμβους ενός aggregator (hard handover).

Επίσης, η αλλαγή αυτή δεν θα πρέπει να γίνεται αισθητή στον χρήστη καθώς επίσης πολλές cloud-native εφαρμογές είναι mission-critical. Απαιτείται συνεπώς επικοινωνία μεταξύ των aggregator και μεταφορά της τρέχουσας κατάστασης μιας εφαρμογής στους καινούργιους κόμβους και aggregators με αποτέλεσμα τα soft και τα hard handovers να επιβαρύνουν την υποδομή για να διατηρηθεί η συνεχή εξυπηρέτηση της εφαρμογής.

Πιο συγκεκριμένα, εξετάζεται μία edge cloud υποδομή που αποτελείται από τρία επίπεδα near edge-far edge-cloud και ο κάθε κόμβος έχει διαφορετικές υπολογιστικές και αποθηκευτικές δυνατότητες. Οι κόμβοι της υποδομής είναι οργανωμένοι σε πλατφόρμες, όπου δρουν διαφορετικοί aggregators ο καθένας υπεύθυνος για ένα υποσύνολο από κόμβους και κάποιοι aggregators έχουν τη δυνατότητα να συνεργάζονται μεταξύ τους για την εξυπηρέτηση μίας cloud-native εφαρμογής. Οι cloud-native εφαρμογές προέρχονται από ένα σύνολο από κινούμενους χρήστες που έχουν να εκτελέσουν ετερογενή φορτία εργασίας με διαφορετικές υπολογιστικές απαιτήσεις που δεν μπορούν να εξυπηρετήσουν οι ίδιες οι συσκευές των χρηστών. Η κάθε εφαρμογή έχει ένα αποδεκτό χρόνο απόκρισης που απαιτείται για την συνολική επεξεργασία της εφαρμογής ανά χρονική στιγμή εκτέλεσής της για αυτό απαιτείται κατάλληλη κατανομή των εφαρμογών στους περιορισμένους πόρους των edge κόμβων. Όταν ξεκινάει μία εφαρμογή, δεσμεύει τους πόρους που χρειάζεται από τους κόμβους της υποδομής πληρώντας τις υπολογιστικές και αποθηκευτικές απαιτήσεις του και την αποδεκτή καθυστέρηση με βάση την θέση του εκείνη τη στιγμή. Καθώς ο χρήστης μετακινείται, ενδέχεται να παραβιαστεί ο περιορισμός της συνολικής καθυστέρησης του, τότε μπορεί είτε να γίνει αλλαγή μερικών μικροϋπηρεσιών σε νέους κόμβους εντός της ίδιας συνεργασίας aggregators (soft handover) είτε να μεταφερθεί ολόκληρη σε κόμβους διαφορετικού aggregator (hard handover). Τα soft και τα hard handovers κοστίζουν στην υποδομή για την μεταφορά της τρέχουσας κατάστασης της εφαρμογής με αποτέλεσμα να χρειάζεται να ελαχιστοποιηθεί η μεταφορά των μικροϋπηρεσιών μεταξύ κόμβων της υποδομής.

Ένα χαρακτηριστικό παράδειγμα time και mission critical cloud-native εφαρμογών είναι τα αυτόνομα αυτοκίνητα. Ένα σύγχρονο αυτοκίνητο περιλαμβάνει περισσότερους από 60 αισθητήρες και κάμερες που ελέγχουν διάφορες πλευρές του οχήματος, δημιουργώντας ένα τεράστιο όγκο δεδομένων προς επεξεργασία κατά την διάρκεια κίνησης τους. Τα αυτόνομα αυτοκίνητα έχουν πολλές εργασίες με διαφορετικές απαιτήσεις, από το να παρθούν αποφάσεις σε κλάσματα του δευτερολέπτου για την ασφάλεια των επιβατών μέχρι να παρέχει

προσωποποιημένες πληροφορίες και υπηρεσίες για τους επιβάτες αναλύοντας τα παραγόμενα δεδομένα. Η τοποθεσία τους αλλάζει συνεχώς, που αυξάνει τις δυσκολίες για τον μεγάλο όγκο δεδομένων που δημιουργείται, ειδικά για τα αυτόνομα αυτοκίνητα που η λειτουργία τους εξαρτάται από την άμεση επεξεργασία των δεδομένων [1].

Επίσης, με την άνοδο των αυτόνομων αυτοκινήτων προστίθεται και η ανάγκη για επικοινωνία των αυτοκινήτων μεταξύ τους (vehicle-to-vehicle V2V), με την υποδομή (vehicle-to-infrastructure V2I) και με τους πεζούς (vehicle-to-pedestrian V2P). Οι vehicle-to-everything V2X επικοινωνίες ενδέχεται να υιοθετηθούν από πολλές χώρες για την καλύτερη διαχείριση του οδικού δικτύου προσφέροντας πληροφορίες για τις συνθήκες κίνησης και τυχόν εμποδίων στους δρόμους (μέσω V2V επικοινωνίας) και για τις συνθήκες του περιβάλλοντος κυκλοφορίας όπως φανάρια (μέσω V2I επικοινωνίας) και πεζοί (μέσω V2P επικοινωνίας) [2]. Οι άμεσοι χρόνοι απόκρισης είναι απαραίτητοι για την ασφάλεια των επιβατών του αυτοκινήτου και η επικοινωνία μεταξύ πολλαπλών αμαξιών και της υποδομής κινδυνεύει από προβλήματα συνδεσιμότητας. Οι V2X συχνά ασχολούνται με την ανταλλαγή ευαίσθητων πληροφοριών, οπότε είναι σημαντική και η διασφάλιση της ασφάλειας και της ιδιωτικότητας των δεδομένων.

Το edge computing προσφέρει την ανθεκτικότητα, την ασφάλεια και τον ελάχιστο χρόνο απόκρισης για να εξασφαλίσει αυτές τις απαιτήσεις. Το cloud επίπεδο του edge computing προσφέρει καλύτερη ανάλυση και αποθήκευση των δεδομένων για μακροχρόνια χρήση, καθώς τα δεδομένα μπορεί να απαιτούνται για μία παρατεταμένη περίοδο όπου δεν είναι απαραίτητη η άμεση απάντηση. Το near και το far edge επίπεδο συνδέουν το αυτοκίνητο με το cloud προσφέροντας αξιόπιστη επικοινωνία και διαχειρίζονται εργασίες που απαιτούν ταχύτερο χρόνο απόκρισης, όπως η ανίχνευση και η αποφυγή εμποδίων. Το vehicle network επίπεδο είναι υπεύθυνο για την συλλογή των δεδομένων, την προώθηση τους στο edge και την επικοινωνία με άλλα αυτοκίνητα και πεζούς [2].

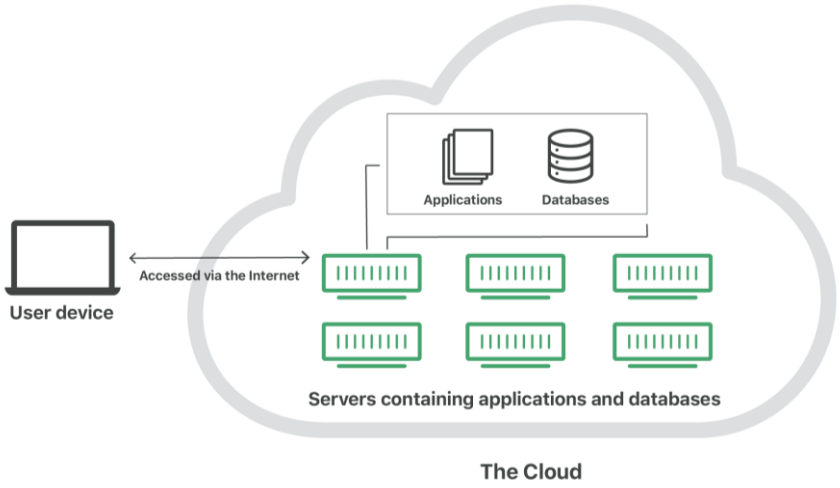
Τέλος, η υιοθέτηση V2X επικοινωνίας αυξάνει την ανάγκη για aggregators. Ο μεγάλος αριθμός διαφορετικών εταιριών, που διαχειρίζονται τα αυτόνομα αυτοκίνητα, καθώς και η μετακίνηση μεταξύ διαφορετικών νομοθετικών πλαισίων ανάμεσα σε περιοχές απαιτούν συντονισμό από ετερόκλητους ενορχηστρωτές. Εξαιτίας της διαρκής μετακίνησης τους, χρειάζεται συχνή επικοινωνία μεταξύ των aggregators και πολλές φορές προκαλούνται συνεργασίες για την κατανομή των πόρων και εκ νέου δρομολογήσεις σε διαφορετικές τοποθεσίες της edge-cloud υποδομής.

# 2 Θεωρητικό Υπόβαθρο

## 2.1 Cloud Computing

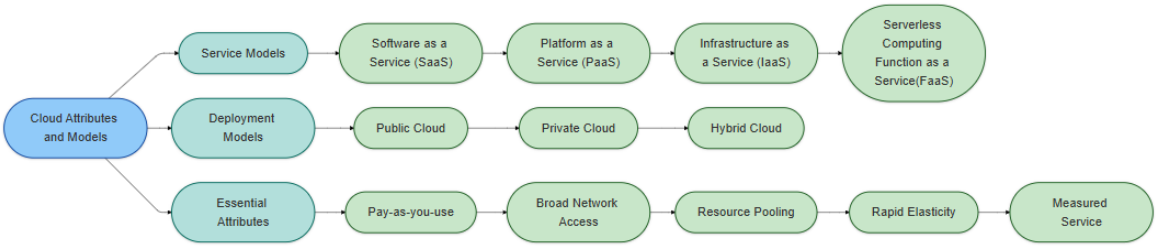
### 2.1.1 Ορισμός του Cloud Computing

Ο όρος cloud computing (ή υπολογιστικό νέφος) χρησιμοποιείται για την αποθήκευση και πρόσβαση δεδομένων και προγραμμάτων μέσω του διαδικτύου αντί για την χρήση τοπικών υποδομών [3]. Η IBM περιγράφει τον όρο ως την κατ' απαίτηση (on-demand) πρόσβαση σε υπολογιστικούς πόρους (είτε φυσικούς είτε εικονικούς), αποθηκευτικούς χώρους δεδομένων, δυνατοτήτων δικτύου, εργαλείων ανάπτυξης εφαρμογών και λογισμικού μέσω του διαδικτύου, τα οποία φιλοξενούνται σε απομακρυσμένα κέντρα δεδομένων και συνήθως προσφέρονται στους χρήστες μέσω συνδρομής [4]. Πιο αναλυτικά, ο όρος cloud computing περιγράφει μια υπολογιστική τεχνική όπου υπηρεσίες πληροφορικής παρέχονται από τεράστιες χαμηλού κόστους υπολογιστικές μονάδες που συνδέονται μέσω ενός IP δικτύου [5].




*Εικόνα 2-1 Cloud Computing*

### 2.1.2 Μοντέλα Cloud




*Εικόνα 2-2 Μοντέλα Υπηρεσιών και Ανάπτυξης του Cloud*

#### 2.1.2.1 Μοντέλα Υπηρεσιών

Οι υπηρεσίες cloud εμπίπτουν σε μία εκ των τεσσάρων κατηγοριών [4],[6]: Λογισμικό ως Υπηρεσία (Software as a Service (SaaS)), Πλατφόρμα ως Υπηρεσία (Platform as a Service (PaaS)), Υποδομή ως Υπηρεσία (Infrastructure as a Service (IaaS)) και το Serverless Computing.

##### 2.1.2.1.1 Λογισμικό ως Υπηρεσία

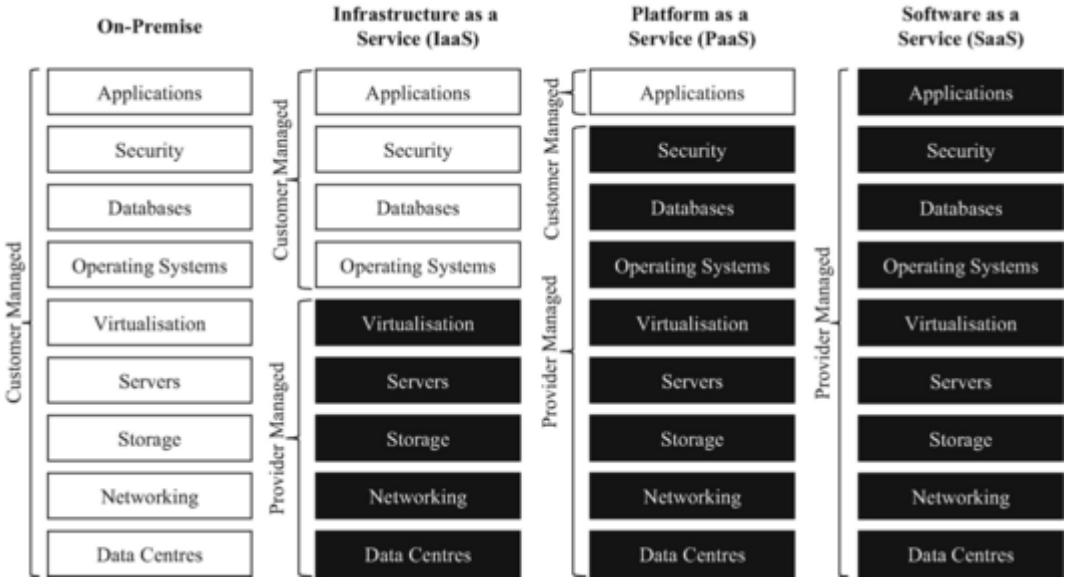
Το Λογισμικό ως Υπηρεσία (Software as a Service (SaaS)), συχνά αναφερόμενο ως cloud-based λογισμικό ή cloud εφαρμογές, είναι ένα μοντέλο διανομής λογισμικού που επιτρέπει στους χρήστες να έχουν πρόσβαση σε εφαρμογές που φιλοξενούνται σε υποδομές ενός παρόχου λογισμικού μέσω του διαδικτύου. Συγκεκριμένα, το SaaS είναι προσβάσιμο από τον χρήστη μέσω ενός προγράμματος περιηγητή ιστού ή μίας διεπαφής προγραμματισμού εφαρμογών (Application Programming Interface API) που ενσωματώνεται στο λειτουργικό σύστημα του υπολογιστή ή της κινητής συσκευής του χρήστη. Η εφαρμογή και τα δεδομένα της εξυπηρετούνται στις υποδομές του παρόχου και έτσι ο χρήστης απαλλάσσεται από την ανάγκη συντήρησης της εφαρμογής και προστατεύεται από τον κίνδυνο απώλειας δεδομένων λόγω τυχόν προβλημάτων της συσκευής. Οι SaaS εφαρμογές αποτελούν το μεγαλύτερο μέρος cloud computing και είναι το βασικό μοντέλο παροχής λογισμικού για πολλές εμπορικές εφαρμογές σήμερα, όπως το Microsoft Office 365, οι υπηρεσίες ηλεκτρονικού ταχυδρομείου και οι πλατφόρμες streaming [4].

##### 2.1.2.1.2 Πλατφόρμα ως Υπηρεσία

Η Πλατφόρμα ως Υπηρεσία (Platform as a Service (PaaS)) παρέχει στους προγραμματιστές ένα πλήρως εξοπλισμένο περιβάλλον για την ανάπτυξη και διαχείριση εφαρμογών χωρίς την πολυπλοκότητα και την ανελαστικότητα διατήρησης ενός τοπικού περιβάλλοντος. Ο πάροχος του cloud είναι υπεύθυνος για την διατήρηση της υπηρεσίας σε δικά του κέντρα δεδομένων, η υπηρεσία αυτή περιλαμβάνει εξυπηρετητές (servers), δίκτυα επικοινωνιών, αποθηκευτικό χώρο, λογισμικό λειτουργικού συστήματος και βάσεις δεδομένων. Η Πλατφόρμα ως Υπηρεσία συνήθως βασίζεται σε containers, ένα μοντέλο εικονικοποιημένης υπολογιστικής που απέχει ένα βήμα από τους εικονικούς διακομιστές [4]. Τα containers εικονικοποιούν το λειτουργικό σύστημα, επιτρέποντας στους προγραμματιστές να ενσωματώσουν την εφαρμογή μαζί με τις απαραίτητες υπηρεσίες του λειτουργικού συστήματος που χρειάζεται για να εκτελεστεί σε οποιαδήποτε πλατφόρμα χωρίς τροποποιήσεις.

### 2.1.2.1.3Υποδομή ως Υπηρεσία

Η Υποδομή ως Υπηρεσία (Infrastructure as a Service (IaaS)) παρέχει κατ' απαίτηση πρόσβαση σε θεμελιώδεις υπολογιστικούς πόρους, όπως φυσικούς και εικονικούς εξυπηρετητές, δίκτυο επικοινωνίας και αποθηκευτικό χώρο, μέσω του διαδικτύου. Με την Υποδομή ως Υπηρεσία, οι χρήστες και οι εταιρίες έχουν την δυνατότητα να χρησιμοποιούν πόρους με βάση τις ανάγκες τους, διατηρώντας την ευελιξία να τους αυξήσουν ή μειώσουν σύμφωνα με την ζήτηση με αυτόματο και ευέλικτο τρόπο και χρέωση. Παραδείγματα IaaS περιλαμβάνουν τα Amazon Web Services EC2, MS Azure VMs και Google Cloud Platform Compute Engine [6].




*Εικόνα 2-3 Επίπεδα ελέγχου χρήστη σε μοντέλα Cloud computing υπηρεσιών*

### 2.1.2.1.4Serverless computing

Το serverless computing, ή απλά serverless, είναι ένα μοντέλο cloud computing που χρησιμοποιεί υπολογιστικούς πόρους κατ' απαίτηση του κώδικα που καλείται να εκτελέσει και χρεώνεται με βάση τους πόρους που χρησιμοποιήθηκαν. Αυτό επιτρέπει στους προγραμματιστές να επικεντρώνονται αποκλειστικά στον κώδικα και στη λογική των επιχειρησιακών τους εφαρμογών. Το serverless εκτελεί τον κώδικα εφαρμογών μόνο κατόπιν αιτήματος και προσαρμόζει αυτόματα την υποστηρικτική υποδομή ανάλογα με τον αριθμό των αιτημάτων.

Το Function as a Service (FaaS) είναι ένα υποσύνολο του serverless computing. Στο FaaS, οι εφαρμογές cloud αποτελούνται από μικρότερα τμήματα που εκτελούνται μόνο όταν χρειάζονται. Το FaaS επιτρέπει στους προγραμματιστές να εκτελούν τμήματα κώδικα εφαρμογών ως απάντηση σε συγκεκριμένα γεγονότα. Πλέον του κώδικα, η φυσική υλικοτεχνική υποδομή, το λειτουργικό σύστημα της εικονικής μηχανής (VM) και η διαχείριση του λογισμικού του εξυπηρετητή διατίθενται από τον πάροχο υπηρεσιών cloud σε πραγματικό χρόνο όσο εκτελείται ο κώδικας και απενεργοποιούνται μόλις ολοκληρωθεί η εκτέλεση.

#### 2.1.2.2 Μοντέλα Ανάπτυξης

Σύμφωνα με τις δυνατότητες, τις απαιτήσεις διαχείρισης και την ευαισθησία των δεδομένων διακρίνονται τρία μοντέλα ανάπτυξης και χρήσης cloud υποδομών [4],[6]: το δημόσιο cloud, το ιδιωτικό cloud και το υβριδικό cloud.

##### 2.1.2.2.1 Δημόσιο Cloud

Το δημόσιο cloud αναφέρεται συχνά ως εξωτερικό cloud. Αυτός ο τύπος cloud είναι ανοιχτός σε όλους τους χρήστες ή σε μεγάλες ομάδες χρηστών μέσω του διαδικτύου, με την ιδιοκτησία να παραμένει στους παρόχους υπηρεσιών cloud. Λειτουργεί από τον πάροχο και επιτρέπει στους χρήστες να έχουν πρόσβαση στις υπηρεσίες του μέσω του διαδικτύου. Το δημόσιο cloud προσφέρει οικονομικούς και ευέλικτους τρόπους ανάπτυξης λύσεων τεχνολογίας πληροφορικής (IT). Στο συγκεκριμένο μοντέλο συνυπάρχουν πολλοί και ετερόκλητοι πελάτες που μοιράζονται τους πόρους του παρόχου και η πρόσβαση σε αυτούς γίνεται μέσω του διαδικτύου, με αποτέλεσμα να απαιτείται ιδιαίτερη προσοχή για την ασφάλεια και ακεραιότητα των δεδομένων.

##### 2.1.2.2.2 Ιδιωτικό Cloud

Το ιδιωτικό cloud αναφέρεται συχνά ως εσωτερικό ή εταιρικό cloud. Αυτός ο τύπος προορίζεται για χρήση από μία μόνο εταιρία ή σύνολο εταιριών. Μπορεί να λειτουργεί από την ίδια την εταιρία ή από ένα ανεξάρτητο μέρος και συνήθως βρίσκεται σε τοπικές, ιδιωτικές εγκαταστάσεις. Συχνά το ιδιωτικό cloud είναι πιο ασφαλές από το δημόσιο, ωστόσο είναι πιο δαπανηρό για την εγκατάσταση και την λειτουργία του. Πολλές εταιρίες προτιμούν το ιδιωτικό cloud για λόγους συμμόρφωσης με κανονιστικά πλαίσια, όπου τα δεδομένα δεν επιτρέπεται να βρίσκονται εκτός της εταιρίας ή της χώρας.

##### 2.1.2.2.3 Υβριδικό Cloud

Αυτός ο τύπος χρήσης cloud προκύπτει λόγω της ποικιλομορφίας των απαιτήσεων ενός οργανισμού, που μπορεί να απαιτεί τον συνδυασμό υπηρεσιών από δύο ή περισσότερων παρόχων δημόσιων ή ιδιωτικών cloud υποδομών. Επιτρέπει στους οργανισμούς να φιλοξενούν ευαίσθητα δεδομένα ή εφαρμογές στο ιδιωτικό cloud και μη ευαίσθητα στο δημόσιο cloud. Επίσης, σε ένα υβριδικό cloud, το δημόσιο cloud επιτρέπει σε εταιρίες να αυξήσουν γρήγορα τους πόρους σε περίπτωση απρόσμενης αύξησης των απαιτήσεων χωρίς να επηρεάζεται ο φόρτος εργασίας του ιδιωτικού cloud, το χαρακτηριστικό αυτό αναφέρεται συχνά ως “cloud bursting”.

#### 2.1.2.3 Βασικά Χαρακτηριστικά

Ένα βασικό χαρακτηριστικό του cloud computing είναι η προσφορά κατ' απαίτηση με αυτοεξυπηρέτηση υπολογιστικών πόρων για την ανάπτυξη και την εξυπηρέτηση εφαρμογών, για την χρήση του cloud computing δεν απαιτείται κάποιος άνθρωπος υπεύθυνος για την λειτουργία των εφαρμογών, αλλά οι χρήστες έχουν την δυνατότητα να προμηθεύονται, να ελέγχουν και να διαχειρίζονται τους πόρους ανάλογα με τις ανάγκες τους [7]. Το cloud computing παρέχει ευρεία πρόσβαση στους πόρους, οι οποίοι είναι διαθέσιμοι από οπουδήποτε μέσω του δικτύου. Το cloud προσφέρει ελαστικότητα στις εφαρμογές, οι πόροι μπορούν να αποδεσμεύονται και να διατίθενται γρήγορα ανάλογα με τις ανάγκες, παρέχονται περισσότεροι πόροι όταν ένας χρήστης τους απαιτεί και μειώνονται όταν σταματήσει η απαίτηση τους. Ακόμη, η συνένωση πόρων (resource pooling) είναι ένα βασικό χαρακτηριστικό του cloud computing που επιτρέπει την εξυπηρέτηση πολλών χρηστών από μια κοινή δεξαμενή υπολογιστικών πόρων που προέρχονται από την ίδια φυσική τοποθεσία. Επίσης, στο cloud computing η χρέωση γίνεται με βάση τη χρήση (Pay-as-you-use), με τους χρήστες να χρεώνονται ανάλογα τους πόρους που καταναλώνουν. Τέλος, το cloud computing προσφέρει ανθεκτικότητα στις εφαρμογές, τα συστήματα cloud είναι φτιαγμένα με πλεονασμό και ανοχή σφαλμάτων εξασφαλίζοντας υψηλή διαθεσιμότητα και αξιοπιστία.

### 2.1.3 Πλεονεκτήματα χρήσης του Cloud

Το cloud computing αποτελεί μια καινοτόμο τεχνολογία που έχει αλλάξει τον τρόπο με τον οποίο χρησιμοποιούμε και διαχειριζόμαστε τους ψηφιακούς πόρους. Είναι προσβάσιμο μέσω του διαδικτύου και επιτρέπει στους χρήστες να αξιοποιούν υπολογιστική ισχύ και αποθηκευτικούς πόρους από απόσταση. Αυτή η αλλαγή προσφέρει νέες δυνατότητες στους χρήστες των υπηρεσιών. Μεταξύ των πλεονεκτημάτων της χρήσης του cloud συγκαταλέγεται η σταθερή προσβασιμότητα και αξιοπιστία, καθώς η πληροφορία που αποθηκεύεται στο cloud είναι διαθέσιμη από οπουδήποτε υπάρχει σύνδεση στο διαδίκτυο, επιτρέποντας τη συνεχή διαθεσιμότητα δεδομένων και εφαρμογών [8]. Επιπλέον, το cloud παρέχει πρακτικά απεριόριστους πόρους υπολογιστικής ισχύος και αποθήκευσης, ικανοποιώντας τις ανάγκες των χρηστών χωρίς τους φυσικούς περιορισμούς των τοπικών συστημάτων. Παράλληλα, προσφέρει επεκτασιμότητα και ευελιξία, αφού οι χρήστες μπορούν να προσαρμόζουν τους χρησιμοποιούμενους πόρους σε πραγματικό χρόνο, χωρίς να απαιτούνται προκαταβολικές επενδύσεις σε υποδομές. Η δυναμική κατανομή των πόρων οδηγεί σε αποδοτικότερη διαχείριση, ενισχύοντας την ευελιξία και την προσαρμοστικότητα. Το cloud επίσης προσφέρει συστήματα που παρακολουθούν και βελτιστοποιούν αυτόματα τη χρήση πόρων, αξιοποιώντας δεδομένα επεξεργασίας, αποθήκευσης και εύρους ζώνης, γεγονός που προσφέρει διαφάνεια τόσο στους χρήστες όσο και στους παρόχους . Ένα ακόμη σημαντικό πλεονέκτημα είναι το μειωμένο κόστος εγκατάστασης και συντήρησης,

καθώς η υποδομή cloud εξαλείφει την ανάγκη για φυσικό εξοπλισμό, ενώ οι χρήστες πληρώνουν μόνο για τους πόρους που πραγματικά χρησιμοποιούν, αποφεύγοντας άσκοπες δαπάνες. Παρότι η ασφάλεια δεδομένων αποτελεί πρόκληση, οι συνεχείς τεχνολογικές βελτιώσεις την καθιστούν ένα από τα ισχυρά πλεονεκτήματα των cloud υπηρεσιών. Τέλος, το cloud διευκολύνει τη συνεργασία και τη διαχείριση, επιτρέποντας σε πολλούς χρήστες να εργάζονται ταυτόχρονα σε κοινή υποδομή, προάγοντας την αποτελεσματικότητα στην ομαδική εργασία και στη διαχείριση έργων [9].

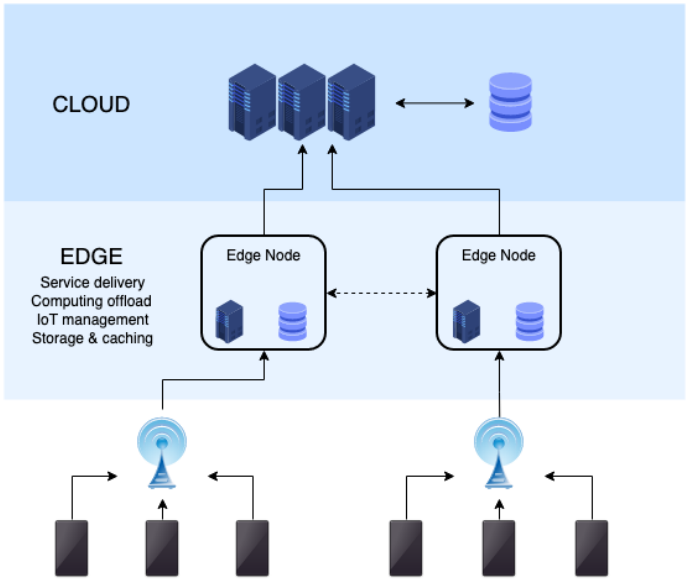
## 2.2 Edge Computing

### 2.2.1 Ορισμός του edge computing

Το edge computing είναι ένα σύγχρονο υπολογιστικό πρότυπο που επεξεργάζεται δεδομένα κοντά στην πηγή τους, μειώνοντας την εξάρτηση από κεντρικά συστήματα. Οι ορισμοί διαφέρουν, αλλά συγκλίνουν στην ιδέα ότι το edge computing ενοποιεί πόρους γεωγραφικά ή δικτυακά κοντά στον χρήστη, ώστε να παρέχει υπηρεσίες με χαμηλή καθυστέρηση και υψηλό εύρος ζώνης. Στόχος του είναι να καλύψει κρίσιμες ανάγκες διαχείρισης πληροφορίας, όπως ανάλυση δεδομένων σε πραγματικό χρόνο, βελτιστοποίηση δεδομένων, ασφάλεια και προστασία ιδιωτικότητας [10].

Σε αντίθεση με τα παραδοσιακά μοντέλα, οι υποδομές άκρου (edge), όπως έξυπνα τηλέφωνα, cloudlets και μικρά κέντρα δεδομένων, λειτουργούν ως ενδιάμεσοι κόμβοι μεταξύ των πηγών δεδομένων, όπως αισθητήρες, κινητές συσκευές, Internet of Things (IoT) συσκευές ή τοπικούς διακομιστές, και του cloud, υποστηρίζοντας λειτουργίες όπως επεξεργασία, αποθήκευση, προσωρινή αποθήκευση και κατανομή φορτίου. Στο edge computing, οι συσκευές δεν αποθηκεύουν απλώς δεδομένα, αλλά τα επεξεργάζονται και παράγουν πληροφορίες, βελτιώνοντας την απόδοση, μειώνοντας την καθυστέρηση και εξοικονομώντας ενέργεια [11].

Το edge computing αναγνωρίζεται ολοένα και περισσότερο ως μια κρίσιμη λύση για την αντιμετώπιση σύγχρονων προκλήσεων στη συνδεσιμότητα, τη διαχείριση δεδομένων και την παροχή έξυπνων υπηρεσιών, αυξάνοντας την αποτελεσματικότητα και την ευελιξία τους, καθιστώντας το ιδανικό για εφαρμογές που απαιτούν γρήγορη απόκριση και προσαρμοστικότητα. Ωστόσο, αντιμετωπίζει προκλήσεις, όπως η εξασφάλιση αξιοπιστίας, η περιορισμένη ενέργεια των συσκευών και η διαχείριση ζητημάτων ασφαλείας και ιδιωτικότητας λόγω της περίπλοκης τοπολογίας δικτύου και της χρήσης φθηνών προσωπικών συσκευών.




*Εικόνα 2-4 Edge Cloud Computing*

### 2.2.2 Internet of Things (IoT)

Το Internet of Things (IoT) περιγράφει φυσικά αντικείμενα (αισθητήρες κ.α.) που συνδέονται στο διαδίκτυο μέσω τεχνολογιών όπως το RFID (radio frequency identification technology), NFC (near-field communication) και οι ασύρματες επικοινωνίες [12]. Αυτά τα "έξυπνα αντικείμενα" έχουν τη δυνατότητα να συλλέγουν, να επεξεργάζονται και να ανταλλάσσουν δεδομένα, δημιουργώντας ένα δίκτυο πληροφοριών που συνδέει τον φυσικό με τον ψηφιακό κόσμο [13]. Το IoT βασίζεται στη χρήση μικρών ηλεκτρονικών εξαρτημάτων που ενσωματώνονται στα αντικείμενα, παρέχοντας τους μικρή τοπική υπολογιστική δυνατότητα και σύνδεση με το διαδίκτυο. Παραδείγματα περιλαμβάνουν έξυπνες συσκευές, όπως ψυγεία που παρακολουθούν την κατάσταση των τροφίμων ή εργαλεία που συλλέγουν δεδομένα χρήσης και τα στέλνουν για ανάλυση.

Η καινοτομία του IoT δεν έγκειται μόνο στη συνδεσιμότητα, αλλά και στον τεράστιο αριθμό συσκευών που αναμένεται να ενταχθούν στο δίκτυο, δημιουργώντας νέες ευκαιρίες, όπως η αυτοδιαχείριση δικτύων, η αυξημένη ασφάλεια και η βελτιστοποίηση λειτουργιών. Ωστόσο, η ευρεία υιοθέτηση του IoT δημιουργεί προκλήσεις, όπως η διασφάλιση της ιδιωτικότητας, η ασφάλεια των δεδομένων και η διαχείριση των δικτύων αυτών των έξυπνων αντικειμένων. Παρόλα αυτά, με την εξέλιξη τεχνολογιών όπως η ασύρματη επικοινωνία χαμηλής κατανάλωσης και οι έξυπνοι αισθητήρες, το IoT θεωρείται ως ένα από τα επόμενα βήματα της ψηφιακής επανάστασης.

### 2.2.3 Αρχιτεκτονική συστήματος Edge Cloud

Η αρχιτεκτονική ενός ιεραρχικού edge cloud συστήματος περιλαμβάνει πολλαπλά επίπεδα διακομιστών που συνεργάζονται για την επεξεργασία φόρτου εργασίας από τερματικές συσκευές. Η αρχιτεκτονική αυτή χωρίζεται σε τρία επίπεδα [10]. Το IoT-Τερματικό επίπεδο, που περιλαμβάνει όλες τις συσκευές που είναι συνδεδεμένες στο δίκτυο edge, όπως αισθητήρες, κινητά τηλέφωνα, έξυπνα αυτοκίνητα και κάμερες. Αυτές οι συσκευές λειτουργούν τόσο ως καταναλωτές όσο και ως πάροχοι δεδομένων. Συλλέγουν ακατέργαστα δεδομένα και τα ανεβάζουν στο επόμενο επίπεδο για επεξεργασία, χωρίς να διαθέτουν ισχυρή υπολογιστική ισχύ. Στη συνέχεια είναι το επίπεδο Edge, το οποίο είναι ο πυρήνας της τριεπίπεδης αρχιτεκτονικής, τοποθετημένος στο άκρο του δικτύου, και αποτελείται από κόμβους edge, όπως σταθμούς βάσης, routers, switches και gateways. Υποστηρίζει την πρόσβαση των τερματικών συσκευών και επεξεργάζεται τα δεδομένα που αυτές ανεβάζουν. Η εγγύτητα του επιπέδου edge προς τον χρήστη το καθιστά ιδανικό για ανάλυση δεδομένων σε πραγματικό χρόνο και έξυπνη επεξεργασία, με μεγαλύτερη αποδοτικότητα και ασφάλεια σε σχέση με το cloud. Τέλος το επίπεδο Cloud, όπου παραμένει το πιο ισχυρό κέντρο επεξεργασίας δεδομένων, αποτελούμενο από εξυπηρετητές υψηλών επιδόσεων και συσκευές αποθήκευσης. Αναλαμβάνει τη μόνιμη αποθήκευση δεδομένων και την επεξεργασία μεγάλων όγκων δεδομένων που απαιτούνται για ανάλυση. Επιπλέον, το cloud μπορεί να προσαρμόζει δυναμικά τη στρατηγική και τους αλγόριθμους που χρησιμοποιεί το επίπεδο edge, ενισχύοντας τη συνεργασία και την αποτελεσματικότητα.

### 2.2.4 Πλεονεκτήματα χρήσης του Edge Cloud

Το edge computing συνδυάζει τα πλεονεκτήματα της αποκεντρωμένης επεξεργασίας και της ευελιξίας, αποτελώντας μία σύγχρονη λύση για απαιτητικές εφαρμογές τεχνολογίας και την επεξεργασία μεγάλων όγκων δεδομένων. Η χρήση του προσφέρει σημαντικά οφέλη, όπως η μειωμένη καθυστέρηση, αφού η επεξεργασία των δεδομένων πραγματοποιείται κοντά στην πηγή τους, περιορίζοντας την απόσταση που αυτά πρέπει να διανύσουν για ανάλυση και επιτυγχάνοντας άμεση απόκριση σε πραγματικό χρόνο. Επιπλέον, μειώνεται το κόστος και αυξάνεται η ενεργειακή αποδοτικότητα, καθώς περιορίζεται ο όγκος των δεδομένων που αποστέλλονται στο cloud, με αποτέλεσμα την εξοικονόμηση εύρους ζώνης και τη μείωση των εξόδων αποθήκευσης και χρήσης δικτύου [14]. Η τοπική επεξεργασία καθιστά περιττή τη συνεχή σύνδεση με το cloud, ενισχύοντας την αυτονομία των συστημάτων.

Παράλληλα, το edge computing βελτιώνει την αποδοτικότητα του δικτύου, μειώνοντας τη συμφόρηση που προκαλείται από την υπερφόρτωση των κεντρικών υποδομών του cloud μέσω του διαδικτύου, και διασφαλίζει ταχύτερη και πιο αξιόπιστη επεξεργασία δεδομένων. Συνεισφέρει επίσης στην αξιοπιστία

των υπηρεσιών, καθώς λειτουργεί συμπληρωματικά με το cloud, επιτρέποντας τη συνεχή διαθεσιμότητα κρίσιμων λειτουργιών ακόμη και σε περίπτωση αποτυχίας του κεντρικού διακομιστή.

Η ασφάλεια των δεδομένων ενισχύεται σημαντικά, αφού η τοπική επεξεργασία και αποθήκευση περιορίζει την ανάγκη μεταφοράς ευαίσθητων πληροφοριών μέσω του διαδικτύου [15]. Επιπλέον, το edge computing απαιτεί μικρότερη εξάρτηση από τυποποιημένα πρότυπα δικτύου, καθώς περιορίζει την ανάγκη αποστολής δεδομένων στο cloud, άρα και την απαίτηση για κοινά πρωτόκολλα. Τέλος, ενισχύει το cloud computing, ιδίως σε εφαρμογές όπου η ταχύτητα είναι κρίσιμη, όπως στα αυτόνομα οχήματα, όπου το edge αναλαμβάνει τη λήψη άμεσων αποφάσεων, όπως η αποφυγή ατυχήματος, ενώ το cloud επεξεργάζεται πιο χρονοβόρες και σύνθετες διεργασίες, όπως η ανάλυση κυκλοφοριακών δεδομένων [16].

## 2.3 Cloud-native εφαρμογές

### 2.3.1 Ορισμός μίας Cloud-native Εφαρμογής

Η cloud-native αρχιτεκτονική αποτελεί μια σύγχρονη προσέγγιση για τον σχεδιασμό, την υλοποίηση και τη διαχείριση εφαρμογών που φιλοξενούνται στο cloud και αξιοποιούν στο έπακρο τα πλεονεκτήματα του cloud computing. Σύμφωνα με το Cloud Native Computing Foundation (CNCF), οι cloud-native τεχνολογίες επιτρέπουν την ανάπτυξη και διαχείριση επεκτάσιμων εφαρμογών σε δυναμικά περιβάλλοντα, είτε αυτά είναι δημόσια, ιδιωτικά ή υβριδικά cloud [17].

Βασικό χαρακτηριστικό των cloud-native εφαρμογών [18] είναι ότι αποτελούνται από μικροϋπηρεσίες (microservices), δηλαδή επιμέρους, ανεξάρτητες υπηρεσίες που εξυπηρετούν έναν συγκεκριμένο σκοπό και διαχειρίζονται τα δικά τους δεδομένα. Αυτές οι υπηρεσίες επικοινωνούν μεταξύ τους μέσω APIs (application programming interfaces), τα οποία λειτουργούν ως συνδετικός κρίκος, προσφέροντας ταυτόχρονα απλοποιημένη συντήρηση και ασφάλεια. Οι cloud-native εφαρμογές υλοποιούνται μέσω περιεκτών (containers), δηλαδή λογισμικού που απομονώνει λογικά τις εφαρμογές και επιτρέπει τη λειτουργία τους ανεξάρτητα από τους φυσικούς πόρους. Οι περιέκτες αποτρέπουν τις μικροϋπηρεσίες από το να επηρεάζουν η μία την άλλη, διασφαλίζουν την καλύτερη κατανομή πόρων και υποστηρίζουν τη δημιουργία πολλαπλών αντιγράφων της ίδιας υπηρεσίας. Η διαχείριση αυτών των περιεκτών γίνεται μέσω δυναμικής ενορχήστρωσης, με τη χρήση εργαλείων που αναλαμβάνουν την κατανομή του φόρτου εργασίας, τη διαχείριση των πόρων, τη διαθεσιμότητα και την ανάθεση των περιεκτών στους κατάλληλους διακομιστές.

Επιπλέον, οι cloud-native εφαρμογές υποστηρίζουν τις πρακτικές της συνεχούς ενοποίησης και συνεχούς παράδοσης (CI/CD), οι οποίες αυτοματοποιούν τη δοκιμή, τη συσκευασία και την ανάπτυξη των εφαρμογών σε διάφορα περιβάλλοντα. Η υποστήριξη CI/CD επιτρέπει ταχύτερη διάθεση νέων χαρακτηριστικών και ενημερώσεων, μειώνοντας τους κύκλους ζωής των

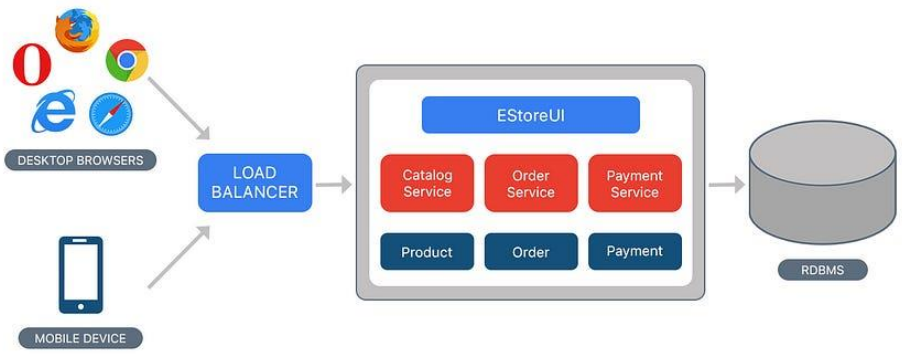
εφαρμογών και αυξάνοντας την ευελιξία των ομάδων ανάπτυξης. Τέλος, οι cloud-native εφαρμογές προσφέρουν ευελιξία στην επιλογή γλωσσών και πλαισίων ανάπτυξης, επιτρέποντας στους προγραμματιστές να δημιουργήσουν διαφορετικά μέρη της εφαρμογής με τις τεχνολογίες που εξυπηρετούν καλύτερα κάθε ανάγκη. Για παράδειγμα, το περιβάλλον χρήστη μπορεί να αναπτυχθεί με Node.js, ενώ τα APIs να υλοποιηθούν σε Java, καθιστώντας την αρχιτεκτονική προσαρμόσιμη και πολυγλωσσική.

## 2.3.2 Μικροϋπηρεσίες

### 2.3.2.1 Μονολιθική Αρχιτεκτονική

Το παραδοσιακό μοντέλο ανάπτυξης εφαρμογών ονομάζεται πλέον “μονολιθικό”, όπου τα επιμέρους κομμάτια ενώνονται σε ένα πρόγραμμα μίας πλατφόρμας και δεν μπορούν να εκτελεστούν ανεξάρτητα [19].

Στην Εικόνα 2-5 φαίνεται η μονολιθική αρχιτεκτονική που χρησιμοποιείται για την εφαρμογή ενός ηλεκτρονικού καταστήματος. Η εφαρμογή αποτελείται από μικρότερα κομμάτια που εκτελούνται από το ίδιο πρόγραμμα και συνδέονται με μία κοινή βάση δεδομένων [20].



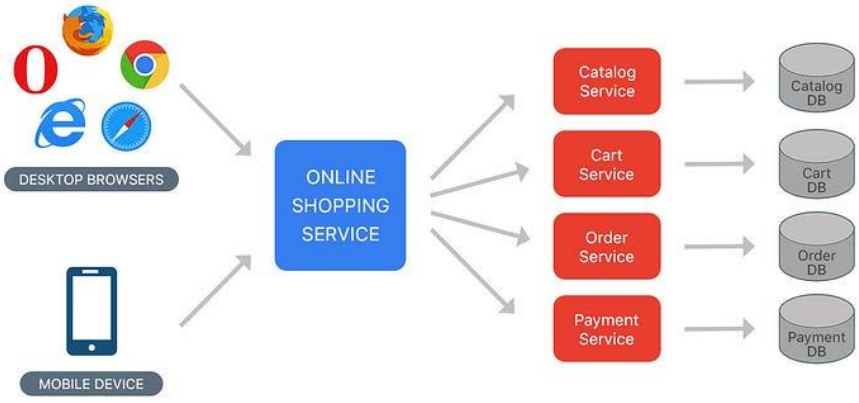

*Εικόνα 2-5 Μονολιθική Αρχιτεκτονική*

Η μονολιθική αρχιτεκτονική προσφέρει ευκολία στην ανάπτυξη και στην δοκιμή εφαρμογών και οριζόντια κλιμάκωση των απαιτούμενων πόρων για την διαχείριση πολλαπλών αντιγράφων. Ωστόσο, με το πέρασμα του χρόνου και καθώς οι εφαρμογές αναπτύσσονται, η μονολιθική αρχιτεκτονική παρουσιάζει δισεπίλυτα προβλήματα. Όταν οι εφαρμογές μεγαλώνουν και γίνονται πιο περίπλοκες, είναι δύσκολη η συντήρηση τους, καθώς κάθε μικρή αλλαγή μπορεί να επιφέρει αλλαγές σε ολόκληρο το πρόγραμμα. Επίσης, για κάθε ενημέρωση μίας εφαρμογής απαιτείται να γίνει επανέκδοση ολόκληρης της εφαρμογής. Στις μονολιθικές εφαρμογές είναι δύσκολο να υπολογιστούν οι απαιτούμενοι πόροι λόγω των αντικρουόμενων αναγκών πόρων των επιμέρους κομματιών μίας εφαρμογής. Τέλος, είναι δύσκολο να προσαρμοστούν σε νέες τεχνολογίες, αφού αλλαγές στην γλώσσα ή στο περιβάλλον ανάπτυξης μίας εφαρμογής επηρεάζουν ολόκληρη την εφαρμογή [20].

### 2.3.2.2 Αρχιτεκτονική Μικροϋπηρεσιών

Στην αρχιτεκτονική μικροϋπηρεσιών, οι εφαρμογές αναπτύσσονται ως ένας συνδυασμός από μικρότερα κομμάτια, τις μικροϋπηρεσίες. Η κάθε μικροϋπηρεσία έχει συγκεκριμένο στόχο και χρησιμοποιεί μία απλή διεπαφή για την επικοινωνία με τις υπόλοιπες μικροϋπηρεσίες. Επίσης, έχει τη δική της βάση δεδομένων που εγγυάται λιγότερη εξάρτηση μεταξύ των μικροϋπηρεσίων. Οι βάσεις δεδομένων τους μπορούν να διαφέρουν ανάλογα με τις ανάγκες της κάθε μικροϋπηρεσίας.

Στην Εικόνα 2-6 φαίνεται η αρχιτεκτονική μικροϋπηρεσιών που χρησιμοποιείται για την εφαρμογή ενός ηλεκτρονικού καταστήματος. Κάθε κομμάτι της εφαρμογής ορίζεται ως μία ξεχωριστή χαλαρά συνδεδεμένη υπηρεσία ανάλογα με τις απαιτήσεις, που μπορούν να συνεργαστούν μεταξύ τους ανάλογα με την περίσταση.

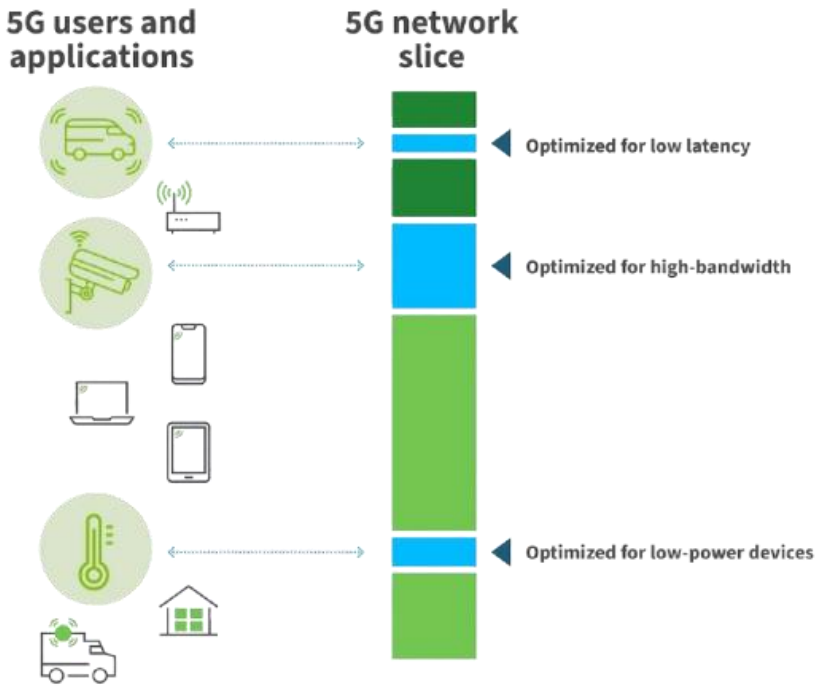



*Εικόνα 2-6 Αρχιτεκτονική Μικροϋπηρεσιών*

Οι μικροϋπηρεσίες, σε αντίθεση με την μονολιθική αρχιτεκτονική, επιτρέπουν την συνεχή ανάπτυξη μεγάλων και περίπλοκων εφαρμογών βελτιώνοντας την ευκολία δοκιμών και ανάπτυξης των δυνατοτήτων των εφαρμογών, καθώς αποτελούνται από μικρότερα, ταχύτερα και ανεξάρτητα κομμάτια. Επίσης, είναι πιο εύκολη η οργάνωση της ανάπτυξης εφαρμογών μεταξύ διαφορετικών ομάδων, όπου η καθεμία είναι υπεύθυνη για ένα συγκεκριμένο κομμάτι της εφαρμογής. Οι μικροϋπηρεσίες περιορίζουν τυχόν προβλήματα που μπορεί να προκύψουν κατά την διάρκεια εκτέλεσης μίας εφαρμογής, όπως διαρροές μνήμης, σε ένα μόνο κομμάτι, ενώ σε μία μονολιθική εφαρμογή το ίδιο πρόβλημα θα μπορούσε να δημιουργήσει πρόβλημα σε ολόκληρη την εφαρμογή. Τέλος, απαλλάσσουν τις εφαρμογές από οποιαδήποτε μελλοντική εξάρτηση σε υπάρχουσες τεχνολογίες, αφού η κάθε μικροϋπηρεσία μπορεί να χρησιμοποιεί διαφορετική τεχνολογία και λόγω της μικρότερης έκτασης της μικροϋπηρεσίας συγκριτικά με ολόκληρη την εφαρμογή, η αλλαγή σε μία νέα τεχνολογία απαιτεί σημαντικά λιγότερο χρόνο [20].

## 2.4 Τεμαχισμός Δικτύου

Ο τεμαχισμός δικτύου (network slicing) είναι μία τεχνολογία για τη δημιουργία πολλαπλών εικονικών δικτύων πάνω από μία κοινή υποδομή, όπου το κάθε εικονικό υποδίκτυο (τεμάχιο) να είναι λογικά διαχωρισμένο από τα υπόλοιπα με τη δυνατότητα να παίρνει ξεχωριστά παραμέτρους και να λαμβάνει συγκεκριμένους πόρους. Έτσι, κάθε τεμάχιο μπορεί να βελτιστοποιηθεί ανεξάρτητα για την εξυπηρέτηση μίας συγκεκριμένης εφαρμογής (ή μέρους των μικροϋπηρεσιών της εφαρμογής), υπηρεσιών, ομάδων χρηστών κ.λπ., χρησιμοποιώντας ορισμένους δικτυακούς πόρους όπως το εύρος ζώνης (bandwidth) και δικτυακές λειτουργίες (network functions). Η δημιουργία ευέλικτων και επεκτάσιμων δικτυακών τεμαχίων γίνεται εφικτή με την υιοθέτηση τεχνολογιών, όπως η δικτύωση καθοριζόμενη από λογισμικό (Software Defined Networking SDN) και η εικονικοποίηση δικτυακών λειτουργιών (Network Functions Virtualization NFV) [21]. Η τοποθέτηση της εφαρμογής σε περιέκτες (containers) εξασφαλίζει την εξατομικευμένη διαχείριση υπολογιστικών πόρων για κάθε μικροϋπηρεσία ξεχωριστά, ενώ ο τεμαχισμός επιτρέπει την ανεξάρτητη διαχείριση των δικτυακών πόρων. Η τεχνολογία αυτή, ενώ δεν αποτελεί εγγενές χαρακτηριστικό της νεφογενούς αρχιτεκτονικής, θεωρείται απαραίτητη για την υιοθέτησή των νεφογενούς μοντέλου, ιδίως σε 5G περιβάλλοντα.




*Εικόνα 2-7 Τεμαχισμός Δικτύου με Διαφορετικά Κριτήρια Βελτιστοποίησης ανά Τεμάχιο*

## 2.5 Εικονικοποίηση

Η εικονικοποίηση (virtualization) ορίζεται ως η τεχνολογία που επιτρέπει την δημιουργία λογικών υπηρεσιών μέσω πόρων ενός φυσικού υλικού (hardware) [22]. Με άλλα λόγια, επιτρέπει την κατανομή των δυνατοτήτων ενός υπολογιστικού πόρου ή μίας φυσικής μηχανής σε διάφορους χρήστες ή περιβάλλοντα. Υπάρχουν διαφορετικοί τύποι εικονικοποίησης, όπως εικονικοποίηση δικτύου (network virtualization), αποθηκευτικού χώρου (storage virtualization), εξυπηρετητή (server virtualization), εφαρμογής (application virtualization). Οι κύριες τεχνολογίες εικονικοποίησης που προσφέρουν αφηρημένα υπολογιστικά περιβάλλοντα είναι η εικονικοποίηση πλατφόρμας (platform virtualization) και η εικονικοποίηση λειτουργικού συστήματος (operating system-level virtualization).

Από τη μία μεριά, η εικονικοποίηση πλατφόρμας παρέχει στους χρήστες υπολογιστικά περιβάλλοντα, τα οποία ονομάζονται εικονικές μηχανές (virtual machines VMs), που τρέχουν πάνω από ένα λειτουργικό σύστημα οικοδεσπότη σε ένα φυσικό σύστημα. Το κάθε VM έχει το δικό του λειτουργικό σύστημα και λειτουργεί ανεξάρτητα από τα υπόλοιπα. Από την άλλη μεριά, η εικονικοποίηση λειτουργικού συστήματος παρέχει απομονωμένα υπολογιστικά περιβάλλοντα, που ονομάζονται περιέκτες (containers) σε ένα κοινό λειτουργικό σύστημα. Ένας περιέκτης είναι μία ομάδα από μία ή περισσότερες διεργασίες που είναι ανεξάρτητες από το υπόλοιπο σύστημα και αποτελούνται από μία εφαρμογή, όλες τις βιβλιοθήκες από τις οποίες εξαρτάται και τα αρχεία ρυθμίσεών της. Αυτή τη περίοδο, η πιο διαδεδομένη μέθοδος OS-level εικονικοποίησης είναι η χρήση του Docker.

### 2.5.1 Εικονικές Μηχανές

Μια εικονική μηχανή (Virtual Machine VM) είναι το λειτουργικό που τρέχει προγράμματα και εφαρμογές χωρίς να είναι συνδεδεμένο σε μία φυσική μηχανή [23]. Σε έναν υπολογιστή οικοδεσπότη (host) μπορούν να συνυπάρχουν μία ή περισσότερες φιλοξενούμενες εικονικές μηχανές (guest). Κάθε VM έχει το δικό του λειτουργικό σύστημα και δουλεύει ανεξάρτητα από άλλα VMs, ακόμα και αν βρίσκονται στο ίδιο φυσικό μηχάνημα.

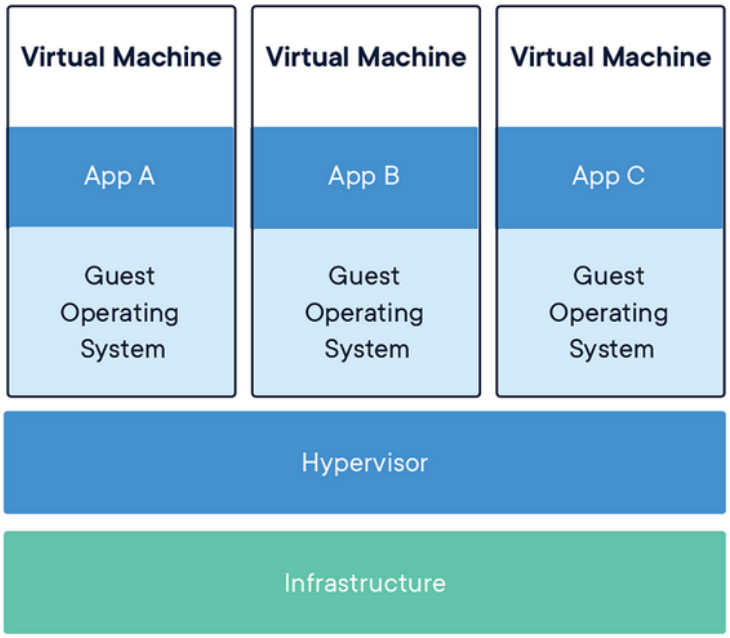
Η χρήση VMs έχει αυξηθεί με την υιοθέτηση εικονικοποίησης εξυπηρετητών (server virtualization) για την αξιοποίηση της υπολογιστικής δύναμης των φυσικών εξυπηρετητών πιο αποδοτικά, μειώνοντας των αριθμό των φυσικών εξυπηρετητών και σώζοντας χώρο στα κέντρα δεδομένων. Εφαρμογές με διαφορετικές απαιτήσεις λειτουργικού συστήματος μπορούν να τρέξουν σε ένα φυσικό μηχάνημα, έτσι διαφορετικό υλικό εξυπηρετητών δεν είναι απαραίτητο για κάθε εφαρμογή.

Ένας hypervisor, ή αλλιώς Virtual Machine Monitor (VMM), είναι μια διεργασία λογισμικού που δημιουργεί και χειρίζεται τις εικονικές μηχανές και κατανέμει τους υπολογιστικούς, αποθηκευτικούς και δικτυακούς πόρους του οικοδεσπότη σε κάθε VM [24]. Οι hypervisors εξασφαλίζουν ότι κάθε VM λειτουργεί ανεξάρτητα, αποτρέποντας συγκρούσεις και βελτιστοποιώντας την επίδοση σε πολλαπλές εργασίες. Η εικονικοποίηση αναφέρεται στην χρήση λογισμικού για την προσομοίωση φυσικών πόρων, για τα VMs οι hypervisors είναι υπεύθυνοι για την αφηρημένη και απομονωμένη λειτουργία της πάνω από τους φυσικούς πόρους ενός μηχανήματος.

Οι hypervisors διακρίνονται σε δύο κύριους τύπους τους τύπου 1 και τους τύπου 2. Οι τύπου 1 hypervisors, αλλιώς bare metal hypervisor, λειτουργούν απευθείας στον φυσικό εξυπηρετητή έχοντας άμεση πρόσβαση στους πόρους του, καθιστώντας τους πιο αποδοτικούς και ασφαλείς. Οι τύπου 1 hypervisors είναι οι πιο διαδεδομένοι σε μεγάλα κέντρα δεδομένων λόγο της υψηλής ασφάλειας, της σταθερότητας και της δυνατότητας κλιμάκωσης που προσφέρουν. Οι τύπου 2 hypervisors, αλλιώς hosted hypervisors, λειτουργούν ως εφαρμογή πάνω στο προϋπάρχων λειτουργικό σύστημα ενός φυσικού εξυπηρετητή. Με ένα type 2 hypervisor, χειροκίνητα δημιουργείται ένα VM και εγκαθίσταται το λειτουργικό σύστημα σε αυτό, έτσι οι χρήστες μπορούν να επιλέξουν χειροκίνητα το πλήθος από υπολογιστικούς πυρήνες (CPU cores) και μνήμη που επιθυμούν. Οι type 2 hypervisors δεν είναι ιδανικοί για server-based περιβάλλοντα λόγω του μεγαλύτερου χρόνου απόκρισης και των κινδύνων ασφαλείας, ωστόσο λόγω της απλότητας στην εγκατάσταση και την χρήση τους προτιμάται σε ορισμένες περιπτώσεις, όπως σε προσωπικούς υπολογιστές που χρειάζεται να τρέξουν περισσότερα από ένα λειτουργικά συστήματα και όπου η απόδοση και η ασφάλεια δεν είναι τόσο σημαντικά.

Λόγω της ευελιξίας και της φορητότητάς τους, οι εικονικές μηχανές προσφέρουν σημαντικά πλεονεκτήματα έναντι των παραδοσιακών φυσικών υπολογιστών. Με την ταυτόχρονη εκτέλεση πολλαπλών εικονικών περιβαλλόντων στο ίδιο φυσικό σύστημα, αξιοποιούνται καλύτερα οι πόροι και περιορίζεται η ανάγκη για επιπλέον αφιερωμένα μηχανήματα. Η δυνατότητα ταχείας δημιουργίας εικονικών μηχανών καθιστά την ανάπτυξη και χρήση εφαρμογών πιο γρήγορη και αποτελεσματική, χωρίς να απαιτείται ξεχωριστός φυσικός εξοπλισμός. Επιπλέον, σε περιπτώσεις βλάβης, οι εικονικές μηχανές μπορούν εύκολα να μεταφερθούν σε άλλον hypervisor, ελαχιστοποιώντας τον νεκρό χρόνο και αυξάνοντας την ανθεκτικότητα. Η επεκτασιμότητα αποτελεί επίσης βασικό πλεονέκτημα, καθώς οι πόροι μπορούν να προσαρμόζονται δυναμικά ανάλογα με τις ανάγκες των εφαρμογών και οι διεργασίες να διανέμονται σε περισσότερες εικονικές μηχανές. Τέλος, η απομόνωση των εικονικών μηχανών ενισχύει την ασφάλεια, καθώς τυχόν

κακόβουλο λογισμικό δεν μπορεί να επηρεάσει άλλες εικονικές μηχανές ή τον οικοδεσπότη.




*Εικόνα 2-8 Εικονικές Μηχανές πάνω σε μια Υποδομή*

## 2.5.2 Περιέκτες

Οι περιέκτες (Containers) αποτελούν μία μέθοδο για δημιουργία, πακετάρισμα και ανάπτυξη λογισμικού [25]. Σε αντίθεση με τις παραδοσιακές εικονικές μηχανές, οι περιέκτες δεν περιέχουν ολόκληρο το λειτουργικό σύστημα, αλλά χρησιμοποιούν εικονικοποίηση λειτουργικού συστήματος, δηλαδή μοιράζονται τον ίδιο πυρήνα και τις βιβλιοθήκες του οικοδεσπότη και δημιουργούν απομονωμένες διαδικασίες, όπου η καθεμία χρησιμοποιεί ανεξάρτητα υπολογιστικούς πόρους.

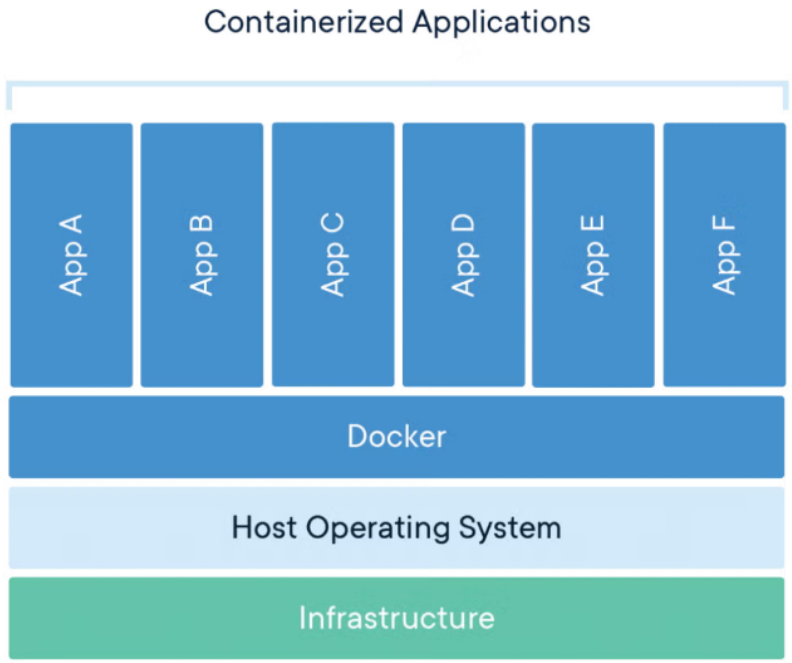
Μια εικόνα περιέκτη (container image) συγκεντρώνει όλα τα απαραίτητα κομμάτια, όπως ο κώδικας μίας μικροϋπηρεσίας και οι εξαρτήσεις της, για την δημιουργία ενός περιέκτη σε ένα λειτουργικό σύστημα και αποτελείται από διαφορετικά επίπεδα εικόνας (image layers) που στοιβάζονται το ένα πάνω στο άλλο. Οι εικόνες περιέκτη είναι αμετάβλητες και μοιράζονται τις ίδιες λειτουργίες ως πρότυπα [26].

Οι εικόνες περιέκτη αποθηκεύονται σε ένα μητρώο εικόνας περιέκτη (container image registry), ή μητρώο περιέκτη (container registry), όπου λειτουργεί ως ένα είδος συστήματος αρχείων. Τα μητρώα περιέκτη είναι αποθήκες δεδομένων (data repositories), όπου διατηρούν εικόνες περιέκτη για αποθήκευση και εύκολη πρόσβαση.

Μια εφαρμογή που δομείται με τη χρήση περιεκτών (containerized application), μπορεί να αποτελείται από πολλές εικόνες, οι οποίες εκτελούνται ανεξάρτητα.

Κατά την έναρξη της εφαρμογής, οι εικόνες μετατρέπονται σε στιγμιότυπα περιέκτη (container instances), όπου διαφορετικά στιγμιότυπα μίας ίδιας εικόνας μπορούν να εκτελούνται ταυτόχρονα, ενώ ένα στιγμιότυπο μπορεί να αντικατασταθεί από καινούργιο χωρίς διακοπή της λειτουργίας της εφαρμογής σε περίπτωση δυσλειτουργίας του στιγμιότυπου.

Η χρήση των περιεκτών προσφέρει σημαντικά πλεονεκτήματα λόγω του μικρού τους μεγέθους και της πλήρης πληροφορίας που περιέχεται σε κάθε εικόνα για την εκτέλεση της. Αρχικά, προσδίδει εξαιρετική φορητότητα και προσαρμοστικότητα, καθώς η ίδια εικόνα μπορεί να εκτελείται χωρίς τροποποιήσεις τόσο σε απομακρυσμένες εικονικές μηχανές όσο και σε ιδιόκτητα υπολογιστικά συστήματα. Αυτό ενισχύει την παραγωγικότητα των προγραμματιστών, καθώς τους επιτρέπει να αναπτύσσουν κώδικα με συνέπεια, χωρίς να ανησυχούν για την εκτέλεσή του σε διαφορετικά περιβάλλοντα από τον τοπικό υπολογιστή, σε τοπικούς εξυπηρετητές, μέχρι και το δημόσιο cloud. Επιπλέον, μπορούν να δημιουργηθούν πολλαπλά στιγμιότυπα της ίδιας εικόνας, προκειμένου να εξυπηρετηθούν περισσότεροι χρήστες ή να μειωθεί η καθυστέρηση φέρνοντας την τοποθεσία εκτέλεσης της εικόνας πιο κοντά στον χρήστη. Επίσης, οι περιέκτες επιταχύνουν την ανάπτυξη και την ενημέρωση εφαρμογών, ιδίως στην αρχιτεκτονική μικροϋπηρεσιών, όπου οι μικροϋπηρεσίες μπορούν να αναπτυχθούν, να τροποποιηθούν ή να αποσυρθούν ανεξάρτητα χωρίς να απαιτείται να ενημερωθεί ολόκληρη η εφαρμογή. Τέλος, οι περιέκτες απαιτούν λιγότερους υπολογιστικούς πόρους, επιτρέποντας την ταυτόχρονη εκτέλεση πολλών περιεκτών σε ένα μόνο φυσικό σύστημα.




*Εικόνα 2-9 Περιέκτες πάνω σε μια Υποδομή*

### 2.5.3 Σύγκριση Εικονικών Μηχανών και Περιεκτών

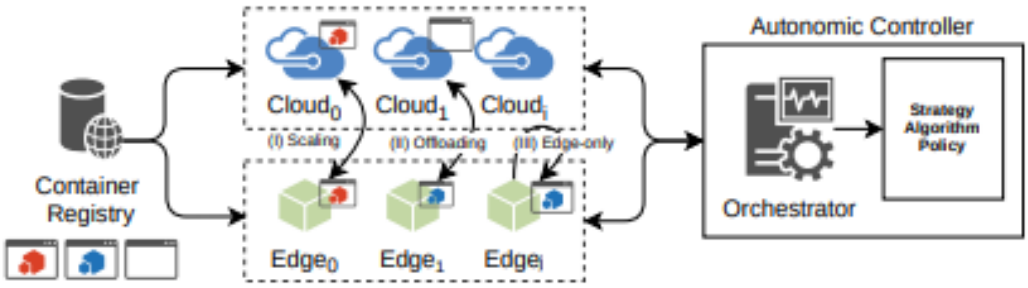
Οι περιέκτες και οι εικονικές μηχανές είναι πολύ παρόμοιες τεχνολογίες εικονικοποίησης πόρων, ωστόσο υπάρχουν σημαντικές διαφορές μεταξύ τους [27]. Αρχικά, διαφέρουν ως προς την αρχιτεκτονική τους, οι εικονικές μηχανές εκτελούνται πάνω από hypervisors και περιλαμβάνουν το δικό τους λειτουργικό σύστημα, ενώ οι περιέκτες μοιράζονται τον πυρήνα του λειτουργικού συστήματος του οικοδεσπότη και πακετάρουν μόνο την εφαρμογή και τις εξαρτήσεις της. Οι εικονικές μηχανές έχουν μεγαλύτερη κατανάλωση πόρων λόγω του επιπλέον λειτουργικού συστήματος από τους περιέκτες που δεν απαιτούν ξεχωριστό λειτουργικό σύστημα για κάθε στιγμιότυπο. Επίσης, οι εικονικές μηχανές έχουν μεγαλύτερο χρόνο εκκίνησης, καθώς απαιτείται η φόρτωση και εκκίνηση του δικού τους λειτουργικού συστήματος, ενώ οι περιέκτες έχουν πολύ μικρότερο φόρτο και μπορούν να εκκινούν μέσα σε λίγα δευτερόλεπτα. Οι εικονικές μηχανές προσφέρουν υψηλότερο επίπεδο ασφάλειας σε σύγκριση με τους περιέκτες λόγω του βαθμού απομόνωσης που παρέχουν, οι εικονικές μηχανές εξασφαλίζουν πλήρη απομόνωση από τις υπόλοιπες, ενώ όλοι οι περιέκτες καθίστονται ευάλωτοι αν παραβιαστεί ο πυρήνας του λειτουργικού συστήματος. Τέλος, Οι εικονικές μηχανές είναι ογκώδεις και μπορεί να είναι δύσκολο να μετακινηθούν λόγω προβλημάτων συμβατότητας μεταξύ διαφορετικών περιβαλλόντων, αντίθετα οι περιέκτες μπορούν να μετακινηθούν μεταξύ διαφορετικών περιβαλλόντων με ελάχιστη προσπάθεια και χωρίς προβλήματα συμβατότητας. Συμπερασματικά, συνήθως η χρήση εικονικών μηχανών προτιμάται σε εφαρμογές με συγκεκριμένες απαιτήσεις υπολογιστικού υλικού ή όταν δεν χρειάζεται να εγγυηθεί η συμβατότητα μεταξύ διαφορετικών λειτουργικών, ενώ οι περιέκτες προτιμώνται σε εφαρμογές με απαιτήσεις που αφορούν κυρίως το λογισμικό. Η χρήση περιεκτών ενδείκνυνται για την ενθυλάκωση μικροϋπηρεσιών αφού πληθαίνει ο αριθμός των παράλληλων συστημάτων που μπορούν να τρέχουν στον ίδιο οικοδεσπότη.

### 2.5.4 Ενορχήστρωση Περιεκτών

Η εκτέλεση εφαρμογών με χρήση περιεκτών μπορεί να εξελιχθεί σε μια ιδιαίτερα απαιτητική διαδικασία, ειδικά όταν χρησιμοποιούνται σε συνδυασμό με μικροϋπηρεσίες, οι οποίες συνήθως εκτελούνται η καθεμία σε ξεχωριστό περιέκτη. Μια εφαρμογή βασισμένη σε περιέκτες μπορεί να απαιτεί την ταυτόχρονη διαχείριση εκατοντάδων ή και χιλιάδων περιεκτών, κυρίως κατά την ανάπτυξη και την λειτουργία συστημάτων μεγάλης κλίμακας, δημιουργώντας συνεπώς την ανάγκη για την ύπαρξη ενός προγράμματος για την αυτοματοποίηση του συντονισμού και της διαχείρισής τους. Τέτοια λογισμικά που αυτοματοποιούν τις περισσότερες λειτουργίες που απαιτούνται για την εύρυθμη εκτέλεση μίας containerized εφαρμογής ονομάζονται ενορχηστρωτές περιεκτών (container

orchestrators)[25]. Οι ενορχηστρωτές περιεκτών είναι υπεύθυνοι για την παρατήρηση της ορθής λειτουργίας των περιεκτών, την επανεκκίνηση τους σε περίπτωση μη ορθής εκτέλεσης και την δέσμευση και την κλιμάκωση των υπολογιστικών πόρων για την ανταπόκριση στις ανάγκες τοι εισερχόμενου φόρτου εργασίας.

Η χρήση edge cloud υποδομών για την εκτέλεση μεγάλου πλήθους διαφορετικών υπηρεσιών συνοδεύεται από νέες πολυπλοκότητες, λόγω της προσθήκης πολλών και ετερόκλητων edge τοποθεσιών. Η ενορχήστρωση σε περιβάλλοντα edge cloud είναι επίσης υπεύθυνη για την κατάλληλη κατανομή των φόρτων εργασίας ανάμεσα στο cloud, edge και IoT επίπεδο. Ένας ενορχηστρωτής περιεκτών αποφασίζει την τοποθέτηση τους ανάλογα με τους στόχους της εκάστοτε εφαρμογής, προσπαθώντας να ικανοποιήσει της απαιτήσεις τους όσον αφορά τον χρόνο απόκρισης (latency), το εύρος ζώνης (bandwidth) κ.α, και λαμβάνοντας υπόψη τη ζήτηση και τη διαθεσιμότητα πόρων, όπως CPU, μνήμη, δίσκο και χρήση δικτύου.




*Εικόνα 2-10 Αρχιτεκτονική Ενορχηστρωτή Περιεκτών σε Edge Cloud Υποδομή*

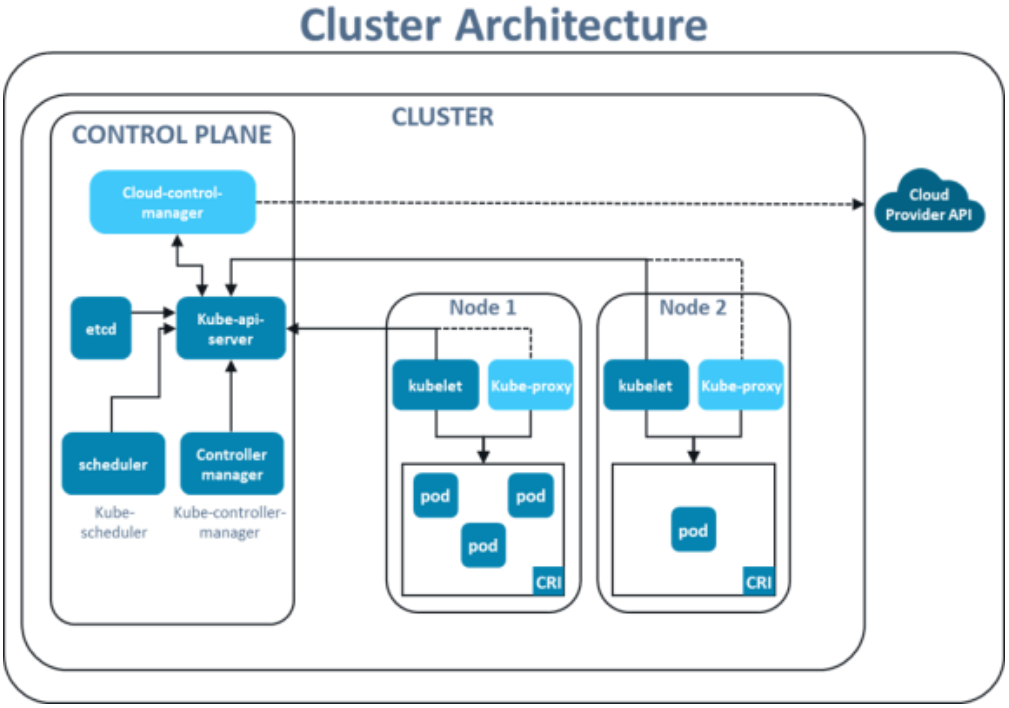
## 2.5.5 Κυβερνήτης

Ο πιο διαδεδομένος ενορχηστρωτής σήμερα είναι ο Κυβερνητής (Kubernetes ή K8s), ένα ανοιχτού κώδικα λογισμικό, αρχικά σχεδιασμένο από την Google και αποτελεί άμεση εξέλιξη του Borg. Ο Κυβερνήτης προσφέρει εκτενείς δυνατότητες διαχείρισης περιεκτών και είναι ιδιαίτερα επεκτάσιμος και φορητός, κάτι που του επιτρέπει να εκτελείται σε ευρύ φάσμα περιβαλλόντων. Επίσης, δίνει τη δυνατότητα σε προγραμματιστές και διαχειριστές να καθορίζουν την επιθυμητή κατάσταση ενός συστήματος, και αυτός αναλαμβάνει να εκτελέσει δυναμικά αυτήν την κατάσταση, εξασφαλίζοντας ότι το σύστημα λειτουργεί όπως έχει οριστεί [25].

Όταν εκτελείται ο Κυβερνητής [28], δημιουργείται μια συστάδα (cluster) αποτελούμενη από ένα σύνολο κόμβων-εργατών (worker nodes), καθένας εκ των οποίων αποτελείται από το kubelet, το kube-proxy και τη μηχανή περιεκτών (container runtime). Το kubelet είναι ένας πράκτορας (agent) που είναι υπεύθυνος για την ορθή λειτουργία των περιεκτών που βρίσκονται εντός ενός pod. Τα pods είναι ένα σύνολο από έναν ή περισσότερους περιέκτες εντός του ίδιου κόμβου και αποτελούν τη βασική διαχειρίσιμη μονάδα του Κυβερνήτη. Το kube-proxy είναι

ένας δικτυακός μεσολαβητής (network proxy) που αποφασίζει για τη δικτύωση, όπως τη δρομολόγηση της κίνησης, των περιεκτών. Τέλος, η μηχανή περιεκτών (container runtime) είναι το λογισμικό για τη δημιουργία των περιεκτών, το πιο διαδεδομένο τέτοιο λογισμικό είναι το Docker.

Πέρα από τους κόμβους-εργάτες μία συστάδα περιέχει και έναν κόμβο ελέγχου (master node) που αποτελείται από το kube-api-server, το etcd, το kube-controller-manager και το kube-scheduler. Το kube-api-server είναι το front-end του πεδίου ελέγχου της συστάδας. Το etcd είναι μία κατανεμημένη μέθοδος αποθήκευσης κλειδιών τιμών (key value) που αποθηκεύει την παραμετροποίηση της συστάδας και αντικατοπτρίζει τη συνολική κατάστασή της σε κάθε χρονική στιγμή. Το kube-controller-manager εκτελεί τις διαδικασίες ελέγχου για την επίτευξη και την διατήρηση της «ποθητής κατάστασης» (desired state) που έχει ορίσει ο χρήστης μέσω του API, εκτελώντας διαδικασίες όπως η δημιουργία και η καταστροφή pods. Το kube-scheduler αποφασίζει τον κατάλληλο κόμβο για την τοποθέτηση των καινούργιων pods σύμφωνα με την διαθεσιμότητα των πόρων και με τις παραμέτρους που έχουν εισάγει οι εφαρμογές, όπως η ζητούμενη ποιότητα υπηρεσίας (quality of service QoS).




*Εικόνα 2-11 Αρχιτεκτονική Συστάδας Κυβερνήτη*

Ο Κυβερνήτης προσφέρει μία σειρά από λειτουργίες για την αποτελεσματική διαχείριση containerized εφαρμογών, όπως αυτόματη κατανομή φορτίων, εξισορρόπηση φόρτου εργασίας, ενορχήστρωση αποθήκευσης είτε σε τοπικές είτε σε cloud εγκαταστάσεις (storage orchestration), αυτόματη αποκατάσταση σε περίπτωση σφάλματος και αυτόματη κλιμάκωση (autoscaling) [29]. Ωστόσο, ο

Κυβερνήτης, όπως και άλλοι παρόμοιοι διαχειριστές, εστιάζουν σε μεγάλα κέντρα δεδομένων και δεν είναι σχεδιασμένοι για μία γεωγραφική διεσπαρμένη υποδομή, όπως οι edge cloud υποδομές. Συνεπώς, δημιουργείται η ανάγκη για τη δημιουργία νέων διαχειριστών πόρων που να μπορούν να ανταπεξέλθουν στον έλεγχο και στην κατάλληλη κατανομή υπολογιστικών πόρων μίας σύγχρονης κατανεμημένης υποδομής.

## 2.6 Εφαρμογές Edge Cloud Τεχνολογίας

### 2.6.1 5G στο Edge Cloud

Το 5G και το edge computing είναι δύο στενά συνδεδεμένες τεχνολογίες, οι οποίες αλληλοσυμπληρώνονται και οδηγούν στη βελτιστοποίηση των σύγχρονων υπολογιστικών και τηλεπικοινωνιακών υποδομών. Το 5G προσφέρει πλεονεκτήματα όπως χαμηλή καθυστέρηση (low latency), μεγάλο εύρος ζώνης (high bandwidth) και μεγάλη χωρητικότητα (large capacity), επιλύοντας προβλήματα των παραδοσιακών συστημάτων επικοινωνίας, αλλά παράλληλα οδηγεί σε ταχεία αύξηση του όγκου δεδομένων [10].

Το edge computing μπορεί να κατανείμει έξυπνα τους πόρους μεταξύ συσκευών IoT, edge συσκευών και cloud υποδομών, λαμβάνοντας υπόψη την εμπειρία χρήστη, την κατανάλωση ενέργειας, το υπολογιστικό φορτίο, την απόδοση και το κόστος. Αυτή η σχέση καθιστά την ανάπτυξη του edge computing άρρηκτα συνδεδεμένη με το 5G. Από τη μία πλευρά, το edge computing υποστηρίζει το 5G, καθώς αποτελεί βασικό στοιχείο της αρχιτεκτονικής του. Από την άλλη πλευρά, το 5G είναι βασισμένο σε λογισμικό, επιτρέποντας την ευέλικτη εφαρμογή του edge computing.

### 2.6.2 Multi-access Edge Computing (MEC)

Το Multi-access Edge Computing (MEC), γνωστό παλαιότερα ως Mobile Edge Computing, είναι ένας τύπος edge computing που επεκτείνει τις δυνατότητες του cloud computing, φέρνοντάς το πιο κοντά στο άκρο του δικτύου. Το MEC προέκυψε από την πρωτοβουλία του Ευρωπαϊκού Ινστιτούτου Τηλεπικοινωνιακών Προτύπων (ETSI), η οποία αρχικά επικεντρώθηκε στην τοποθέτηση edge nodes στο κινητό δίκτυο, αλλά πλέον έχει επεκταθεί και στα σταθερά δίκτυα. Ενώ το παραδοσιακό cloud computing πραγματοποιείται σε απομακρυσμένους διακομιστές, που βρίσκονται μακριά από τον χρήστη και τη συσκευή, το MEC επιτρέπει την εκτέλεση διεργασιών σε σταθμούς βάσης, κεντρικά γραφεία και άλλα σημεία συγκέντρωσης δεδομένων στο δίκτυο [30].

Το MEC, το οποίο αναπτύσσεται ως βασικό στοιχείο του 5G, επιτρέπει τη διαχείριση δεδομένων σε πραγματικό χρόνο και τη βελτίωση της εμπειρίας χρήστη. Το MEC προκύπτει ως φυσική εξέλιξη της τεχνολογίας των σταθμών

βάσης και της ενοποίησης των τηλεπικοινωνιακών και IT δικτύων. Τέλος, το 5G και το MEC προωθούν την εξέλιξη του IoT, ενισχύοντας εφαρμογές όπως δικτυακή διαχείριση IoT, έξυπνα δίκτυα οχημάτων και machine learning για επιλογή κινητών κόμβων διανομής δεδομένων.

### 2.6.3 Mixed Reality, Artificial Reality and Virtual Reality

Οι εφαρμογές μικτής (MR), επαυξημένης (AR) και εικονικής (VR) πραγματικότητας μπορούν να επωφεληθούν από το MEC, υποστηρίζοντας απομακρυσμένους εργαζόμενους στην εκτέλεση εργασιών συντήρησης και επισκευής στο πεδίο [30]. Σήμερα, τα 3D μοντέλα είναι πολύ βαριά για να αποδοθούν από τις ίδιες τις συσκευές, ενώ η απόδοσή τους στο cloud είναι ανέφικτη λόγω υψηλής καθυστέρησης (latency).

Για παράδειγμα, μια λύση MEC μπορεί να παρέχει πληροφορίες σε πραγματικό χρόνο σχετικά με ένα συγκεκριμένο εξάρτημα που επισκευάζει ένας τεχνικός, προβάλλοντάς τες σε ένα headset ή φορητή συσκευή. Το MEC επιτρέπει την επεξεργασία δεδομένων και την απόδοση 3D μοντέλων εκτός της συσκευής, επιτρέποντας τη δημιουργία digital twin models στο οπτικό πεδίο του τεχνικού. Παράλληλα, ένας απομακρυσμένος ειδικός μπορεί να σχολιάζει και να επισημαίνει εικόνες ή βίντεο που μεταδίδονται από το headset ή τη φορητή συσκευή σε πραγματικό χρόνο.

Μια άλλη εφαρμογή MR που επιτρέπει το MEC αφορά τη συνεργασία πολλών χρηστών σε τομείς όπως η αρχιτεκτονική, η μηχανική και οι κατασκευές. Η τεχνολογία αυτή επιτρέπει την απόδοση και κοινή επεξεργασία 3D μοντέλων στο edge cloud, μειώνοντας σημαντικά την καθυστέρηση, καθώς τα σχέδια αυτά αποτελούνται από μεγάλο όγκο δεδομένων. Επιπλέον, η υλοποίηση του MEC διευκολύνει την κοινή χρήση αυτών των δεδομένων μεταξύ διαφόρων εμπλεκόμενων μερών μέσω ενός κατανεμημένου δικτύου.

### 2.6.4 Ultra Reliable and Low Latency Communications (URLLC)

Το Ultra-Reliable Low Latency Communications (URLLC) είναι ένα υποσύνολο της αρχιτεκτονικής δικτύου 5G, που διασφαλίζει πιο αποδοτικό προγραμματισμό μεταφοράς δεδομένων [32]. Επιτυγχάνει ταχύτερες μεταδόσεις χρησιμοποιώντας μεγαλύτερο υποφορέα (subcarrier) και επιτρέπει ακόμη και επικαλυπτόμενες μεταδόσεις. Το URLLC είναι ένα βασικό χαρακτηριστικό του 5G, που επιτρέπει υψηλή αξιοπιστία και χαμηλή καθυστέρηση για εφαρμογές κρίσιμης σημασίας, όπως η αυτοματοποίηση εργοστασίων, η αυτόνομη οδήγηση και η επαυξημένη/εικονική πραγματικότητα (AR/VR). Επιπλέον, η τεχνολογία URLLC υποστηρίζει πολυπλεξία (multiplexing), επιτρέποντας σε πολλούς χρήστες να μοιράζονται τους ίδιους πόρους, αυξάνοντας έτσι την αποδοτικότητα του δικτύου. Το URLLC παρέχει εγγυημένη ποιότητα υπηρεσίας (QoS) με εξαιρετικά χαμηλή

καθυστέρηση (1 ms) και υψηλή αξιοπιστία, διασφαλίζοντας άμεση και σταθερή επικοινωνία σε εφαρμογές όπου η καθυστέρηση και οι διακοπές μπορεί να έχουν σοβαρές επιπτώσεις [31].

Για παράδειγμα, τέτοιου είδους συνδεσιμότητα είναι απαραίτητη στην αυτόνομη οδήγηση που απαιτεί αντίδραση σε χρόνο αντίστοιχο των ανθρώπων, καθώς τυχόν καθυστέρηση στη μετάδοση δεδομένων ενέχει σοβαρούς κινδύνους. Η αυτόνομη οδήγηση προσφέρει οφέλη όπως εξοικονόμηση χρόνου και μείωση ατυχημάτων, απαιτεί τη διασύνδεση όλων των οχημάτων μεταξύ τους καθώς και με υποδομές, όπως φωτεινούς σηματοδότες, υπηρεσίες έκτακτης ανάγκης και συστήματα συντήρησης δρόμων. Τα δεδομένα πρέπει να ανταλλάσσονται σε πραγματικό χρόνο και με ελάχιστη καθυστέρηση, καθώς η ασφάλεια απαιτεί εξαιρετικά αξιόπιστες συνδέσεις [24].

### 2.6.5 Τεχνητή Νοημοσύνη

Το Edge AI (Τεχνητή Νοημοσύνη στο Edge) αναφέρεται στην ανάπτυξη αλγορίθμων και μοντέλων AI απευθείας σε τοπικές edge συσκευές, όπως αισθητήρες και IoT συσκευές, επιτρέποντας επεξεργασία και ανάλυση δεδομένων σε πραγματικό χρόνο, χωρίς συνεχή εξάρτηση από το cloud. Το Edge AI συνδυάζει το edge computing με την τεχνητή νοημοσύνη, επιτρέποντας την εκτέλεση εργασιών μηχανικής μάθησης σε διασυνδεδεμένες edge συσκευές. Το edge computing διατηρεί τα δεδομένα κοντά στη συσκευή, ενώ οι αλγόριθμοι AI τα επεξεργάζονται απευθείας στο άκρο του δικτύου, με ή χωρίς σύνδεση στο διαδίκτυο. Αυτό επιτρέπει την ανάλυση δεδομένων μέσα σε χιλιοστά του δευτερολέπτου, προσφέροντας άμεση απόκριση.

Η δημοτικότητα του Edge AI αυξάνεται καθώς διάφοροι κλάδοι το χρησιμοποιούν για να βελτιστοποιήσουν ροές εργασιών, αυτοματοποιήσουν επιχειρηματικές διαδικασίες και ανακαλύψουν νέες ευκαιρίες καινοτομίας, ενώ ταυτόχρονα αντιμετωπίζουν προκλήσεις όπως καθυστέρηση (latency), ασφάλεια και μείωση κόστους [33].

# 3 Σχετικές Εργασίες

Η ανάπτυξη του Internet of Things, της μηχανικής μάθησης, της εικονικής και επαυξημένης πραγματικότητας, της αυτόνομης οδήγησης, των συστημάτων ασφαλείας, των ηλεκτρονικών παιχνιδιών και άλλων υπολογιστικά απαιτητικών εφαρμογών έχουν αυξήσει την πολυπλοκότητα και τις υπολογιστικές ανάγκες των συστημάτων. Οι παραδοσιακές αρχιτεκτονικές που συγκεντρώνουν σε ένα κέντρο την απαιτούμενη υπολογιστική ισχύ, μοιάζει να μην μπορεί πάντα να ικανοποιήσει τις απαιτήσεις τους, κυρίως λόγω της καθυστέρησης που εισάγετε από την απομακρυσμένη επικοινωνία των κατανεμημένων χρηστών με κεντρικά συστήματα. Η συνδυαστική χρήση κοντινών edge υποδομών και cloud υποδομών με απεριόριστους υπολογιστικούς πόρους για την εξυπηρέτηση εφαρμογών φαίνεται ότι πλεονεκτεί σε πολλές περιπτώσεις επιτρέποντας σε πραγματικό χρόνο την παροχή υπηρεσιών που είναι αδύνατη με την χρήση κεντρικών υποδομών. Η βελτιστοποίηση κατά περίπτωση της χρήση κατανεμημένων και σε διαφορετικά επίπεδα πόρων έχει οδηγήσει τα τελευταία χρόνια σε μεγάλο αριθμό σχετικών ερευνών. Οι προσεγγίσεις του προβλήματος ποικίλλουν ανάλογα με την τοπολογία, την παράμετρο βελτιστοποίησης και τις διάφορες τεχνολογίες που εξετάζονται, χρησιμοποιώντας διαφορετικές τεχνολογίες από το ευρύτερο πεδίο των μαθηματικών και της επιστήμης των υπολογιστών.

Στην έρευνα [34] εξετάζεται το πρόβλημα βελτιστοποίησης της κατανομής υπολογιστικών πόρων σε edge cloud υποδομές με πολλά επίπεδα με στόχο την ελαχιστοποίηση του συνολικού κόστους και του συνολικού χρόνου απόκρισης της κάθε εφαρμογής λαμβάνοντας υπόψη τους περιορισμούς τους. Η κάθε cloud-native εφαρμογή αποτελείται από μικροϋπηρεσίες, όπου σχετίζονται μεταξύ τους και έχουν διαφορετικές απαιτήσεις από το σύστημα. Συγκεκριμένα, θεωρούνται διαφορετικές υπολογιστικές υποδομές ενός ιεραρχικού edge cloud συστήματος με διαφορετικές υπολογιστικές δυνατότητες, κόστη χρήσης και κόστη μεταφοράς δεδομένων, που εξυπηρετούν εφαρμογές που αποτελούνται από μικροϋπηρεσίες με διαφορετικές υπολογιστικές απαιτήσεις και αποδεκτούς χρόνος καθυστέρησης μεταξύ δύο εξ αυτών. Αναπτύσσονται τρεις αλγόριθμοι με στόχο την ελαχιστοποίηση του βεβαρημένου αθροίσματος του υπολογιστικού και δικτυακού κόστους και της συνολικής καθυστέρησης μεταφοράς δεδομένων της κάθε εφαρμογής. Αρχικά, το πρόβλημα εκφράζεται σε μορφή μικτού ακέραιου γραμμικού προγραμματισμού (MILP) για την εύρεση της βέλτιστης λύσης. Για τη μείωση της υπολογιστικής πολυπλοκότητας, αναπτύσσεται ένας άπληστος best-fit αλγόριθμος, όπου επιλέγει μία εφαρμογή, βρίσκει τις καλύτερες υποδομές για να εκτελεστεί η κάθε μικροϋπηρεσία της με βάση τους περιορισμούς, δεσμεύει τους απαιτούμενους πόρους και μετά συνεχίζει με την επόμενη εφαρμογή. Ωστόσο η επιλογή της πρώτης μικροϋπηρεσίας μπορεί να καταστήσει αδύνατη την εκτέλεση

της εφαρμογής στο edge, οπότε η εφαρμογή εκτελείται στο cloud όπου οι πόροι είναι απεριόριστοι αλλά η καθυστέρηση μετάδοσης είναι αυξημένη. Για την περαιτέρω βελτιστοποίηση του προβλήματος αναπτύσσεται ένας multi-agent rollout αλγόριθμος που χρησιμοποιεί ενισχυτική μάθηση σε συνδυασμό με τον άπληστο αλγόριθμο. Η συγκεκριμένη έρευνα καταλήγει ότι ο άπληστος αλγόριθμος είναι ο πιο γρήγορος αλλά απέχει περισσότερο από την βέλτιστη λύση, ο MILP αλγόριθμος βρίσκει πάντα την βέλτιστη λύση αλλά ο χρόνος υπολογισμού του αυξάνεται εκθετικά και ο multi-agent rollout αλγόριθμος προσεγγίζει αρκετά την βέλτιστη λύση ενώ υπολογίζεται σε πολυωνυμικό χρόνο. Επίσης, όταν δίνεται προτεραιότητα στον χρόνο απόκρισης χρησιμοποιείται περισσότερο το edge με μεγαλύτερο κόστος, ενώ όταν δίνεται προτεραιότητα στο κόστος χρησιμοποιείται περισσότερο το cloud με μεγαλύτερη καθυστέρηση.

Στην [35] δημιουργείται ένα περιβάλλον εκτέλεσης cloud-native εφαρμογών για αποτελεσματική χρήση 5G δικτύων, υπολογιστικών και αποθηκευτικών πόρων μεταξύ συσκευών και διαφορετικών επίπεδων edge-cloud. Η κάθε εφαρμογή αποτελείται από μικροϋπηρεσίες, όπου η σειρά με την οποία εκτελούνται είναι γνωστή και καθορίζει τον αποδεκτό χρόνο απόκρισης μέχρι την ολοκλήρωση της. Το συγκεκριμένο περιβάλλον εκτέλεσης στοχεύει στην αποδοτική κατανομή των μικροϋπηρεσιών στο edge και στο cloud για την ελαχιστοποίηση του συνολικού κόστους και του χρόνου απόκρισης ικανοποιώντας τους επιθυμητούς χρόνους απόκρισης, διατηρώντας την συνέπεια των δεδομένων μεταξύ κατανεμημένων αποθηκευτικών χώρων και επιλύοντας προσωρινά προβλήματα στο δίκτυο. Το περιβάλλον εκτέλεσης αποτελείται από ένα Policy Engine (PE), που ελέγχει διαρκώς την επίδοση των εφαρμογών και του δικτύου και αποφασίζει τον διαμερισμό των μικροϋπηρεσιών στα διάφορα επίπεδα, και ένα Scheduler (S), που λαμβάνει τον διαμερισμό από το PE, κατανέμει τις μικροϋπηρεσίες και είναι υπεύθυνο για την δυναμική τροποποίηση της κατανομής. Το PE χρησιμοποιεί ένα min-cut αλγόριθμο, για να διαχωρίσει τις μικροϋπηρεσίες που θα εκτελεστούν στο edge και στο cloud με το ελάχιστο δυνατό κόστος χωρίς να παραβιάζονται οι απαιτούμενοι χρόνοι απόκρισης, και δίνει τη δυνατότητα με διαδοχική χρήση του αλγόριθμου να γίνει διαμερισμός σε περισσότερα επίπεδα. Το περιβάλλον εκτέλεσης που προτείνεται επιτυγχάνει την βελτίωση του χρόνου απόκρισης και την μείωση του κόστους συγκριτικά με άλλα συστήματα, ενώ η δυνατότητα δυναμικής προσαρμογής του διαμερισμού διατηρεί τον χρόνο απόκρισης των εφαρμογών σε αποδεκτά επίπεδα αποφεύγοντας προβλήματα συμφόρησης στο δίκτυο μεταξύ edge και cloud.

Η [36] χρησιμοποιεί οριζόντια και κατακόρυφη προώθηση εφαρμογών μεταξύ συσκευών στο ίδιο και σε διαφορετικό επίπεδο αντίστοιχα, για την ελαχιστοποίηση του κόστους. Η υποδομή που εξετάζεται αποτελείται από τέσσερα επίπεδα: τις συσκευές που λαμβάνουν απευθείας δεδομένα και μπορούν

να τα επεξεργαστούν ή να τα προωθήσουν, το edge δίκτυο, τα κεντρικά γραφεία που είναι μικρά κέντρα δεδομένων και τα ενοποιημένα κέντρα δεδομένων. Οι υπολογιστικές δυνατότητες και το κόστος εξυπηρέτησης αυξάνονται σε κάθε επίπεδο με το τελευταίο να αναλαμβάνει όλες τις εφαρμογές που δεν μπορούν να εξυπηρετήσουν τα χαμηλότερα επίπεδα. Ο στόχος του συστήματος είναι η ελαχιστοποίηση του συνολικού κόστους υπολογισμού και επικοινωνίας χωρίς να παραβιάζονται οι περιορισμοί των συνδέσμων επικοινωνίας μεταξύ των επιπέδων και των υπολογιστικών πόρων των συσκευών σε κάθε επίπεδο. Το πρόβλημα απεικονίζεται ως μικτός ακέραιος μη-γραμμικός προγραμματισμός (MINLP) που είναι δύσκολο να επιλυθεί και προτείνεται ένας προσεγγιστικός αλγόριθμος που χρησιμοποιεί branch-and-bound μεθόδους για την εύρεση της λύσης. Η δυνατότητα προώθησης εφαρμογών μεταξύ συσκευών στο ίδιο επίπεδο που προτείνεται μπορεί να οδηγήσει σε μείωση του συνολικού κόστους συγκριτικά με παραδοσιακά συστήματα που υποστηρίζουν μόνο κατακόρυφη προώθηση εφαρμογών, ειδικά σε περιπτώσεις όπου το φόρτο εργασίας δεν είναι ισορροπημένο. Επιπλέον, συμπεραίνεται ότι όσο μειώνεται το κόστος χρήσης των υψηλότερων επιπέδων, μειώνεται και η οριζόντια προώθηση εφαρμογών και ότι όταν όλα τα επίπεδα έχουν το ίδιο κόστος, τότε αυξάνεται η οριζόντια προώθηση εφαρμογών χωρίς να συνεπάγεται βελτίωση του κόστους.

Η [37] ασχολείται με συστήματα Internet of Things που επιτρέπουν εικονικοποίηση δικτυακών λειτουργιών (network function virtualization NFV) (NFV-enabled IoT NIoT) και αποτελούνται από τρία επίπεδα: IoT, edge και cloud. Τα δεδομένα προέρχονται από το IoT επίπεδο και δρομολογούνται στο edge και στο cloud με χρήση multihop δρομολόγησης. Οι συσκευές στο IoT επίπεδο διακρίνονται σε τελικούς κόμβους που έχουν περιορισμένη χωρητικότητα και αναλαμβάνουν απλές δουλειές όπως να συλλέγουν δεδομένα και να επικοινωνούν με τα gateways και σε gateways που λαμβάνουν τα δεδομένα και επικοινωνούν με το edge και το cloud επίπεδο. Σε ένα τέτοιο σύστημα η [37] καλείται να αντιμετωπίσει το πρόβλημα της βέλτιστης τοποθέτησης gateways συσκευών και της δρομολόγησης προς αυτές και να βρει την βέλτιστη τοποθεσία εκτέλεσης των εφαρμογών ώστε να ελαχιστοποιηθεί το υπολογιστικό και το ενεργειακό κόστος, όπου το ενεργειακό κόστος ορίζεται ως το πλήθος των εξυπηρετητών στο edge και στο cloud επίπεδο που χρησιμοποιούνται. Τα προβλήματα αρχικά απεικονίζονται σε MILP μορφή για την εύρεση της βέλτιστης λύσης και αναπτύσσονται προσεγγιστικοί αλγόριθμοι που βασίζονται στο Simulated Annealing (SA) μαζί με συναρτήσεις γειτνίασης για την αποτελεσματικότερη αντιμετώπισή τους. Το SA είναι μια ευριστική προσέγγιση που ψάχνει το συνολικό βέλτιστο ενός προβλήματος που το σύνολο λύσεων του περιλαμβάνει ένα τοπικό βέλτιστο, θεωρώντας ένα χειρότερο σενάριο με ορισμένη πιθανότητα ώστε να ξεφεύγει από τοπικά βέλτιστες λύσεις. Η συγκεκριμένη εργασία συμπεραίνει ότι το κόστος

εγκατάστασης μειώνεται όσο αυξάνεται ο μέγιστος αριθμός από hops και όσο πιο πυκνό είναι το δίκτυο. Επίσης, συμπεραίνει ότι IoT εφαρμογές πρέπει να εκτελούνται στο edge επίπεδο μόνο στην περίπτωση όπου υπάρχουν αυστηροί περιορισμοί καθυστέρησης για να επιτευχθεί μικρότερο κόστος.

Στην [38] εξετάζεται η χρήση αποθήκευσης περιεχομένου σε ένα κεντρικό edge σύστημα, όπου το εύρος συχνοτήτων και οι υπολογιστικοί πόροι συνδυάζονται για να ικανοποιήσουν τις απαιτήσεις εφαρμογών ελαχιστοποιώντας τον χρόνο απόκρισης. Οι συσκευές ενώνονται με έναν macro eNodeB (MeNB) εξοπλισμένο με έναν mobile edge computing (MEC) εξυπηρετητή χρησιμοποιώντας Orthogonal Frequency-Division multiplexing, δηλαδή το συνολικό διαθέσιμο εύρος συχνοτήτων χωρίζεται σε μικρότερα κομμάτια και κάθε συσκευή δεσμεύει ένα μέρος του για να μεταδώσει δεδομένα στον εξυπηρετητή. Οι συσκευές έχουν εφαρμογές όπου μπορούν είτε να τις εκτελέσουν τοπικά ή να τις προωθήσουν στον MEC εξυπηρετητή, ο εξυπηρετητής ζητάει περιεχόμενο από το Ιντερνέτ μέσω backhaul σύνδεσης και έχει την δυνατότητα να το αποθηκεύσει για να χρησιμοποιηθεί ξανά. Ο MEC εξυπηρετητής επιλέγει αν θα αποθηκεύσει το περιεχόμενο ανάλογα με την δημοτικότητα του, όπου θεωρείται ότι το περιεχόμενο ακολουθεί Zipf κατανομή. Ο στόχος του συστήματος είναι να ελαχιστοποιηθεί ο συνολικός χρόνος απόκρισης όλων των εφαρμογών στο δίκτυο βρίσκοντας το εύρος που δεσμεύει κάθε συσκευή, την βέλτιστη τοποθεσία εκτέλεσης των εφαρμογών και της βέλτιστης επιλογής περιεχομένων για αποθήκευση. Το πρόβλημα αναλύεται ως ένα κυρτό πρόβλημα βελτιστοποίησης σε MINLP μορφή και για την λύση του αρχικά αναπτύσσεται μία τροποποιημένη branch-and-bound μέθοδος, όπου ελέγχει ένα ασύμμετρο δέντρο και βρίσκει το βέλτιστο μονοπάτι που οδηγεί στην βέλτιστη λύση. Για την περαιτέρω μείωση της υπολογιστικής πολυπλοκότητας του αλγορίθμου σε πολυωνυμικό χρόνο, χρησιμοποιείται μία generalized benders decomposition μέθοδος, που χωρίζει το πρόβλημα σε ένα master πρόβλημα που έχει MILP μορφή και σε ένα μη γραμμικό primal πρόβλημα. Ο αλγόριθμος μπορεί να προσεγγίσει την βέλτιστη λύση από διαδοχικές λύσεις του master και του primal προβλήματος. Ο αλγόριθμος που προτείνεται παρουσιάζει βελτίωση στον χρόνο απόκρισης συγκριτικά με άλλες λύσεις, όπως να εκτελούνται όλες οι εφαρμογές τοπικά ή όλες στο edge ή το εύρος του εξυπηρετητή να είναι ίσα κατανεμημένο στις συσκευές. Η αποθήκευση περιεχομένου στο edge μειώνει την καθυστέρηση επικοινωνίας με το Ίντερνετ και πετυχαίνει μικρότερο χρόνο απόκρισης, ωστόσο ο αλγόριθμος απαιτεί περισσότερο χρόνο για την εκτέλεση του.

Η [39] προτείνει το Edge-CoCaCo, ένα καινούργιο μοντέλο που στοχεύει στην κοινή βελτιστοποίηση της επικοινωνίας, της αποθήκευσης και του υπολογισμού στο edge-cloud με σκοπό την ελαχιστοποίηση του χρόνου απόκρισης των εφαρμογών. Η κάθε εφαρμογή μπορεί να διαιρεθεί σε μικρότερες

εργασίες και μέρος τους να υπολογιστεί τοπικά, ενώ οι υπόλοιπες στο edge-cloud. Η αποθήκευση υπολογιστικών εργασιών σχετίζεται με την δημοτικότητα, το μέγεθος των περιεχομένων και την απαιτούμενη υπολογιστική χωρητικότητα τους. Δεδομένης της ποικιλομορφίας των εφαρμογών και της δυνατότητες αποθήκευσης στο edge-cloud, υπάρχουν τρεις περιπτώσεις: η εργασία δεν είναι αποθηκευμένη, τότε η συσκευή πρέπει να τη προωθήσει στο edge-cloud, να υπολογιστεί και να λάβει την απάντηση, η εργασία είναι αποθηκευμένη, τότε η συσκευή δεν χρειάζεται να την στείλει στο edge-cloud, αλλά γίνεται ο υπολογισμός της για να λάβει η συσκευή την απάντηση, το αποτέλεσμα της εργασίας είναι αποθηκευμένο, τότε η συσκευή λαμβάνει κατευθείαν την απάντηση. Συγκριτικά με την αποθήκευση περιεχομένου, η αποθήκευση υπολογιστικής εργασίας δεν λαμβάνει υπόψη μόνο την δημοτικότητα της, αλλά και το μέγεθος των δεδομένων και τις υπολογιστικές απαιτήσεις της. Το πρόβλημα απεικονίζεται σε MILP μορφή με στόχο την βέλτιστη αποθήκευση υπολογιστικών εργασιών στο edge-cloud και την ελαχιστοποίηση του χρόνου ικανοποίησης τους. Η βέλτιστη λύση για αυτό το μοντέλο μπορεί να βρεθεί με την χρήση των Karush-Kuhn-Tucker συνθηκών και με έναν branch-and-bound αλγόριθμο. Ο τρόπος αποθήκευσης που προτείνεται στο Edge-CoCaCo μοντέλο υπερτερεί σε σύγκριση με άλλα μοντέλα αποθήκευσης υπολογιστικών εργασιών και μειώνει σημαντικά τον χρόνο ολοκλήρωσης των εφαρμογών.

Στην [40] εξετάζεται το πρόβλημα ελαχιστοποίησης του χρόνου απόκρισης σε ένα ιεραρχικό σύστημα μέσω της μερικής ανάθεσης υπολογισμών στο edge-cloud και της κατάλληλης κατανομής υπολογιστικών και δικτυακών πόρων. Οι συσκευές προωθούν ολόκληρες τις εφαρμογές στον edge εξυπηρετητή που τους αντιστοιχεί μέσω ασύρματης σύνδεσης χρησιμοποιώντας time-division multiple access (TDMA) μέθοδο, όπου μπορεί είτε να τις επεξεργαστεί ολόκληρες είτε να προωθήσει μέρος τους στο κεντρικό cloud μέσω backhaul συνδέσεων. Διακρίνονται δύο προβλήματα: το μέρος του εύρους ζώνης που αντιστοιχεί σε κάθε συσκευή για την μετάδοση των εφαρμογών στο edge, που υπολογίζεται με χρήση της ανισότητας Cauchy-Buniakowsky-Schwarz, και ο τρόπος διαμερισμού των εφαρμογών στο edge-cloud. Ο διαμερισμός των εφαρμογών υπολογίζεται μέσω της κανονικοποιημένης χωρητικότητας της backhaul επικοινωνίας και της κανονικοποιημένης υπολογιστικής δυνατότητας του cloud και εξετάζονται τέσσερα διαφορετικά συστήματα. Πρώτον συστήματα με περιορισμένους πόρους επικοινωνίας, όπου ο διαμερισμός των εφαρμογών εξαρτάται μόνο από την κανονικοποιημένη χωρητικότητα της backhaul επικοινωνίας. Δεύτερον συστήματα με περιορισμένους υπολογιστικούς πόρους, όπου ο διαμερισμός εξαρτάται μόνο από την κανονικοποιημένη υπολογιστική δυνατότητα του cloud. Τρίτον συστήματα με κυρίαρχο edge, όπου οι εφαρμογές πρέπει να εκτελούνται μόνο στο edge. Τέταρτον συστήματα με κυρίαρχο cloud, όπου ο διαμερισμός

εξαρτάται μόνο από την κανονικοποιημένη χωρητικότητα της backhaul επικοινωνίας και ανάλογα αν υπερτερεί η καθυστέρηση επικοινωνίας χρησιμοποιείται περισσότερο το edge αλλιώς χρησιμοποιείται περισσότερο το cloud. Τελικά, στην [40] το πρόβλημα απεικονίζεται ως ένα κυρτό πρόβλημα βελτιστοποίησης που λύνεται με χρήση των Karush-Kuhn-Tucker συνθηκών.

Στην [41] προτείνεται το IoT-based Service Components Optimization Model (IoTSCOM) για την ανάθεση IoT data-intensive εφαρμογών σε ένα υβριδικό edge-cloud σύστημα. Ο βασικός στόχος του IoT-SCOM βασίζεται στο χρόνο απόκρισης με τη μικρότερη δυνατή καθυστέρηση και το λιγότερο απαιτούμενο ενεργειακό κόστος από τις IoT συσκευές για την μετάδοση των εφαρμογών στο edge και στο cloud. Το σύστημα που εξετάζεται αποτελείται από το IoT επίπεδο, που μπορεί να επεξεργαστεί εφαρμογές ή να τις προωθήσει ανάλογα με τις απαιτήσεις τους και τη χωρητικότητα του δικτύου, το edge επίπεδο, που προσφέρει γρήγορο χρόνο απόκρισης, και το cloud επίπεδο που προσφέρει άφθονους υπολογιστικούς πόρους. Το IoT-SCOM χρησιμοποιεί simulating annealing μέθοδο για την δημιουργία ενός αλγορίθμου βελτιστοποίησης αποικίας μυρμηγκιών, όπου οι διαδρομές των μυρμηγκιών αντιστοιχούν στην κατανομή των εφαρμογών στα επίπεδα και η διάρκεια της διαδρομής αντιστοιχεί στην συνολική καθυστέρηση. Για την μείωση της πολυπλοκότητας του αλγορίθμου προστίθεται τεχνική multi-agent βελτιστοποίησης, όπου ο πράκτορας εκτέλεσης είναι υπεύθυνος για την εκτέλεση της διαδικασίας βελτιστοποίησης, ο πράκτορας συγχρονισμού διαιρεί τον πληθυσμό των μυρμηγκιών, τον μοιράζει στους πράκτορες εκτέλεσης και σηματοδοτεί τους πράκτορες ελέγχου, και ο πράκτορας ελέγχου εξετάζει τις λύσεις των πρακτόρων εκτέλεσης και προσαρμόζει τον πληθυσμό ανάλογα με τα αποτελέσματα των λύσεων.

Η [42] προτείνει μια πολιτική προώθησης εργασιών σε IoT-fog-cloud συστήματα για την μείωση του χρόνου εξυπηρέτησης. Στο εξεταζόμενο σύστημα οι εργασίες εκτελούνται τοπικά ή προωθούνται σε ανώτερο επίπεδο και δίνεται η δυνατότητα οριζόντιας προώθησης εργασιών μεταξύ υποδομών στο fog επίπεδο. Οι fog κόμβοι που ανήκουν στο ίδιο σύμπλεγμα αλληλοεπιδρούν και μπορούν να προωθήσουν εργασίες μεταξύ τους. Συγκεκριμένα, ένας fog κόμβος διατηρεί ουρά με multiclass traffic για τις εργασίες που έχει να εκτελέσει και υπολογίζει τον αναμενόμενο χρόνο εξυπηρέτησης της ουράς του. Επίσης, οι fog κόμβοι διατηρούν έναν πίνακα με τους αναμενόμενους χρόνους εξυπηρέτησης των γειτονικών κόμβων. Όταν μία εργασία φτάνει σε έναν fog κόμβο, αυτός ελέγχει την αναμενόμενο χρόνο εξυπηρέτησής του στην ουρά του και αποφασίζει αν μπορεί να το δεχτεί, αλλιώς το προωθεί στον γείτονα με τον μικρότερο αναμενόμενο χρόνο εξυπηρέτησης ή στο cloud αν έχει προωθηθεί πολλές φορές. Για τον υπολογισμό του αναμενόμενου χρόνου εξυπηρέτησης χωρίζει τις εργασίες σε ελαφριές και βαριές υπολογιστικά εργασίες με γνωστό μέσο χρόνο εξυπηρέτησης.

Στην [42] παρουσιάζονται οι σχέσεις για τον υπολογισμό της καθυστέρησης υπολογισμού και μετάδοσης των εργασιών που οδηγούν στο ελάχιστο συνολικό χρόνο καθυστέρησης και για την πιθανότητα αποδοχής εργασιών σε fog κόμβους και πιθανής προώθησης στο cloud. Η στρατηγική προώθησης στο fog επίπεδο που προτείνεται οδηγεί σε μείωση του χρόνου εξυπηρέτησης και καταλήγει στο συμπέρασμα ότι αν μία εργασία δεν μπορεί να προωθηθεί τότε αν είναι υπολογιστικά βαριά προτιμάτε το cloud, ενώ αν είναι ελαφριά παραμένει στο fog.

Στην [43] εξετάζεται το πρόβλημα κατανομής εργασιών με το ελάχιστο κόστος και την λιγότερη κατανάλωση ενέργειας για ένα διαχειριστή δικτύου σε ένα ιεραρχικό multi-tier MEC σύστημα. Εξετάζεται ένα MEC σύστημα με δύο επίπεδα που αποτελείται από εξυπηρετητές του διαχειριστή δικτύου στο edge επίπεδο και από ένα δυνατό cloud εξυπηρετητή που νοικιάζει. Οι συσκευές διαθέτουν ένα μέρος συχνοτήτων από το συνολικό διαθέσιμο εύρος του δικτύου για την επικοινωνία με το edge και καταναλώνουν ενέργεια ανάλογη με το μέρος συχνοτήτων που τους αντιστοιχεί για την προώθηση εργασιών σε έναν edge εξυπηρετητή. Ο διαχειριστής δικτύου νοικιάζει τον κεντρικό cloud εξυπηρετητή και κατά συνέπεια το κόστος χρήσης του είναι μεγαλύτερο από ότι στους edge εξυπηρετητές. Το πρόβλημα υπολογισμού του ελάχιστου βεβαρημένου αθροίσματος της κατανάλωσης ενέργειας των συσκευών και του κόστους για τον διαχειριστή δικτύου είναι μικτό ακέραιο μη-κυρτό πρόβλημα και με χρήση της big-M μεθόδου μετατρέπεται σε μικτό ακέραιο γενικευμένο κυρτό πρόβλημα που μπορεί να λυθεί με αρκετή ακρίβεια χρησιμοποιώντας mixed-integer second order cone programming (MISOCP). Αρχικά, υλοποιείται μία branch-and-bound μέθοδο, όπου ψάχνει εξαντλητικά όλες τις πιθανές λύσεις του προβλήματος μέχρι η διαφορά μεταξύ του άνω και του κάτω ορίου να γίνει αμελητέα. Στη συνέχεια, αναπτύσσεται ένας SCA (Successive Convex Approximation)-Based MISOCP αλγόριθμος, όπου χρησιμοποιεί μία SCA-based μέθοδο για να προσεγγίσει τη λύση μετατρέποντας τους περιορισμούς σε κωνική μορφή και υπολογίζοντας μία σειρά από προσεγγιστικά MISOCP που συγκλίνουν στην βέλτιστη λύση. Τέλος, για την περαιτέρω μείωση της πολυπλοκότητας χρησιμοποιείται συνεχής χαλάρωση των μεταβλητών, ώστε το πρόβλημα να γίνει SOCP που λύνεται σε πολυωνυμικό χρόνο με μικρή απόκλιση από την βέλτιστη λύση.

Η [44] ασχολείται με την αποδοτική και ασφαλή κατανομή εργασιών σε mobile edge cloud συστήματα και στην κατανομή φορτίου μεταξύ σταθμών στο edge επίπεδο. Στο σύστημα που εξετάζεται, οι εργασίες μπορούν να εκτελεστούν τοπικά ή να προωθηθούν σε κάποιο άλλο επίπεδο, η συσκευή καταναλώνει ενέργεια είτε για τον υπολογισμό της εργασίας είτε για την προώθηση της στο edge ή στο cloud. Η κατανομή των χρηστών στους edge σταθμούς δεν είναι ισορροπημένη και κάποιοι είναι υπερφορτωμένοι, για αυτό χρειάζεται ανακατανομή του φορτίου μεταξύ edge σταθμών. Για την ανακατανομή του

φορτίου, κάθε σταθμός στέλνει σε έναν κεντρικό έλεγχο μία περίληψη από τους χρήστες του και τους χρήστες που μπορούν να αναδιανεμηθούν σε άλλους κοντινούς σταθμούς. Στη συνέχεια, ο διαχειριστής του κεντρικού ελέγχου εξετάζει έναν χρήστη και τον αναγκάζει να παραδοθεί στον καλύτερο διαθέσιμο edge σταθμό και ανανεώνει τον ρυθμό μετάδοσης δεδομένων και την υπολογιστική δύναμη που έχει αποδοθεί σε κάθε χρήστη. Η διαδικασία επαναλαμβάνεται για όλους τους χρήστες μέχρι να εξυπηρετούνται όλοι από τον βέλτιστο edge σταθμό. Η ισορροπία φορτίου των edge σταθμών μπορεί να επιτευχθεί και με τη χρήση της software-defined network (SDN) τεχνολογίας χειρισμού και του τυποποιημένου OpenFlow πρωτοκόλλου για πιο αποδοτικές και ακριβείς αποφάσεις. Επίσης, προτείνεται ένα νέο επίπεδο ασφαλείας για την ασφαλή μεταφορά δεδομένων προς το edge cloud που ενσωματώνει advanced encryption standard (AES) κρυπτογραφικές τεχνικές μαζί με electrocardiogram (ECG) σήματα που δημιουργούν ένα βιοηλεκτρικό σήμα για την δημιουργία μοναδικών κλειδιών κρυπτογράφησης και αποκρυπτογράφησης. Το τελικό πρόβλημα ασφαλούς προώθησης εργασιών και ισορροπίας φορτίου απεικονίζεται ως πρόβλημα βελτιστοποίησης με γραμμικούς περιορισμούς που μπορεί να λυθεί με χρήση branch-and-bound μεθόδου.

Η [45] στοχεύει στην ελαχιστοποίηση του χρόνου απόκρισης για cyber-physical systems CPS με edge-cloud computing λαμβάνοντας υπόψη τον ενεργειακό προϋπολογισμό και τις απαιτήσεις αξιοπιστίας των CPS εφαρμογών. Στο σύστημα που εξετάζεται εφαρμογές φτάνουν σε βασικούς σταθμούς από τους χρήστες και προωθούνται προς τους ετερογενείς edge εξυπηρετητές ή τον cloud εξυπηρετητή. Ο χρόνος απόκρισης και η ενέργεια που καταναλώνεται για μία εργασία υπολογίζονται από το αντίστοιχο άθροισμα για την μετάδοση της προς έναν εξυπηρετητή και τον υπολογισμό της. Επίσης, ορίζεται η αξιοπιστία των εργασιών σε ένα βασικό σταθμό ως η πιθανότητα να μεταδοθεί επιτυχώς η εργασία στον κατάλληλο εξυπηρετητή και να εκτελεστεί επιτυχώς. Για να επιτευχθούν οι απαιτήσεις αξιοπιστίας του συστήματος, χρησιμοποιείται μια backup τεχνική και εκτελείται ένα τεστ αποδοχής μετά την εκτέλεση του backup σε κάποιον εξυπηρετητή, αν το test δείξει ότι δεν υπάρχουν λάθη τότε γίνεται δεκτό το αποτέλεσμα, αλλιώς τα αποτελέσματα πετιούνται. Για να βρεθεί η βέλτιστη δρομολόγηση εργασιών στους εξυπηρετητές και ο κατάλληλος αριθμός από backup εργασιών για κάθε βασικό σταθμό υλοποιείται μία προσέγγιση με δύο στάδια: τη στατική και τη δυναμική βελτιστοποίηση του χρόνου εξυπηρέτησης. Στο στατικό στάδιο βελτιστοποίησης, χρησιμοποιείται Monte-Carlo προσομοίωση με τεχνική ακέραιου γραμμικού προγραμματισμού (ILP), όπου ορίζεται ένας συντελεστής προσαρμογής λάθους για τον υπολογισμό του αριθμού των backup εργασιών σε κάθε σταθμό, στη συνέχεια βρίσκεται η βέλτιστη δρομολόγηση για δεδομένο συντελεστή με την ILP τεχνική και υπολογίζονται τα bits και soft λάθη για

τον υπολογισμό της αξιοπιστίας του συστήματος από όπου φτιάχνεται ένα δείγμα της Monte-Carlo προσομοίωσης. Με επανάληψη αυτής της διαδικασίας φτιάχνονται αρκετά δείγματα της προσομοίωσης και υπολογίζεται η αξιοπιστία του συστήματος, αν είναι ικανοποιητική ο αλγόριθμος σταματάει, αλλιώς προσαρμόζεται ο συντελεστής προσαρμογής λάθους και ξαναδοκιμάζεται η προσομοίωση. Στο δυναμικό στάδιο, χρησιμοποιείται ένας backup-adaptive δυναμικός μηχανισμός βελτιστοποίησης, όπου όταν βρεθεί ένα επιτυχημένο backup, οι μεταδόσεις και εκτελέσεις άλλων περιττών backup που συμβαίνουν στο στατικό στάδιο ακυρώνονται.

Στην [46] προτείνονται offline και online αλγόριθμοι για την ελαχιστοποίηση του χρόνου εξυπηρέτησης εργασιών σε mobile edge cloud συστήματα μέσω βελτιστοποίησης του διαμοιρασμού τους τοπικά και στο edge cloud και της διαχείρισης του δικτύου για την επικοινωνία μεταξύ χρηστών και edge cloud εξυπηρετητών. Το σύστημα αφορά εφαρμογές που χωρίζονται σε διαδοχικές εργασίες με την πρώτη και την τελευταία να περιγράφουν τα δεδομένα εισόδου και εξόδου αντίστοιχα και να εκτελούνται τοπικά. Οι συσκευές των χρηστών θεωρούνται ετερογενείς, ενώ όλοι οι εξυπηρετητές στο edge cloud έχουν τις ίδιες υπολογιστικές δυνατότητες. Οι χρήστες επικοινωνούν με το edge cloud μέσω κοινών ασύρματων access points ή κινητών σταθμών βάσης. Αρχικά, αναπτύσσεται η Multi-Dimensional Search and Adjust (MDSA) ευριστική, που είναι ένας offline αλγόριθμος, ο οποίος χωρίζει το συνολικό εύρος ζώνης του δικτύου σε μικρότερα ίσα κομμάτια, το καθένα από τα οποία μπορεί να χρησιμοποιείται από μία εφαρμογή σε μία χρονική στιγμή, και βρίσκει ένα αρχικό διαμοιρασμό χωρίς να εξετάζει τους περιορισμούς του συστήματος. Στη συνέχεια, βρίσκει την πρώτη χρονική στιγμή που παραβιάζεται ο αριθμός των εξυπηρετητών ή το συνολικό εύρος ζώνης του δικτύου και προσαρμόζει την λύση καθυστερώντας την εργασία μέχρι επόμενη χρονική στιγμή ή προωθώντας την για εκτέλεση στην συσκευή ώστε να μην παραβιάζεται κάποιος περιορισμός σε αυτήν την στιγμή ούτε σε κάποια προηγούμενη. Μετά, συνεχίζει στην επομένη χρονική στιγμή που παραβιάζεται κάποιος περιορισμός μέχρι να τηρούνται οι περιορισμοί για κάθε χρονική στιγμή και υπολογίζεται το σύνολο των χρονικών στιγμών που απαιτείται για την ολοκλήρωση όλων των εφαρμογών. Επίσης, αναπτύσσεται ο Cooperative Online Scheduling αλγόριθμος που μπορεί να αξιοποιηθεί σε πρακτικά συστήματα. Ο COS είναι μία task-oriented μέθοδος προγραμματισμού με δύο schedulers: τον edge cloud scheduler και τον network scheduler για την ανάθεση υπολογιστικών πόρων στις εργασίες και πόρων δικτύου στις μεταδόσεις δεδομένων αντίστοιχα. Οι δύο schedulers δρουν συνεργατικά ισορροπώντας τους πόρους του συστήματος για την αποφυγή μεγάλων χρόνων αναμονής είτε στους εξυπηρετητές είτε στο δίκτυο. Ο edge cloud scheduler ελέγχει συχνά τον αριθμό των εργασιών που έχουν να εκτελεστούν στο edge cloud και μεταφέρει

εργασίες για εκτέλεση πίσω στην συσκευή αν το φόρτο εργασίας είναι μεγάλο και ο network scheduler ελέγχει την κατάσταση του δικτύου και ακυρώνει την μεταφορά μίας εργασίας τοποθετώντας την όπου ήταν αρχικά προγραμματισμένη να υπολογιστεί.

Η [47] εξετάζει την κατανομή υπολογιστικών πόρων σε κατανεμημένες εφαρμογές μηχανικής μάθησης λαμβάνοντας υπόψη την υπολογιστική απόδοση, το εύρος ζώνης και το κόστος επεξεργασίας των edge cloud υποδομών. Οι συσκευές παράγουν συνέχεια δεδομένα και τα μεταδίδουν στους πόρους δικτύου που τρέχουν τους αλγόριθμους μηχανικής μάθησης σε batches από δεδομένα. Η εκπαίδευση μιας ML εφαρμογής χωρίζεται σε κατανεμημένες ML εργασίες. Κατά τη διάρκεια μίας χρονικής περιόδου, κάθε εξυπηρετητής εκπαιδεύει κάνοντας επεξεργασία ενός batch δεδομένων, που έλαβε στην προηγούμενη χρονική περίοδο, και λαμβάνει δεδομένα που θα χρησιμοποιηθούν στην επόμενη χρονική περίοδο. Το edge δίκτυο έχει περιορισμένους πόρους που μπορούν να χρησιμοποιηθούν για εργασίες μηχανικής μάθησης, ενώ το cloud δίκτυο έχει απεριόριστους και τα δύο δίκτυα έχουν διαφορετικό υπολογιστικό και δικτυακό οικονομικό κόστος. Η [47] χρησιμοποιεί ακέραιο γραμμικό προγραμματισμό (ILP) για τον υπολογισμό της βέλτιστης κατανομής των πόρων του συστήματος με στόχο την ελαχιστοποίηση του υπολογιστικού και του δικτυακού κόστους χωρίς να παραβιάζονται οι περιορισμοί του συστήματος. Τέλος, συμπεραίνει ότι το υπολογιστικό κόστος παίζει σοβαρό ρόλο για την κατανομή των πόρων και ότι η κοινή χρήση edge και cloud καταλήγει σε καλύτερο κόστος από ότι η χρήση μόνο του edge ή μόνο του cloud.

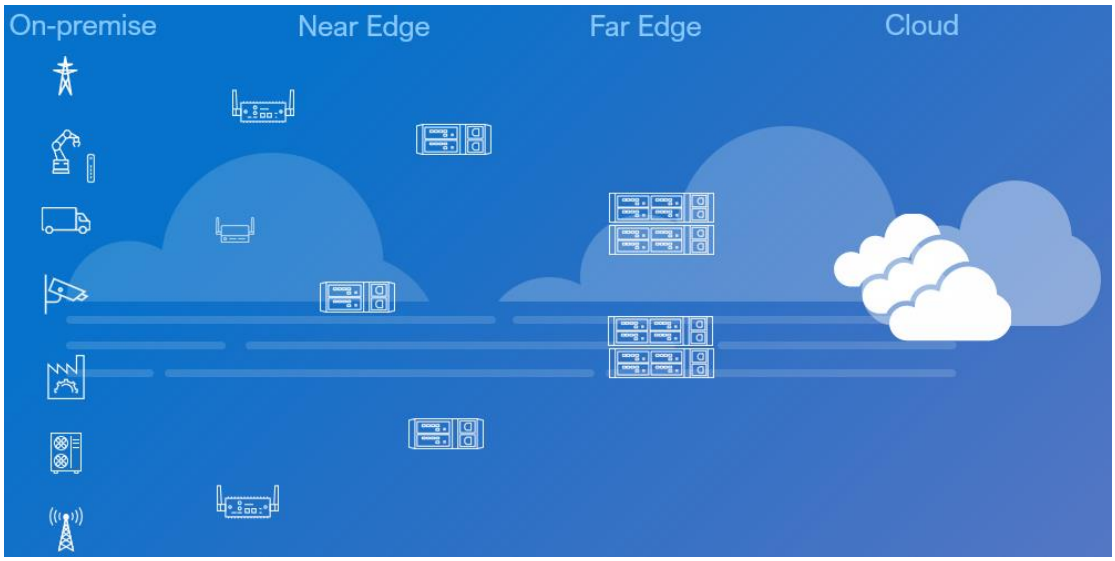
# 4 Το πρόβλημα

## 4.1 Περιγραφή του προβλήματος

### 4.1.1 Υποδομή

Η τεχνολογία του 5G εξελίσσεται και εδραιώνεται, δημιουργώντας πρόσφορο έδαφος για την ανάπτυξη περισσότερων IoT εφαρμογών και άλλων καινοτόμων εφαρμογών για τους χρήστες. Οι εφαρμογές, όπως οι έξυπνες οικιακές συσκευές, τα αυτόνομα οχήματα, η επαυξημένη ή εικονική πραγματικότητα κ.ά., παράγουν έναν γιγαντιαίο όγκο πληροφορίας από αισθητήρες, κάμερες και άλλες τελικές συσκευές. Για την υποστήριξη αυτών των εφαρμογών και την επεξεργασία του τεράστιου όγκου δεδομένων, δεν επαρκεί η κεντρική υποδομή του cloud, για αυτό επιστρατεύεται η αρχιτεκτονική του edge για την ενίσχυση της υποδομής.

Απαιτείται, δηλαδή, η δημιουργία μιας κατανεμημένης και ιεραρχικής τοπολογίας (Εικόνα 4-1) που περιλαμβάνει τοπικούς επεξεργαστές στις IoT συσκευές (on-premise), υπολογιστικούς πόρους είτε σε ίδιες ή κοντινές τοποθεσίες με τις συσκευές (near edge) είτε σε μικρά κέντρα δεδομένων που βρίσκονται κοντά στους κόμβους του αστικού δικτύου (far edge), και μεγάλα κέντρα δεδομένων σε απομακρυσμένες περιοχές (cloud).




*Εικόνα 4-1 Μία Edge Cloud υποδομή*

Συγκεκριμένα, η επεξεργασία μπορεί να γίνει είτε από τις ίδιες τις συσκευές είτε από μικρές υπολογιστικές μονάδες με σχετικά αδύναμους επεξεργαστές, όπως Raspberry Pi, NVIDIA Jetson, Arduino, laptops, desktops, προσφέροντας ταχύτερη απόκριση λόγω της εγγύτητας των πόρων στις τελικές συσκευές. Το near edge επίπεδο βρίσκεται εντός του αστικού δικτύου και περιλαμβάνει πάλι πόρους σχετικά χαμηλών δυνατοτήτων και εξοπλισμό για την προώθηση των εργασιών

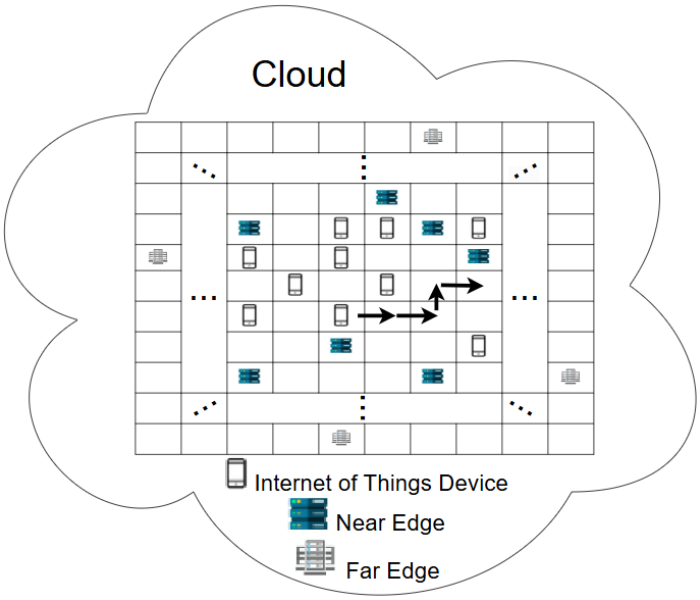
στα ιεραρχικά ανώτερα στρώματα. Ανάμεσα στο αστικό δίκτυο και στα απομακρυσμένα κέντρα δεδομένων του cloud βρίσκεται το far edge επίπεδο, το οποίο αποτελείται από επεξεργαστές αυξημένης χωρητικότητας σε μικρά κέντρα δεδομένων εντός της ευρύτερης μητροπολιτικής περιοχής. Τέλος, το cloud επίπεδο περιλαμβάνει μεγάλα κέντρα δεδομένων σε αποστάσεις χιλιάδων χιλιομέτρων από τις τελικές συσκευές με εικονικά απεριόριστους υπολογιστικούς και αποθηκευτικούς πόρους. Γενικά, σε υψηλότερα επίπεδα από την τελική συσκευή προς το cloud, η διαθεσιμότητα, το κόστος και η αξιοπιστία των πόρων βελτιώνονται. Ωστόσο, αυξάνονται η καθυστέρηση και το τηλεπικοινωνιακό κόστος, καθώς τα δεδομένα μεταφέρονται σε μεγαλύτερες αποστάσεις. Ενδεικτικά χαρακτηριστικά μιας τέτοιας υποδομής φαίνονται στον Πίνακας 4-1.

| | Πλήθος | Μέση Απόσταση | Μέση Καθυστέρηση |
|---|---|---|---|
| IoT Device | >1.000.000 | | |
| Near Edge | 100 | 1-100km | 2-5ms |
| Far Edge | 10 | 100-1.000km | 10-20ms |
| Cloud | <10 | >1.000km | >20ms |

*Πίνακας 4-1 Χαρακτηριστικά Υποδομής*

### 4.1.2 Εφαρμογές

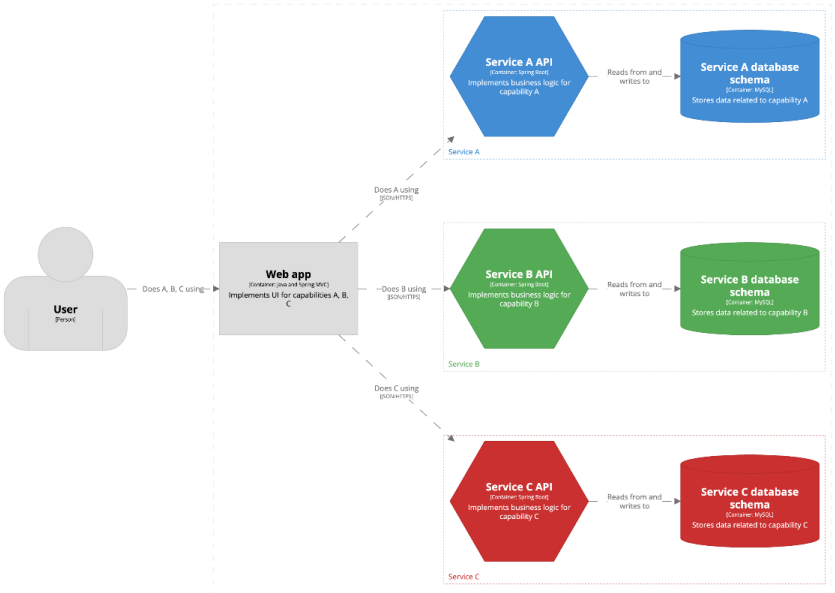
Με την εξέλιξη των cloud native εφαρμογών έχει αυξηθεί η ανάγκη για αδιάκοπη και γρήγορη παροχή υπηρεσιών. Συσκευές, όπως αυτόνομα αυτοκίνητα και drones, μπορούν να κάνουν μεγάλες αποστάσεις χωρίς να σταματάει η παραγωγή δεδομένων και η ανάγκη για υπολογιστικούς πόρους, με αποτέλεσμα να χρειάζεται να αλλάξουν οι edge τοποθεσίες που χρησιμοποιούν για να εξυπηρετηθούν οι απαιτήσεις τους.




*Εικόνα 4-2 Απεικόνιση του χώρου κίνησης των χρηστών*

Έτσι, οι τελικές συσκευές που εξετάζονται μπορούν να κινούνται αυτόνομα στον χώρο προκαλώντας αλλαγή στην κατανομή τον πόρων καθώς απομακρύνονται από ορισμένους κόμβους της υποδομής και πλησιάζουν άλλους (Εικόνα 4-2). Οι συσκευές δεσμεύουν πόρους ανάλογα με την τοποθεσία τους κάθε στιγμή και τους περιορισμούς της υποδομής για την εξυπηρέτηση της εφαρμογής τους.

Οι συσκευές στέλνουν διαρκώς δεδομένα στην υποδομή για τον υπολογισμό των εφαρμογών τους μέχρι να ολοκληρωθούν. Οι εφαρμογές αποτελούνται από μικροϋπηρεσίες που υπολογίζονται παράλληλα και έχουν αλληλεξαρτήσεις μεταξύ τους (Εικόνα 4-3). Η κάθε μικροϋπηρεσία έχει διαφορετικά δεδομένα εισόδου και αποθηκευτικές και υπολογιστικές απαιτήσεις για την εξυπηρέτηση της. Μεταξύ δύο μικροϋπηρεσιών της εφαρμογής ορίζεται η μέγιστη επιτρεπτή καθυστέρηση για την επικοινωνία τους που ικανοποιεί το ζητούμενο επίπεδο ποιότητας υπηρεσιών του χρήστη, έτσι οι μικροϋπηρεσίες με μικρή επιτρεπτή καθυστέρηση πρέπει να εκτελεστούν σε πιο κοντινούς κόμβους του ίδιου επιπέδου της υποδομής. Για την κάθε μικροϋπηρεσία δεσμεύονται υπολογιστικοί και αποθηκευτικοί πόροι από τους κατάλληλους κόμβους τηρώντας τους περιορισμούς της εφαρμογής.




*Εικόνα 4-3 Παράδειγμα εφαρμογής*

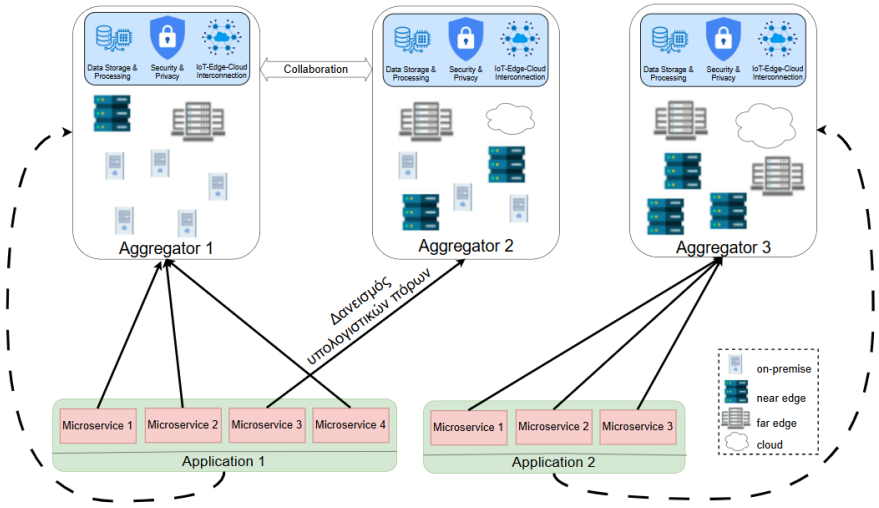
### 4.1.3 Aggregators

Ο μεγάλος αριθμός των ανεξάρτητων και ετερόκλητων μικροϋπηρεσίων που καλείται να εξυπηρετήσει η edge cloud υποδομή οδηγεί στην ανάγκη για την ύπαρξη ενορχηστρωτών για την κατανομή του φόρτου εργασίας. Έτσι, οι κόμβοι της υποδομής οργανώνονται σε πλατφόρμες, όπου δρουν διαφορετικοί aggregators ο καθένας υπεύθυνος για ένα υποσύνολο της υποδομής (Εικόνα 4-4).

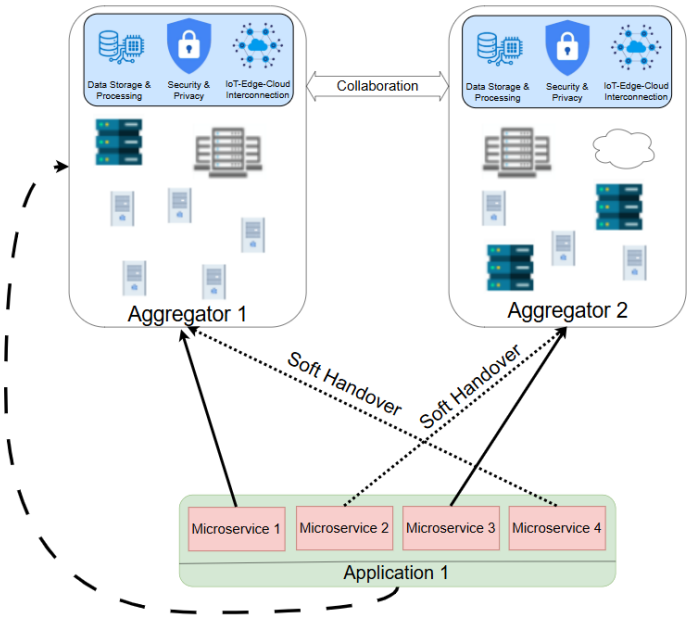
Η κάθε συσκευή αντιστοιχείται σε έναν aggregator, ο οποίος είναι υπεύθυνος να δρομολογήσει τις μικροϋπηρεσίες της εφαρμογής στους κόμβους που ελέγχει.

Οι aggregators έχουν την δυνατότητα συνεργασίας μεταξύ τους, δηλαδή έχουν τη δυνατότητα να δανείζονται πόρους από άλλους aggregators για την εξυπηρέτηση των εφαρμογών που έχουν αναλάβει. Έτσι, ένας aggregator αναλαμβάνει την εξυπηρέτηση μίας εφαρμογής και αν δεν επαρκούν οι δικοί του υπολογιστικοί πόροι μπορεί να προωθήσει ένας μέρος των μικροϋπηρεσιών της εφαρμογής σε κόμβους άλλων aggregators που συνεργάζονται με αυτόν. Επιπλέον, κατά τη διάρκεια της κίνησης του χρήστη αλλάζουν οι κόμβοι της υποδομής που χρησιμοποιεί με αποτέλεσμα να χρειάζεται να τροποποιηθεί η αρχική δρομολόγηση της εφαρμογής. Για την περίπτωση αυτή διακρίνονται δύο περιπτώσεις: αρχικά το soft handover, όπου δεν χρειάζεται ολόκληρη μεταφορά της εφαρμογής και ανακατανέμονται ορισμένες μικροϋπηρεσίες σε κόμβους του ίδιου aggregator ή συνεργατών του, και το hard handover, όπου δεν αρκούν οι πόροι του προηγούμενοι aggregator και προτιμάται η εκ νέου δρομολόγηση ολόκληρης της εφαρμογής, έτσι επιλέγεται καινούργιος aggregator που δεν συνεργάζεται με τον προηγούμενο και απαιτείται ολόκληρη η μεταφορά της κατάστασης της εφαρμογής σε κόμβους του καινούργιου aggregator με αποτέλεσμα να εισάγεται μια επιπλέον καθυστέρηση.

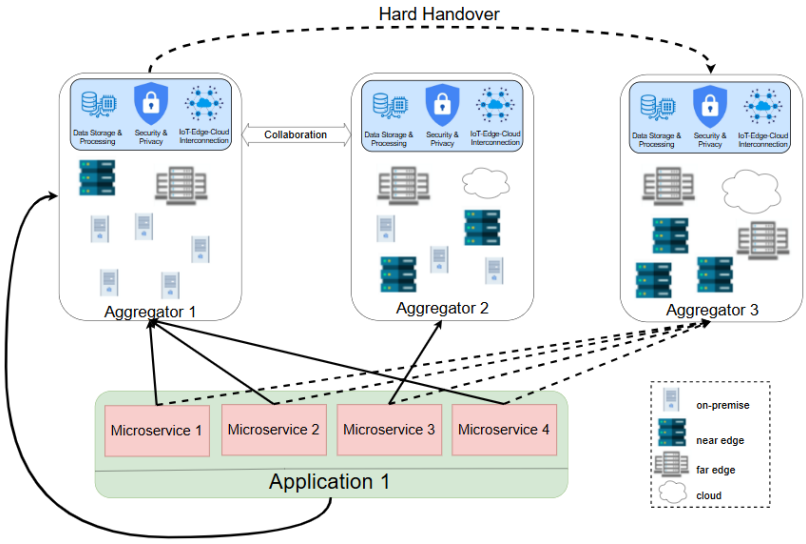
Στην Εικόνα 4-4 φαίνεται ότι οι aggregators 1 και 2 συνεργάζονται, ενώ ο aggregator 3 δεν συνεργάζεται με κάποιον άλλον aggregator. Στην Εικόνα 4-4α, ο aggregator 1 έχει αναλάβει την εφαρμογή 1 και εκτελεί τις μικροϋπηρεσίες 1,2,4 στους κόμβους του και δανείζεται πόρους από τον aggregator 2 για την μικροϋπηρεσία 3, την εφαρμογή 2 την έχει αναλάβει ο aggregator 3 που δεν συνεργάζεται με κάποιον άλλον και εκτελεί όλες τις μικροϋπηρεσίες ο ίδιος. Στην Εικόνα 4-4β φαίνεται ότι την εφαρμογή 1 την έχει αναλάβει ο aggregator 1 και έχει δρομολογήσει τις μικροϋπηρεσίες της αρχικά όπως και στην παραπάνω εικόνα, ωστόσο αναγκάζεται να μεταφέρει την μικροϋπηρεσία 4 σε νέο δικό του κόμβο και την μικροϋπηρεσία 2 σε κόμβο του aggregator 2 που συνεργάζεται με αυτόν. Στην Εικόνα 4-4γ η εφαρμογή 1 μεταφέρεται στον aggregator 3 και δρομολογούνται εκ νέου όλες οι μικροϋπηρεσίες.




(α) Δανεισμός πόρων μεταξύ Aggregators που συνεργάζονται




(β) Soft Handover: Ανακατανομή μικροϋπηρεσιών εντός της συνεργασίας Aggregators



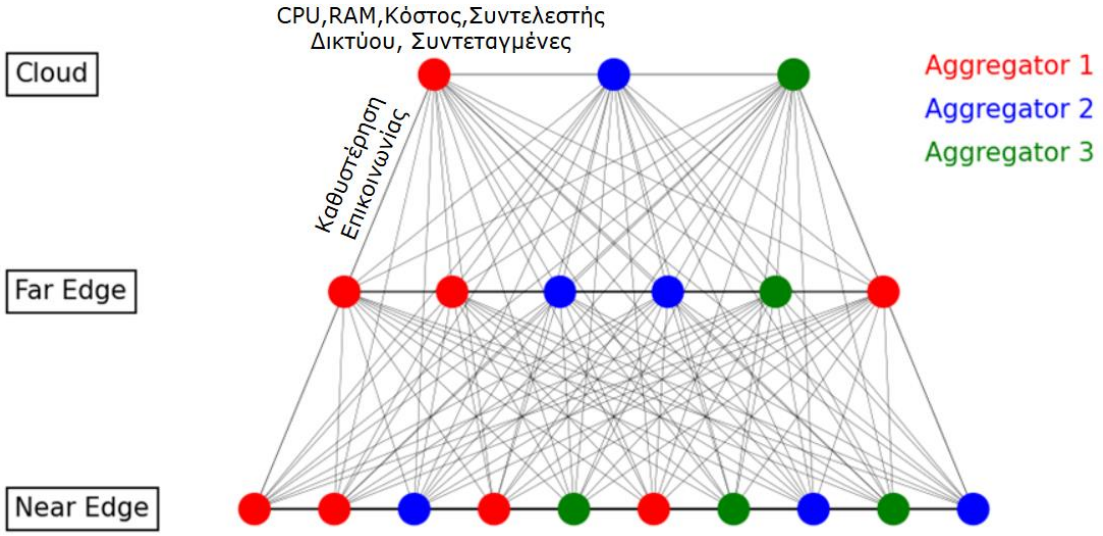

(γ) Hard Handover: Ανακατανομή Εφαρμογής σε νέο Aggregator

*Εικόνα 4-4 Handovers μεταξύ Aggregators*

## 4.2 Μαθηματική Μοντελοποίηση

Για την απεικόνιση της υποδομής χρησιμοποιείται ένας πλήρως συνδεδεμένος γράφος G(V, E) (Εικόνα 4-5) με κόμβους τις near edge-far edge-cloud τοποθεσίες όπου υπάρχουν υπολογιστικά συστήματα και ακμές τις τηλεπικοινωνιακές συνδέσεις του δικτύου μεταξύ των κόμβων. Ο κάθε κόμβος $v \in V$ περιγράφεται από την πλειάδα $(c_v, r_v, o_v, n_v, x_v, y_v)$ , όπου με $c_v$ συμβολίζεται η επεξεργαστική ισχύς CPU του κόμβου, $r_v$ είναι η διαθέσιμη μνήμη RAM, $o_v$ είναι το κόστος χρήσης του, $n_v$ είναι ο συντελεστής δικτύου, που δείχνει το κόστος δικτύου για την μεταφορά δεδομένων από τον τελικό χρήστη προς αυτόν τον κόμβο, και $x_v$, $y_v$ είναι οι συντεταγμένες του κόμβου στον χώρο. Η τιμή $l_{v,v'}$ της ακμής που ενώνει τους κόμβους $v,v' \in V$ αντιπροσωπεύει την καθυστέρηση για την μεταφορά των δεδομένων ανάμεσα στους κόμβους που είναι ανάλογη με την απόσταση μεταξύ τους.

Οι κόμβοι του γράφου χωρίζονται σε υποσύνολα που αντιπροσωπεύουν τους N aggregators της υποδομής. Με $N_n$ συμβολίζεται το σύνολο των κόμβων $v \in V$ που ανήκουν στον aggregator $n \in N$. Το σύνολο των συνεργασιών μεταξύ των aggregators συμβολίζεται με C, με $Z_c$ συμβολίζεται το σύνολο των aggregators $n \in N$ που ανήκουν στην συνεργασία $c \in C$ και με $V_c$ συμβολίζεται το σύνολο των κόμβων $v \in V$ που ανήκουν στην συνεργασία $c \in C$.




*Εικόνα 4-5 Αναπαράσταση γράφου της υποδομής*

Το υπολογιστικό φορτίο του συστήματος περιγράφεται από ένα σύνολο cloud-native εφαρμογών A. Η τοποθεσία του χρήστη που έχει την εφαρμογή $a \in A$ κάθε χρονική στιγμή t συμβολίζεται με τις συντεταγμένες $\chi_{a,t}$, $\psi_{a,t}$. Η εφαρμογή $a \in A$ ξεκινάει την χρονική στιγμή $st_a$ και διαρκεί μέχρι την χρονική στιγμή $ft_a$. Επιπλέον, η εφαρμογή $a \in A$ έχει ένα αποδεκτό όριο καθυστέρησης $\Lambda_a$, το οποίο αντιπροσωπεύει τη συνολική καθυστέρηση που επιδέχεται η εφαρμογή για την

εξυπηρέτηση της. Οι εφαρμογές μπορούν να θεωρηθούν ως μια αλυσίδα από μικροϋπηρεσίες. Η απεικόνιση της εφαρμογής $a \in A$ γίνεται ως μια μη κατευθυνόμενη αλυσίδα $G_a(V_a, E_a)$ από μικροϋπηρεσίες, όπως φαίνεται στην Εικόνα 4-6. Η εφαρμογή $a \in A$ αποτελείται από μικροϋπηρεσίες $I_a \equiv V_a$, η μικροϋπηρεσία $i \in I_a$ περιγράφεται από την πλειάδα $(\varepsilon_{a,i}, \rho_{a,i}, s_{a,i})$, όπου με $\varepsilon_{a,i}$ συμβολίζεται η απαίτηση σε CPU της μικροϋπηρεσίας, $\rho_{a,i}$ είναι η απαιτούμενη μνήμη RAM, $s_{a,i}$ είναι τα δεδομένα εισόδου που χρειάζεται η μικροϋπηρεσία από τον τελικό χρήστη και από ακμές $E_a$, που περιλαμβάνουν την μέγιστη αποδεκτή καθυστέρηση επικοινωνίας μεταξύ των υπολογιστικών κόμβων εξυπηρέτησης των μικροϋπηρεσιών i-1, $i \in I_a$. Η τιμή μίας ακμής $E_a$ (i-1, i) συμβολίζεται και ως $\lambda_{a,i}$ για την αποδεκτή καθυστέρηση της μικροϋπηρεσίας $i \in I_a - \{1\}$ με την προηγούμενη της.


*Εικόνα 4-6 Παράδειγμα εφαρμογής*

Ο στόχος του προβλήματος είναι να κατανεμηθούν οι μικροϋπηρεσίες της κάθε εφαρμογής με αποδοτικό τρόπο με βάση διαφορετικά κριτήρια. Τα κριτήρια που εξετάζονται είναι η ελαχιστοποίηση του συνολικού χρόνου απόκρισης, του συνολικού κόστος εξυπηρέτησης των εφαρμογών, του πλήθους των δανειζόμενων πόρων μεταξύ των aggregators που συνεργάζονται, του πλήθους των ανακατανομών κάποιων μικροϋπηρεσιών (soft handover) και των ανακατανομών ολόκληρων των εφαρμογών σε νέο aggregator (hard handover) και του πλήθους των ενεργών κόμβων, όπου ενεργός κόμβος θεωρείται ο κόμβος όπου χρησιμοποιείται για την εκτέλεση τουλάχιστον μίας μικροϋπηρεσίας.

## 4.3 Πρόβλημα Μεικτού Ακέραιου Γραμμικού Προγραμματισμού

Υποδομή

| Σύμβολο | Ερμηνεία |
|---|---|
| G | Μη κατευθυνόμενος Βεβαρημένος Γράφος για την Υποδομή |
| V | Σύνολο κόμβων της υποδομής |
| $c_v$ | CPU χωρητικότητα του κόμβου $v \in V$ |
| $r_v$ | RAM χωρητικότητα του κόμβου $v \in V$ |
| $o_v$ | Κόστος χρήσης του κόμβου $v \in V$ |
| $n_v$ | Συντελεστής κόστους δικτύου του κόμβου $v \in V$ |

| | |
|---|---|
| $x_v$, $y_v$ | Συντεταγμένες του κόμβου $v \in V$ |
| $l_{v,v'}$ | Καθυστέρηση επικοινωνίας μεταξύ των κόμβων $v,v' \in V$ |

Aggregators

| | |
|---|---|
| $N$ | Σύνολο aggregators |
| $N_n$ | Σύνολο κόμβων που ανήκουν στον aggregator $n \in N$ |
| $C$ | Σύνολο συνεργασιών aggregators |
| $Z_c$ | Σύνολο aggregators $n \in N$ που ανήκουν στην συνεργασία $c \in C$ |
| $V_c$ | Σύνολο κόμβων $v \in V$ που ανήκουν στην συνεργασία $c \in C$ |

Εφαρμογές

| | |
|---|---|
| $G_a$ | Μη κατευθυνόμενη Βεβαρημένη Αλυσίδα της εφαρμογής $a \in A$ |
| $A$ | Σύνολο cloud-native εφαρμογών |
| $\chi_{a,t}$, $\psi_{a,t}$ | Συντεταγμένες χρήστη της εφαρμογής $a \in A$ την χρονική στιγμή $t \in [st_a, ft_a]$ |
| $I_a$ | Σύνολο μικροϋπηρεσίων της εφαρμογής $a \in A$ |
| $\varepsilon_{a,i}$ | CPU απαιτήσεις της μικροϋπηρεσίας $i \in I_a$ |
| $\rho_{a,i}$ | RAM απαιτήσεις της μικροϋπηρεσίας $i \in I_a$ |
| $s_{a,i}$ | Μέγεθος δεδομένων εισόδου της μικροϋπηρεσίας $i \in I_a$ |
| $\lambda_{a,i}$ | Αποδεκτή καθυστέρηση μεταξύ των μικροϋπηρεσιών $i \in I_a$-{1} και i-1 |
| $\Lambda_a$ | Συνολική αποδεκτή καθυστέρηση της εφαρμογής $a \in A$ |
| $T$ | Συνολική διάρκεια του προβλήματος |
| $st_a$ | Χρονική στιγμή έναρξης εφαρμογής $a \in A$ |
| $ft_a$ | Χρονική στιγμή λήξης εφαρμογής $a \in A$ |
| $\tau_{a,t}$ | Δυαδική παράμετρος αν η εφαρμογή $a \in A$ είναι ενεργή την χρονική στιγμή $t \in T$ |

Μεταβλητές Απόφασης

| | |
|---|---|
| $p_{v,a,i,t}$ | Δυαδική μεταβλητή για την τοποθέτηση της μικροϋπηρεσίας $i \in I_a$ στον κόμβο $v \in V$ την χρονική στιγμή $t \in [st_a, ft_a]$ |
| $q_{n,a,t}$ | Δυαδική μεταβλητή για την τοποθέτηση της εφαρμογής $a \in A$ στον aggregator $n \in N$ την χρονική στιγμή $t \in [st_a, ft_a]$ |

Μεταβλητές Βελτιστοποίησης

| | |
|---|---|
| $\sigma_{a,t}$ | Συνολικό κόστος για την εξυπηρέτηση της εφαρμογής $a \in A$ την χρονική στιγμή $t \in [st_a, ft_a]$ |
| $\theta_{a,i,t}$ | Καθυστέρηση επικοινωνίας της μικροϋπηρεσίας $i \in I_a$ της εφαρμογής $a \in A$ την χρονική στιγμή $t \in [st_a, ft_a]$ |
| $b_{n,n2,a,t}$ | Πλήθος μικροϋπηρεσιών $i \in I_a$ εφαρμογής $a \in A$ που έχει αναλάβει ο aggregator $n \in N$ και χρησιμοποιούν κόμβους του aggregator $n2 \in N$ την χρονική στιγμή $t \in [st_a, ft_a]$ |

| | |
|---|---|
| $SH_{a,i,t}$ | Δυαδική μεταβλητή για τον υπολογισμό αν η μικροϋπηρεσία i ∈Ia της εφαρμογής a∈A άλλαξε κόμβο εντός της ίδιας συνεργασίας aggregators μεταξύ της χρονικής στιγμής t-1 και t ∈[ $st_a$+1, $ft_a$] |
| $HH_{a,t}$ | Δυαδική μεταβλητή για τον υπολογισμό αν ο aggregator της εφαρμογής a∈A άλλαξε μεταξύ της χρονικής στιγμής t-1 και t ∈[ $st_a$+1, $ft_a$] |
| $k_{v,t}$ | Δυαδική μεταβλητή για τον υπολογισμό αν ο κόμβος v∈V είναι ενεργός την χρονική στιγμή t∈T |
| w1, w2, w3, w4,w5 | Συντελεστές βάρους για τον στόχο βελτιστοποίησης |

*Πίνακας 4-2 Περίληψη παραμέτρων προβλήματος*

Συνάρτηση Βελτιστοποίησης:

$$\min\left\{ \mathrm{w1}\sum_{\mathrm{a=1}}^{\mathrm{A}}\sum_{\mathrm{t=st_a}}^{\mathrm{ft_a}}\sigma_{\mathrm{a,t}} + \mathrm{w2}\sum_{\mathrm{a=1}}^{\mathrm{A}}\sum_{\mathrm{i=1}}^{\mathrm{I_a}}\sum_{\mathrm{t=st_a}}^{\mathrm{ft_a}}\theta_{\mathrm{a,i,t}} + \mathrm{w3}\sum_{c\in C}\sum_{n\in Z_c}\sum_{n2\in Z_c}\sum_{a=1}^{A}\sum_{t=st_a}^{ft_a} b_{n,n2,a,t} \right.$$
$$+ \mathrm{w4}\sum_{\mathrm{a=1}}^{\mathrm{A}}\sum_{\mathrm{i=1}}^{\mathrm{I_a}}\sum_{\mathrm{t=st_a+1}}^{\mathrm{ft_a}} SH_{\mathrm{a,i,t}} + w5\sum_{\mathrm{a=1}}^{\mathrm{A}}\sum_{\mathrm{t=st_a+1}}^{\mathrm{ft_a}} \mathrm{HH_{a,t}}$$
$$\left. + (1 - \mathrm{w1} - \mathrm{w2} - w3 - w4 - w5)\sum_{\mathrm{t=1}}^{\mathrm{T}}\sum_{\mathrm{v=1}}^{\mathrm{V}} \mathrm{k_{v,t}} \right\}$$

Υπό τους περιορισμούς:

1. Ένα προς ένα αντιστοίχιση μεταξύ των μικροϋπηρεσιών και των υπολογιστικών κόμβων. Κάθε μικροϋπηρεσία πρέπει να εξυπηρετηθεί από ακριβώς έναν υπολογιστικό κόμβο της υποδομής.

$$\sum_{\mathrm{v=1}}^{\mathrm{V}} \mathrm{p_{v,a,i,t}} = 1\,, \forall a \in A, i \in I_a, t \in [st_a, ft_a]$$

2. Ένα προς ένα αντιστοίχιση μεταξύ των cloud-native εφαρμογών και των aggregators. Κάθε εφαρμογή πρέπει να εξυπηρετηθεί από ακριβώς έναν aggregator.

$$\sum_{\mathrm{n=1}}^{\mathrm{N}} \mathrm{q_{n,a,t}} = 1\,, \forall a \in A, t \in [st_a, ft_a]$$

3. Ένας υπολογιστικός κόμβος που εξυπηρετεί μία μικροϋπηρεσία της εφαρμογής πρέπει να ανήκει στον aggregator που εξυπηρετεί την εφαρμογή ή σε κόμβους aggregators που συνεργάζονται με αυτόν.

$$\sum_{\mathrm{v}\in \mathrm{V_c}} \mathrm{p_{v,a,i,t}} = \sum_{\mathrm{n}\in \mathrm{Z_c}} \mathrm{q_{n,a,t}}\,, \forall c \in C, a \in A, i \in I_a, t \in [st_a, ft_a]$$

4. Υπολογισμός της καθυστέρησης επικοινωνίας της πρώτη μικροϋπηρεσίας με τον κόμβο εξυπηρέτησης της.

$$\theta_{a,1,t} = \sum_{v=1}^{V} (p_{v,a,1,t} * \sqrt{(x_v - \chi_{a,t})^2 + (y_v - \psi_{a,t})^2}), \forall \mathrm{a} \in \mathrm{A}, \mathrm{t} \in [st_a, ft_a]$$

5. Η βοηθητική δυαδική μεταβλητή $z_{a,i,t,v,v2}$ δείχνει αν οι κόμβοι v,v2 χρησιμοποιούνται για την εξυπηρέτηση δύο διαδοχικών μικροϋπηρεσιών μίας εφαρμογής.

$$\begin{cases} z_{a,i,t,v,v2} \leq p_{v,a,i,t} \\ z_{a,i,t,v,v2} \leq p_{v2,a,i-1,t} \\ z_{a,i,t,v,v2} \geq p_{v,a,i,t} + p_{v2,a,i-1,t} - 1 \end{cases}, \forall \begin{matrix} a \in A, i \in I_a - \{1\}, \\ t \in [st_a, ft_a], v, v2 \in V \end{matrix}$$

Υπολογισμός της καθυστέρησης μεταξύ των υπολογιστικών κόμβων που θα εξυπηρετήσουν δύο διαδοχικές μικροϋπηρεσίες.

$$\theta_{a,i,t} = \sum_{v \in V} \sum_{v2 \in V} I_{v,v2} * z_{a,i,t,v,v2}, \forall a \in A, i \in I_a - \{1\}, t \in [st_a, ft_a]$$

Η καθυστέρηση μεταξύ των υπολογιστικών κόμβων που θα εξυπηρετήσουν δύο διαδοχικές μικροϋπηρεσίες δεν πρέπει να ξεπερνά το μεταξύ τους όριο ούτε το συνολικό όριο της εφαρμογής.

$$\theta_{\mathrm{a,i,t}} \leq \lambda_{\mathrm{a,i}}, \forall a \in A, i \in I_a - \{1\}, t \in [st_a, ft_a]$$

6. Ο συνολικός χρόνος εξυπηρέτησης μίας εφαρμογής δεν πρέπει να ξεπερνάει την συνολική αποδεκτή καθυστέρησή της.

$$\sum_{\mathrm{i}=1}^{\mathrm{I_a}} \theta_{\mathrm{a,i,t}} \leq \Lambda_\alpha, \forall \mathrm{a} \in \mathrm{A}, \mathrm{t} \in [st_a, ft_a]$$

7. Η συνολική ζήτηση σε υπολογιστική ισχύ των μικροϋπηρεσιών που θα εξυπηρετηθούν από έναν κόμβο δεν πρέπει να ξεπερνά την υπολογιστική ισχύ του κόμβου.

$$\sum_{\mathrm{a}=1}^{\mathrm{A}} \sum_{\mathrm{i}=1}^{\mathrm{I_a}} \tau_{\mathrm{a,t}} * \varepsilon_{\mathrm{a,i}} * \mathrm{p_{v,a,i,t}} \leq \mathrm{c_v}, \forall v \in V, t \in T$$

8. Η συνολική ζήτηση σε μνήμη RAM των μικροϋπηρεσιών που θα εξυπηρετηθούν από έναν κόμβο δεν πρέπει να ξεπερνά την μνήμη RAM του κόμβου.

$$\sum_{\mathrm{a}=1}^{\mathrm{A}} \sum_{\mathrm{i}=1}^{\mathrm{I_a}} \tau_{\mathrm{a,t}} * \rho_{\mathrm{a,i}} * \mathrm{p_{v,a,i,t}} \leq \mathrm{r_v}, \forall v \in V, t \in T$$

9. Υπολογισμός του κόστους χρήσης των υπολογιστικών πόρων για μια εφαρμογή.

$$\sigma_{\mathrm{a,t}} = \sum_{\mathrm{v=1}}^{\mathrm{V}} \sum_{\mathrm{i=1}}^{\mathrm{I_a}} \left(\mathrm{o_v} + \mathrm{n_v} * \mathrm{s_{a,i}}\right) \mathrm{p_{v,a,i,t}}, \forall \mathrm{a} \in \mathrm{A}, \mathrm{t} \in [st_a, ft_a]$$

10. Υπολογισμός πλήθους μικροϋπηρεσιών μίας εφαρμογής όπου ο aggregator n χρησιμοποιεί πόρους του aggregator n2 με τον οποίο συνεργάζεται για την εξυπηρέτηση τους για μία χρονική στιγμή.

$$\begin{cases} \mathrm{b_{n,n2,}}a,t \geq \sum_{\mathrm{i=1}}^{\mathrm{I_a}} \sum_{\mathrm{v}\epsilon\mathrm{N_{n2}}} \mathrm{p_{v,a,i,t}} - \left(1 - \mathrm{q_{n,a,t}}\right) \sum_{i=1}^{\mathrm{I}_a} 1 \\ \mathrm{b_{n,n2,}}a,t \geq 0 \end{cases}, \forall \begin{matrix} c \in C, n, n2 \in c, \\ a \in A, t \in [st_a, ft_a] \end{matrix}$$

11. Υπολογισμός αν απαιτείται ένα soft handover για την εξυπηρέτηση μίας μικροϋπηρεσίας για μία χρονική στιγμή.

$$\mathrm{p}_{v,\mathrm{a,i,t}} + \mathrm{p}_{v2,\mathrm{a,}i,\mathrm{t-1}} \leq SH_{\mathrm{a,i,t}} + 1, \forall v, v2 \in V, a \in A, i \in I_a, \mathrm{t} \in [st_a + 1, ft_a]$$

12. Υπολογισμός αν απαιτείται hard handover για την εξυπηρέτηση μίας εφαρμογής για μία χρονική στιγμή.

$$\mathrm{q_{n,a,t}} + \mathrm{q_{n2,a,t-1}} \leq \mathrm{HH_{a,t}} + 1, \forall c \in C, n \in c, n2 \notin c, a \in A, \mathrm{t} \in [st_a, +1ft_a]$$

13. Υπολογισμός των ενεργών κόμβων, δηλαδή των κόμβων που χρησιμοποιούνται για την εκτέλεση τουλάχιστον μίας μικροϋπηρεσίας.

$$\mathrm{k_{v,t}} \geq \mathrm{p_{v,a,i,t}}, \forall v \in V, a \in A, i \in I_a, t \in [st_a, ft_a]$$

Η συνάρτηση βελτιστοποίησης είναι το βεβαρημένο άθροισμα της καθυστέρηση επικοινωνίας και του κόστους εξυπηρέτησης για κάθε εφαρμογή, του πλήθους των δανεισμών μεταξύ των aggregators, του πλήθους των soft και των hard handovers και του πλήθους των ενεργών κόμβων για την εξυπηρέτηση όλων των εφαρμογών. Συγκεκριμένα, όταν w1=1 (w2=w3=w4=w5=0) υπολογίζεται το πρόβλημα ελαχιστοποίησης κόστους, για w2=1 (w1=w3=w4=w5=0) υπολογίζεται το πρόβλημα ελαχιστοποίησης της καθυστέρησης επικοινωνίας, για w3=1 (w1=w2=w4=w5=0) λαμβάνεται υπόψη μόνο η ελαχιστοποίηση του πλήθους των δανεισμών μεταξύ των aggregators, για w4=1 (w1=w2=w3=w5=0) ελαχιστοποιείται το πλήθος των ανακατανομών μικροϋπηρεσιών εντός κάθε συνεργασίας aggregators, ενώ για w5=1 (w1=w2=w3=w4=0) ελαχιστοποιείται το πλήθος των ανακατανομών ολόκληρων εφαρμογών σε νέους aggregators, και, τέλος, για w1=w2=w3=w4=w5=0 λύνεται το πρόβλημα ελαχιστοποίησης των ενεργών κόμβων για την εξυπηρέτηση όλων των εφαρμογών. Για τις ενδιάμεσες τιμές των βαρών λαμβάνονται υπόψη οι παράμετροι με διαφορετική συνεισφορά

στην ελαχιστοποίηση της τελικής τιμής, όλα τα βάρη και το άθροισμα τους πρέπει να είναι μεταξύ 0 και 1.

Οι περιορισμοί 1 και 2 εξασφαλίζουν ότι χρησιμοποιείται ακριβώς ένας υπολογιστικός κόμβος για την εξυπηρέτηση των μικροϋπηρεσιών και ακριβώς ένας aggregator αντίστοιχα για την κάθε εφαρμογή. Ο περιορισμός 3 εξασφαλίζει ότι όλοι οι κόμβοι που χρησιμοποιούνται για την εξυπηρέτηση μίας εφαρμογής ανήκουν στον aggregator που την έχει αναλάβει ή σε άλλους που συνεργάζονται με αυτόν. Οι περιορισμοί 4-6 υπολογίζουν την καθυστέρηση εξυπηρέτησης για κάθε μικροϋπηρεσία, όπου ο περιορισμός 4 υπολογίζει την καθυστέρηση για την πρώτη μικροϋπηρεσία και ο περιορισμός 5 για τις υπόλοιπες τηρώντας την αποδεκτή καθυστέρηση μεταξύ διαδοχικών μικροϋπηρεσιών και ο περιορισμός 6 εξασφαλίζει η συνολική καθυστέρηση της εφαρμογής να είναι αποδεκτή. Οι περιορισμοί 7 και 8 ελέγχουν οι μικροϋπηρεσίες που δρομολογούνται σε έναν κόμβο να μην απαιτούν περισσότερους από τους διαθέσιμους πόρους, ενώ ο περιορισμός 9 υπολογίζει το συνολικό κόστος για την εξυπηρέτηση κάθε εφαρμογής. Ο περιορισμός 10 υπολογίζει το πλήθος των μικροϋπηρεσιών που ένας aggregator χρησιμοποιεί πόρους άλλου aggregator για την εξυπηρέτηση τους. Οι περιορισμοί 11 και 12 υπολογίζουν τα soft και τα hard handovers που απαιτούνται. Τέλος, ο περιορισμός 13 υπολογίζει αν ένας κόμβος είναι ενεργός για μία χρονική στιγμή. Η προτεινόμενη διατύπωση του προβλήματος μπορεί να χρησιμοποιηθεί και για την διαχείριση μονολιθικών εφαρμογών σε edge cloud υποδομές, όπου οι μονολιθικές εφαρμογές μπορούν να αντιμετωπιστούν ως εφαρμογές με μία μόνο μικροϋπηρεσία.

# 5 Άπληστος Αλγόριθμος

Το συγκεκριμένο πρόβλημα ανήκει στην κλάση NP-hard προβλημάτων και η MILP αντιμετώπισή του είναι χρονοβόρα και υπολογιστικά ακριβή ακόμα και για σενάρια με λίγες εφαρμογές και κόμβους. Συγκεκριμένα, για ένα πρόβλημα με A εφαρμογές και I μέγιστο αριθμό μικροϋπηρεσιών η καθεμία, δηλαδή για ένα φόρτο εργασίας A*I, για μία τοπολογία K κόμβων και T χρονικών στιγμών, η τοποθέτηση τους σε κόμβους της τοπολογίας αποτελείται από K*A*I δυνατούς συνδυασμούς κάθε χρονική στιγμή. Αντίστοιχα, για N aggregators η επιλογή aggregator για τις εφαρμογές ανέρχεται σε N*A κάθε χρονική στιγμή. Ο συνολικός αριθμός των μεταβλητών απόφασης ανέρχεται σε (K*A*I + N*A)*T και οι περιορισμοί περιλαμβάνουν πολλαπλάσιες ανισώσεις.

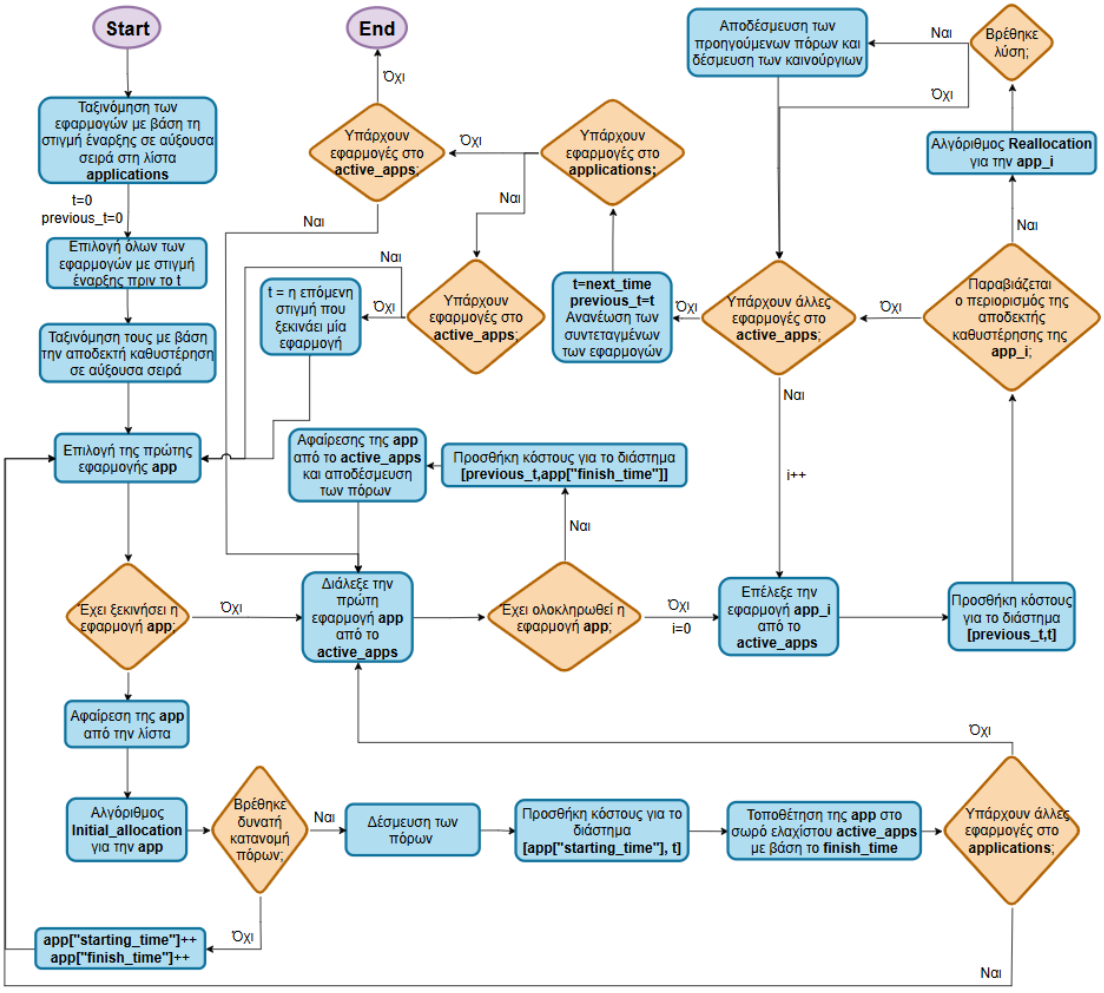
Συνεπώς, για την αποτελεσματικότερη επίλυση του προβλήματος παρουσιάζεται ένας άπληστος ευριστικός αλγόριθμος (greedy heuristic algorithm) που προσπαθεί να προσεγγίσει την βέλτιστη τοποθέτηση κάθε εφαρμογής, δεδομένων των περιορισμών στη χωρητικότητα των κόμβων και της αποδεκτής καθυστέρησης των εφαρμογών. Η λύση του προβλήματος αποτελείται από τρεις αλγόριθμους: τον βασικό αλγόριθμο, που περικλείει ολόκληρο το πρόβλημα και καλεί τους άλλους αλγόριθμους οπότε χρειάζονται, τον αλγόριθμο αρχικής κατανομής, ο οποίος αποφασίζει την κατανομή μίας εφαρμογής στους κόμβους της υποδομής και τον αλγόριθμο ανακατανομής, ο οποίος φροντίζει για την αλλαγή μικροϋπηρεσιών σε καλύτερους κόμβους ανάλογα με την θέση της εφαρμογής στο χώρο.

## 5.1 Βασικός Αλγόριθμος

Ο βασικός αλγόριθμος είναι υπεύθυνος για την εκτέλεση του προβλήματος, τον έλεγχο της υποδομής και την ενημέρωση της ανάλογα με την κατάσταση κάθε χρονική στιγμή. Ο αλγόριθμος παίρνει ως είσοδο την υποδομή G=(V, E) και το σύνολο των εφαρμογών $a \in A$ με Ga = (Va, Ea) και επιστρέφει την τελική τιμή της συνάρτησης βελτιστοποίησης που υπολογίστηκε και την κατανομή των εφαρμογών για κάθε χρονική στιγμή.

Αρχικά, ο αλγόριθμος ταξινομεί όλες τις εφαρμογές με βάση την χρονική στιγμή έναρξης τους και ξεκινάει για την στιγμή που φτάνει η πρώτη εφαρμογή. Στη συνέχεια, επιλέγει όλες τις εφαρμογές που ξεκινάνε αυτήν την στιγμή και τις ταξινομεί με βάση την αποδεκτή καθυστέρηση της κάθε εφαρμογής. Η ταξινόμηση αυτή γίνεται με σκοπό οι εφαρμογές που απαιτούν να χρησιμοποιήσουν κοντινούς κόμβους στην τοποθεσία τους, οι οποίοι όμως έχουν περιορισμένους υπολογιστικούς πόρους, να έχουν προτεραιότητα σε αντίθεση με τις εφαρμογές που μπορούν να εξυπηρετηθούν και από τους κόμβους του cloud που έχουν πρακτικά απεριόριστους πόρους. Για τις ταξινομημένες εφαρμογές καλεί τον

αλγόριθμο αρχικής κατανομής για κάθε μία με τη σειρά, αν δεν βρεθεί κατανομή για μία εφαρμογή τότε η εφαρμογή ξαναμπαίνει στις εφαρμογές προς κατανομή και θα δοκιμαστεί ξανά την επόμενη χρονική στιγμή. Για τις εφαρμογές που βρέθηκε κατανομή δεσμεύονται οι υπολογιστικοί πόροι της υποδομής που χρησιμοποιούνται, προστίθεται το αποτέλεσμα της συνάρτησης βελτιστοποίησης στην τελική τιμή και τοποθετούνται στις εφαρμογές που εξυπηρετούνται.




*Εικόνα 5-1 Τυπικό διάγραμμα ροής βασικού αλγορίθμου*

Αφού ολοκληρωθεί η τοποθέτηση των εφαρμογών που ξεκινάνε αυτήν την χρονική στιγμή, ελέγχονται οι εφαρμογές που εκτελούνται ήδη. Οι εφαρμογές ελέγχονται διαδοχικά, αν κάποια ολοκληρώνεται αυτήν την χρονική στιγμή, τότε προστίθεται το αποτέλεσμα της συνάρτησης βελτιστοποίησης στην τελική τιμή και για αυτήν την στιγμή, η εφαρμογή αφαιρείται από τις εφαρμογές που εξυπηρετούνται και ελευθερώνονται οι υπολογιστικοί πόροι που χρησιμοποιεί από την επόμενη χρονική στιγμή. Για τις εφαρμογές που δεν ολοκληρώνονται, ελέγχεται αν παραβιάζεται η αποδεκτή καθυστέρηση για κάθε μία από τις εφαρμογές και αν δεν παραβιάζεται προστίθεται το αποτέλεσμα της συνάρτησης βελτιστοποίησης στην τελική τιμή και συνεχίζει στην επόμενη. Στην περίπτωση

που παραβιάζεται ο περιορισμός, ο αλγόριθμος καλεί τον αλγόριθμο ανακατανομής και είτε κάνει τις απαραίτητες αλλαγές που αποφασίστηκαν είτε αν δεν βρεθεί λύση κρατάει την εφαρμογή στους ίδιους κόμβους.

Αφού ολοκληρωθεί και αυτή η διαδικασία για όλες τις εφαρμογές, ο αλγόριθμος συνεχίζει στην επόμενη χρονική στιγμή. Τέλος, ο αλγόριθμος σταματάει όταν δεν υπάρχουν εφαρμογές προς εκτέλεση ούτε εφαρμογές που εκτελούνται στην υποδομή. Μια τυπική εκτέλεση του αλγορίθμου παρουσιάζεται στο διάγραμμα ροής στην Εικόνα 5-1.

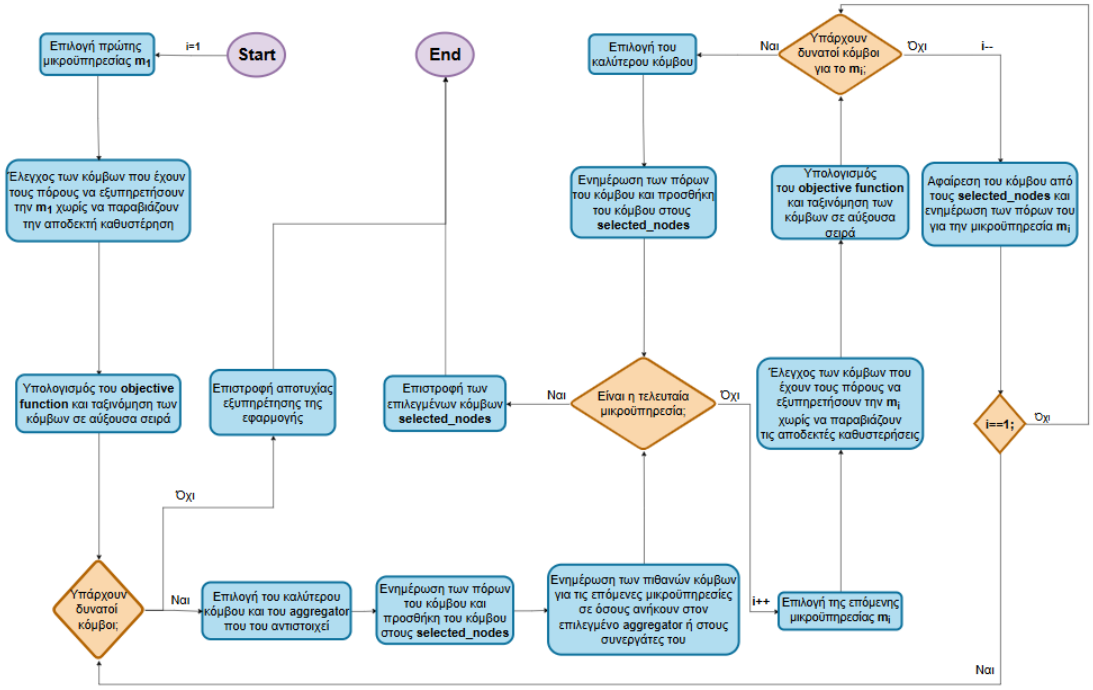
## 5.2 Αλγόριθμος Αρχικής Κατανομής (Initial Allocation)

Ο άπληστος ευριστικός αλγόριθμος αρχικής κατανομής αναζητεί μια ικανοποιητική λύση στο πρόβλημα, εξυπηρετώντας τις μικροϋπηρεσίες των εφαρμογών μια-προς-μια με την τεχνική καλύτερου ταιριάσματος (best-fit). Ο αλγόριθμος παίρνει ως είσοδο την υποδομή G=(V, E) στην κατάσταση της εκείνη την χρονική στιγμή δεδομένου των πόρων που χρησιμοποιούνται ήδη και μία εφαρμογή a με $G_a = (V_a, E_a)$ και επιστρέφει την κατανομή των μικροϋπηρεσιών της στους κόμβους.

Αρχικά, ο αλγόριθμος επιλέγει την πρώτη μικροϋπηρεσία και ελέγχει ποιοι κόμβοι διαθέτουν επαρκή πόρους για την εξυπηρέτηση της και είναι σε απόσταση που δεν παραβιάζουν την αποδεκτή καθυστέρηση της εφαρμογής, και στη συνέχεια τους ταξινομεί σε αύξουσα σειρά με βάση την συνάρτηση βελτιστοποίησης που έχει παρουσιαστεί στην 4.3. Ο καλύτερος κόμβος επιλέγεται και οι πόροι που πρόκειται να χρησιμοποιήσει αυτή η μικροϋπηρεσία θεωρούνται δεσμευμένοι. Επίσης, επιλέγεται ο aggregator στον οποίο ανήκει αυτός ο κόμβος ως υπεύθυνος για ολόκληρη την εφαρμογή και πλέον οι διαθέσιμοι κόμβοι για τις υπόλοιπες μικροϋπηρεσίες αποτελούνται μόνο από τους κόμβους αυτού του aggregator ή από κόμβους aggregators που συνεργάζονται μαζί του.

Αν η εφαρμογή αποτελείται από περισσότερες από μία μικροϋπηρεσίες, επιλέγεται η επόμενη μικροϋπηρεσία της αλυσίδας. Η ίδια διαδικασία ακολουθείται για την επιλογή του καλύτερου κόμβου για την επόμενη μικροϋπηρεσία λαμβάνοντας υπόψη και την αποδεκτή καθυστέρηση μεταξύ των διαδοχικών μικροϋπηρεσιών. Συνεπώς, δεδομένου της τοποθεσίας της πρώτης μικροϋπηρεσίας, ο αλγόριθμος επιλέγει τους κόμβους που παρουσιάζουν καθυστέρηση επικοινωνίας μικρότερη του ορίου της $l_{v1,v2} \leq \lambda_{a,2}$ και που η συνολική καθυστέρηση λόγω και της καθυστέρησης της πρώτης μικροϋπηρεσίας $\theta_{a,1}$ είναι μικρότερη του ορίου $\theta_{a,1}+l_{v1,v2} \leq \Lambda_a$. Αν υπάρχουν περισσότεροι από ένας δυνατός κόμβος επιλέγεται ο καλύτερος, ο οποίος μπορεί να είναι και ο ίδιος της πρώτης μικροϋπηρεσίας. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να τοποθετηθούν όλες οι μικροϋπηρεσίες σε κάποιον κόμβο της υποδομής.

Είναι πιθανό να μην είναι δυνατό να τοποθετηθεί μία μικροϋπηρεσία στους διαθέσιμους κόμβους της υποδομής, σε αυτήν την περίπτωση ο αλγόριθμος αναιρεί την επιλογή που είχε κάνει για την προηγούμενη μικροϋπηρεσία και επιλέγει τον αμέσως επόμενο καλύτερο κόμβο που είχε υπολογίσει. Αν δεν υπάρχουν άλλοι διαθέσιμοι κόμβοι για την προηγούμενη μικροϋπηρεσία συνεχίζει αναιρώντας και για την προηγούμενη μικροϋπηρεσία αυτής και επιλέγοντας τον επόμενο. Έτσι, ο αλγόριθμος εκτελεί μία αναζήτηση κατά βάθος (depth first search DFS) στις δυνατές κατανομές των μικροϋπηρεσιών σε κόμβους μέχρι να βρει μία εφικτή λύση, όταν βρεθεί μία λύση, ο αλγόριθμος επιστρέφει την κατανομή των μικροϋπηρεσίων που βρήκε στον βασικό αλγόριθμο. Στην περίπτωση που δεν βρέθηκε λύση ο αλγόριθμος επιστρέφει ότι απέτυχε και η εφαρμογή αναμένει για να ξαναδοκιμάσει να δρομολογηθεί όταν τελειώσει κάποια εφαρμογή.




*Εικόνα 5-2 Τυπικό διάγραμμα ροής αλγορίθμου αρχικής κατανομής*

Ο αλγόριθμος εγγυάται ότι θα βρει λύση αν υπάρχει για την συγκεκριμένη εφαρμογή δεδομένων των εφαρμογών που χρησιμοποιούν ήδη πόρους της υποδομής. Για να πετύχει καλύτερη απόδοση στη μέση περίπτωση, ο αλγόριθμος σταματάει όταν βρει μία εφικτή λύση χωρίς να μπορεί να εγγυηθεί ότι αυτή η λύση είναι η βέλτιστη. Ο αλγόριθμος αυτός στην χειρότερη περίπτωση, που όλες οι μικροϋπηρεσίες μπορούν να τοποθετηθούν σε όλους τους κόμβους εκτός από την τελευταία μικροϋπηρεσία που δεν μπορεί να τοποθετηθεί πουθενά, έχει υπολογιστική πολυπλοκότητα $O(V * V^{I_a}) = O(V^{I_a+1})$ , που είναι πολυωνυμική ως προς το πλήθος των κόμβων της υποδομής για μικρό αριθμό μικροϋπηρεσιών. Ωστόσο μπορεί να βελτιωθεί αν είναι αποδεκτή η πιθανή αποτυχία εύρεσης λύσης αποθηκεύοντας μόνο τους N καλύτερους κόμβους για κάθε μικροϋπηρεσία οπότε

η πολυπλοκότητα γίνεται $O(V * N^{I_a})$ ή επιλέγοντας να αποθηκεύει όλους τους δυνατούς κόμβους μόνο για την πρώτη μικροϋπηρεσία οπότε η πολυπλοκότητα γίνεται $O(I_\alpha * V^2)$. Μια τυπική εκτέλεση του αλγορίθμου παρουσιάζεται στο διάγραμμα ροής στην Εικόνα 5-2.

## 5.3 Αλγόριθμος Ανακατανομής (Reallocation)

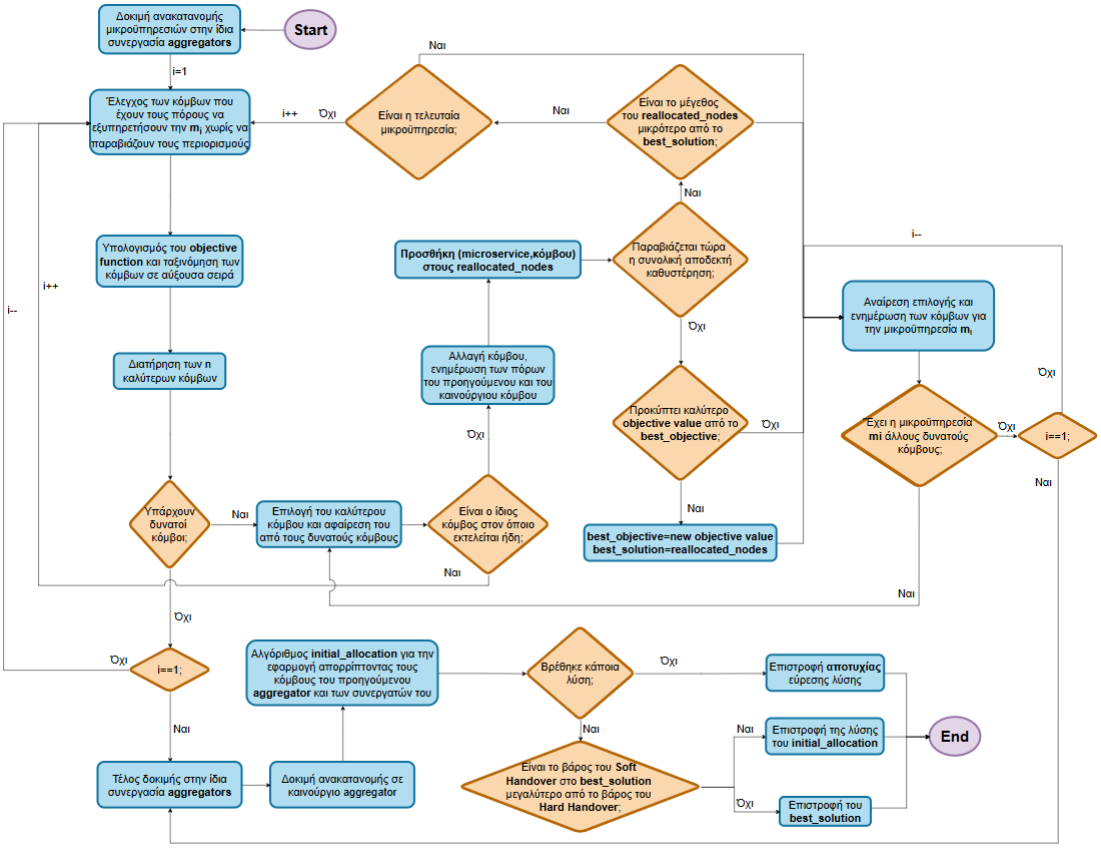
Ο άπληστος αλγόριθμος ανακατανομής στοχεύει στην προσαρμογή των κόμβων της υποδομής που εκτελούν τις μικροϋπηρεσίες μίας εφαρμογής ώστε να μην παραβιάζεται ο περιορισμός της αποδεκτής καθυστέρησής της κατά την διάρκεια της κίνησής της. Ο αλγόριθμος παίρνει ως είσοδο την υποδομή G=(V, E) στην κατάσταση της εκείνη την χρονική στιγμή δεδομένου των υπολογιστικών πόρων που χρησιμοποιούνται ήδη και μία εφαρμογή a με $G_a$ = ($V_a$, $E_a$) της οποίας παραβιάζεται η αποδεκτή καθυστέρηση καθώς και την τρέχουσα κατανομής στους κόμβους της υποδομής. Ο αλγόριθμος επιστρέφει τις τροποποιήσεις που απαιτούνται ώστε η συνολική καθυστέρηση να είναι σε αποδεκτά όρια.

Αρχικά, ο αλγόριθμος προσπαθεί να βρει ανακατανομή μερικών μικροϋπηρεσιών σε κόμβους της ίδιας συνεργασίας aggregators με το λιγότερο πλήθος ανακατανομών που χρειάζονται. Ο αλγόριθμος ξεκινάει από την πρώτη μικροϋπηρεσία και υπολογίζει τους κόμβους που μπορούν να την εξυπηρετήσουν και τους ταξινομεί με βάση την συνάρτηση βελτιστοποίησης αποθηκεύοντας τους N καλύτερους κόμβους. Ο καλύτερος κόμβος επιλέγεται κι αν είναι ο ίδιος κόμβος στον οποίο εκτελείται ήδη τότε συνεχίζει στην επόμενη μικροϋπηρεσία αν υπάρχει, αλλιώς θεωρεί ότι η μικροϋπηρεσία μεταφέρεται σε αυτόν τον κόμβο και ελέγχει αν συνεχίζει να παραβιάζεται ο περιορισμός της αποδεκτής καθυστέρησης. Στην περίπτωση που σταμάτησε να παραβιάζεται, ο αλγόριθμος αποθηκεύει την λύση που βρέθηκε και συνεχίζει ψάχνοντας λύσεις που οδηγούν σε καλύτερη συνάρτηση βελτιστοποίησης με τον ίδιο ή καλύτερο αριθμό soft handovers, ενώ στην περίπτωση που συνεχίζει να παραβιάζεται συνεχίζει ελέγχοντας για την επόμενη μικροϋπηρεσία. Ο αλγόριθμος συνεχίζει με παρόμοιο τρόπο για τις επόμενες μικροϋπηρεσίες επιλέγοντας τους N καλύτερους κόμβους και διαλέγει τον καλύτερο μέχρι να επανέλθει η καθυστέρηση σε αποδεκτό όριο.

Όταν ο αλγόριθμος έχει φτάσει στην τελευταία μικροϋπηρεσία ή το πλήθος των αλλαγών που έχει κάνει στην τρέχουσα λύση είναι ίσο με την καλύτερη λύση που έχει βρει, τότε ο αλγόριθμος αναιρεί την τελευταία επιλογή που έκανε στην προηγούμενη μικροϋπηρεσία και διαλέγει τον επόμενο κόμβο, αν δεν υπάρχει επόμενος κόμβος συνεχίζει αναιρώντας και για την προηγούμενη μικροϋπηρεσία της. Έτσι, ο αλγόριθμος δοκιμάζει πολλές διαφορετικές λύσεις χωρίς να υπολογίζει περιττές λύσεις με περισσότερα soft handovers από όσα στην λύση που έχει ήδη βρεθεί. Στην περίπτωση που βρεθούν δύο λύσεις με τον ίδιο αριθμό από soft handovers, διατηρείται η λύση με την καλύτερη συνάρτηση βελτιστοποίησης. Η

διαδικασία αυτή σταματάει όταν εξαντληθούν οι πιθανές λύσεις με λιγότερες ή ίσες αλλαγές από την καλύτερη λύση.

Στη συνέχεια, ο αλγόριθμος δοκιμάζει να μεταφέρει ολόκληρη την εφαρμογή σε καινούργιο aggregator και να αλλάξουν όλες οι μικροϋπηρεσίες κόμβο εκτέλεσης. Ο αλγόριθμος καλεί τον αλγόριθμο αρχικής κατανομής απαγορεύοντας την χρήση των κόμβων του συγκεκριμένου aggregator και των συνεργατών του, καθώς αν υπήρχε λύση μέσα σε αυτήν την συνεργασία τότε θα είχε βρεθεί στην προηγούμενη διαδικασία. Στην περίπτωση που δεν βρέθηκε καμία λύση ούτε με soft ούτε με hard handover τότε ο αλγόριθμος επιστρέφει ότι δεν βρήκε εφικτή λύση, αλλιώς επιστρέφει την προτιμότερη λύση από τις δύο ανάλογα με τις τιμές που έχουμε δώσει στα βάρη για τον κάθε τύπο handover. Οι συντελεστές βάρους w4 και w5 χρησιμοποιούνται για να δηλώσουν την προτίμηση μεταξύ soft και hard handovers, για w4=1 προτιμάται το hard handover ενώ για w5=1 προτιμάται το soft handover. Στις ενδιάμεσες περιπτώσεις εξαρτάται από το πλήθος των μικροϋπηρεσιών που χρειάζεται να αλλάξουν κόμβο στο soft handover, δηλαδή αν το γινόμενο του πλήθους των μικροϋπηρεσιών με τον συντελεστή βάρους w4 είναι μικρότερο από τον συντελεστή βάρους w5 προτιμάται το soft handover αλλιώς προτιμάται το hard handover.




*Εικόνα 5-3 Τυπικό διάγραμμα ροής αλγορίθμου ανακατανομής*

Ο αλγόριθμος στο πρώτο σκέλος του εκτελεί μία αναζήτηση κατά βάθος στις δυνατές κατανομές των μικροϋπηρεσιών σε κόμβους απορρίπτοντας λύσεις που οδηγούν σε χειρότερο αποτέλεσμα από αυτό που έχει ήδη βρεθεί. Στην χειρότερη περίπτωση, όπου ο αλγόριθμος δοκιμάζει για όλες τις μικροϋπηρεσίες σε N κόμβους χωρίς να βρει καμία λύση ή η λύση που βρήκε να απαιτεί ανακατανομή όλων των μικροϋπηρεσιών, η υπολογιστική πολυπλοκότητα του πρώτου σκέλους του αλγορίθμου είναι $O(V_1 * N^{I_a})$, όπου $V_1$ είναι όλοι οι κόμβοι της συγκεκριμένης συνεργασίας aggregators. Στο δεύτερο σκέλος ο αλγόριθμος καλεί τον αλγόριθμο αρχικής κατανομής για τους aggregators εκτός της συνεργασίας, οπότε η υπολογιστική πολυπλοκότητα του είναι $O(V_2^{I_a+1})$, όπου $V_2$ είναι $V\text{-}V_1$. Τελικά, η πολυπλοκότητα του αλγορίθμου είναι $O\left(V_1 * N^{I_a} + V_2^{I_a+1}\right) = O(V^{I_a+1})$. Μια τυπική εκτέλεση του αλγορίθμου παρουσιάζεται στο διάγραμμα ροής στην Εικόνα 5-3.

# 6 Αποτελέσματα

Σε αυτήν την ενότητα εξετάζεται η αποτελεσματικότητα του άπληστου αλγορίθμου σε διαφορετικά σενάρια και συγκρίνεται με την βέλτιστη λύση που προκύπτει μέσω μεικτού ακέραιου γραμμικού προγραμματισμού. Αρχικά, παρουσιάζονται οι παράμετροι που χρησιμοποιήθηκαν στην προσομοίωση. Στη συνέχεια, αναλύεται η επίδοση του άπληστου ευριστικού αλγορίθμου σε σύγκριση με τη βέλτιστη λύση του προβλήματος γραμμικού προγραμματισμού. Έπειτα, εξετάζονται ξεχωριστά οι επιμέρους αλγόριθμοι και τα διαφορετικά κριτήρια. Τέλος, εξετάζονται τα αποτελέσματα ολόκληρου του ευριστικού αλγορίθμου σε κατάσταση συνεχούς εκτέλεσης για cloud-native εφαρμογές.

## 6.1 Παράμετροι Προσομοίωσης

Η υλοποίηση και η εκτέλεση των αλγορίθμων γίνεται με χρήση της Python. Η Python προσφέρει ένα μεγάλο σύνολο από βιβλιοθήκες για την διαχείριση των γραφημάτων, τον υπολογισμό προβλημάτων μικτού ακεραίου γραμμικού προγραμματισμού και την εξαγωγή γραφικών παραστάσεων. Συγκεκριμένα, χρησιμοποιήθηκε η βιβλιοθήκη networkx για την υλοποίηση των γραφημάτων, η βιβλιοθήκη pulp για την υλοποίηση του MILP προβλήματος και το Gurobi ως μηχανή επίλυσης του και η βιβλιοθήκη matplotlib για την εξαγωγή των αποτελεσμάτων.

Η υποδομή εκτείνεται στο edge-cloud συνεχές και οργανώνεται σε τρία ξεχωριστά επίπεδα υπολογιστικών πόρων, καθένα από τα οποία χαρακτηρίζεται από τις δικές του δυνατότητες. Αυτά τα επίπεδα αποτελούνται από near και far edge πόρους, που προσφέρουν δυνατότητες τοπικής και άμεσης επεξεργασίας με χαμηλότερες υπολογιστικές δυνατότητες, μέχρι cloud επεξεργαστές με περισσότερη υπολογιστική δύναμη με μεγαλύτερο χρόνο απόκρισης.

Εξετάζονται δύο διαφορετικές edge-cloud τοπολογίες για την αντιμετώπιση διαφορετικών πειραματικών σεναρίων. Αρχικά, η “βασική τοπολογία” αποτελείται από 20 near edge κόμβους, 5 far edge κόμβους και 2 cloud κόμβους. Η βασική τοπολογία στοχεύει στην σύγκριση της μη βέλτιστης λύσης του άπληστου αλγορίθμου με την βέλτιστη λύση του MILP, που μπορεί να υπολογιστεί αποδοτικά σε τέτοιες μικρής έκτασης τοπολογίες. Επίσης, εξετάζεται και η “εκτεταμένη τοπολογία” που αποτελείται από 100 near edge κόμβους, 20 far edge κόμβους και 4 cloud κόμβους. Η εκτεταμένη τοπολογία στοχεύει στην καλύτερη εξαγωγή συμπερασμάτων σε μεγαλύτερα προβλήματα και την επεκτασιμότητα και αποτελεσματικότητα του άπληστου αλγορίθμου.

Οι υπολογιστικοί πόροι του cloud αποτελούνται από ένα σύμπλεγμα 400 με 600 CPU πυρήνων ανά κόμβο και μέγεθος μνήμης RAM μεταξύ 600 και 800 GBs. Το far

edge επίπεδο περιλαμβάνει κόμβους από 40 μέχρι 80 πυρήνες CPU και 50 με 100 GB μνήμης RAM. Τέλος, το near edge επίπεδο αποτελείται από κόμβους με 2 με 4 πυρήνες και 2 με 8 GBs RAM.

Το οικονομικό κόστος εξυπηρέτησης αποτελείται από το κόστος δικτύου που αυξάνεται από το near edge στο cloud και το υπολογιστικό κόστος που αυξάνεται αντίστροφα. Η χρήση του cloud είναι η πιο οικονομική λύση, ενώ το near edge έχει τη μεγαλύτερη τιμή, αντικατοπτρίζοντας τις πολλές γεωγραφικές τοποθεσίες και τους περιορισμένους πόρους. Συγκεκριμένα, η χρήση ενός κόμβου του near edge για μία μικροϋπηρεσία κοστίζει μεταξύ 6 και 8 μονάδες κόστους (cost units c.u.) για τον υπολογισμό της και 0.1 με 0.2 c.u. για την αποστολή των δεδομένων της. Για το far edge το κόστος χρήσης είναι 3 με 4 c.u. και το κόστος του δικτύου είναι 0.4 με 0.6 c.u.. Τέλος, οι κόμβοι του cloud κοστίζουν 0.5 με 1 c.u. για τον υπολογισμό μίας μικροϋπηρεσίας και 1 με 1.2 για την αποστολή των δεδομένων. Οι υπολογιστικοί πόροι και τα κόστη ανά επίπεδο φαίνονται στον Πίνακας 6-1.

| Επίπεδο | Πλήθος κόμβων | | Πυρήνες CPU | Μνήμη RAM | Κόστος χρήσης | Κόστος δικτύου |
|---|---|---|---|---|---|---|
| | Βασική | Εκτεταμένη | | | | |
| Near Edge | 20 | 100 | [2,4] | [2,8] | [6,8] | [0.1,0.2] |
| Far Edge | 5 | 20 | [40,80] | [50,100] | [3,4] | [0.4,0.6] |
| Cloud | 2 | 4 | [400,600] | [600,800] | [0.5,1] | [1,1.2] |

*Πίνακας 6-1 Χαρακτηριστικά κόμβων σε διαφορετικά επίπεδα της υποδομής*

Το φόρτο εργασίας αποτελείται από cloud-native εφαρμογές, αποτελούμενες από διάφορες μικροϋπηρεσίες. Η κάθε εφαρμογή έχει από 1 μέχρι 7 μικροϋπηρεσίες που συνδέονται μεταξύ τους σαν αλυσίδα. Η αποδεκτή καθυστέρηση μίας εφαρμογής είναι μεταξύ 2 και 6 μονάδων χρόνου (time units t.u.), έτσι ώστε κάποιες να μπορούν να εξυπηρετηθούν μόνο από κόμβους του near edge ενώ κάποιες να μπορούν να εξυπηρετηθούν και από το cloud. Οι μικροϋπηρεσίες έχουν συγκεκριμένες απαιτήσεις σε πυρήνες CPU και μνήμη RAM από 0.5 μέχρι 1 πυρήνα και από 0.75 μέχρι 1.5 GBs αντίστοιχα. Το μέγεθος των δεδομένων εισόδου που στέλνει κάθε μικροϋπηρεσία είναι από 1 μέχρι 5 GB. Τέλος, η αποδεκτή καθυστέρηση μεταξύ δύο διαδοχικών μικροϋπηρεσίων μίας εφαρμογής είναι μεταξύ 0.5 και 5 t.u.. Τα χαρακτηριστικά των εφαρμογών φαίνονται στον Πίνακας 6-2.

| | |
|---|---|
| Πλήθος μικροϋπηρεσιών | [1,7] |
| Αποδεκτή καθυστέρηση | [2,6] |
| Απαίτηση CPU μικροϋπηρεσίας | [0.5,1] |
| Απαίτηση RAM μικροϋπηρεσίας | [0.75,1.5] |
| Δεδομένα εισόδου μικροϋπηρεσίας | [1,5] |
| Αποδεκτή καθυστέρηση διαδοχικών μικροϋπηρεσιών | [0.5,5] |

*Πίνακας 6-2 Χαρακτηριστικά εφαρμογών*

Οι συντεταγμένες των εφαρμογών και των κόμβων της υποδομής ορίζονται γύρω από ένα κεντρικό σημείο, έτσι ώστε η απόσταση των εφαρμογών με το σημείο να είναι μικρότερη από 1km, κάθε κόμβος του near edge να απέχει από 1 μέχρι 100km, του far edge μεταξύ 100 και 1.000km και του cloud πάνω από 1.000km. Η μέση καθυστέρηση που προκύπτει φαίνεται στον Πίνακας 6-3.

| | Εφαρμογή | Near Edge | Far Edge | Cloud |
|---|---|---|---|---|
| Near Edge | 0.20549 | 0.30126 | 2.68898 | 4.74026 |
| Far Edge | 2.18516 | 2.68898 | 3.96503 | 4.90835 |
| Cloud | 4.713856 | 4.74026 | 4.90835 | 6.66667 |

*Πίνακας 6-3 Μέση καθυστέρηση επικοινωνίας σε t.u.*

## 6.2 Αξιολόγηση Επίδοσης Άπληστου Αλγορίθμου

Ο άπληστος ευριστικός αλγόριθμος θυσιάζει την ακρίβεια του γραμμικού προγραμματισμού ως προς την εύρεση της βέλτιστης λύσης με αντάλλαγμα την μεγαλύτερη ταχύτητα, έτσι ο αλγόριθμος αξιολογείται με βάση τον χρόνο εκτέλεσης του και το συνολικό κόστος. Για την αξιολόγηση του αλγορίθμου χρησιμοποιείται η βασική τοπολογία για φόρτο εργασίας που κυμαίνεται από 50 μέχρι 300 cloud-native εφαρμογές. Τα βάρη w1 και w2 τίθενται ίσα με 0.2 και 0.8 αντίστοιχα, ενώ τα βάρη w3, w4 και w5 είναι ίσα με 0, δηλαδή στόχος είναι κυρίως η ελαχιστοποίηση της συνολικής καθυστέρησης με μικρή επίδραση του χρηματικού κόστους. Τα αποτελέσματα των δύο μηχανισμών φαίνονται στον Πίνακας 6-4.

| Αριθμός Εφαρμογών | MILP με εξαντλητική αναζήτηση | | Άπληστος ευριστικός αλγόριθμος | |
|---|---|---|---|---|
| | Αντικειμενικό Κόστος | Χρόνος Εκτέλεσης | Αντικειμενικό Κόστος | Χρόνος Εκτέλεσης |
| 50 | 244.86455037275 | 353 s | 272.44264553822 | 0.02 s |
| 100 | 590.18029784381 | 741 s | 654.88413396945 | 0.04 s |
| 150 | 902.72479884674 | 1345 s | 1001.2798964003 | 0.05 s |
| 200 | 1185.8626127278 | 2532 s | 1333.4554562464 | 0.06 s |
| 250 | 1461.8433418066 | 3605 s | 1622.2271199502 | 0.1 s |
| 300 | 1777.8701804162 | 4000 s | 1939.4050495973 | 0.12 s |

*Πίνακας 6-4 Σύγκριση συνολικού κόστους και χρόνου εκτέλεσης για w1=0.2 και w2=0.8*

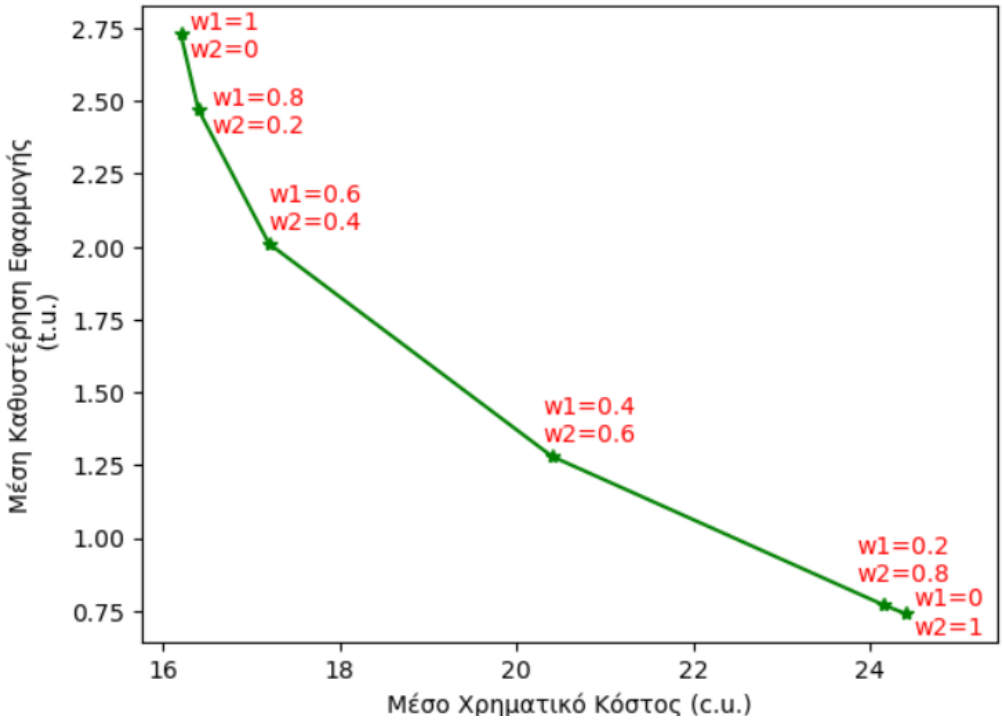
Ο ευριστικός αλγόριθμος παρήγαγε λύσεις που απέχουν περίπου 11% από τη βέλτιστη λύση του MILP. Σχετικά με τον χρόνο εκτέλεσης, ο ευριστικός αλγόριθμος εκτελέστηκε σε διάστημα μερικών milliseconds ακόμα για μεγαλύτερα φορτία εργασίας, ενώ ο χρόνος για την εκτέλεση του MILP αυξάνεται εκθετικά που τον καθιστά αδύνατο να ανταπεξέλθει σε μεγαλύτερο πλήθος εφαρμογών ή σε μεγαλύτερες τοπολογίες.

## 6.3 Αξιολόγηση Επιμέρους Τμημάτων Προβλήματος

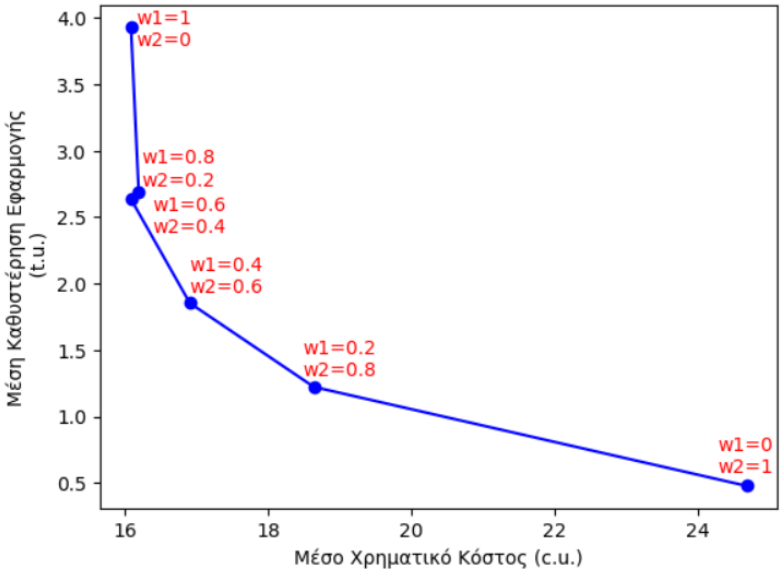
### 6.3.1 Αξιολόγηση Αλγορίθμου Αρχικής Κατανομή

Αρχικά, εξετάζεται η αποτελεσματικότητα του άπληστου ευριστικού αλγορίθμου αρχικής κατανομής. Το φόρτο εργασίας που εξετάζεται αποτελείται από 50 cloud-native εφαρμογές όπου όλες φτάνουν ταυτόχρονα και εκτελούνται για μία χρονική στιγμή. Η υποδομή περιλαμβάνει μόνο έναν aggregator ο οποίος είναι υπεύθυνος για όλους τους κόμβους. Ο βασικός στόχος είναι η ελαχιστοποίηση του οικονομικού κόστους και της καθυστέρησης επικοινωνίας και η επίδραση τους στην κατανομή στους κόμβους που προκύπτει για διαφορετικές τιμές των συντελεστών βάρους w1 και w2 με w1+w2=1. Τα βάρη w3,w4 και w5 είναι ίσα με 0 και δεν απασχολούν σε αυτό το σημείο το πρόβλημα.

Στην Εικόνα 6-1 φαίνεται το διάγραμμα Pareto για τον ευριστικό αλγόριθμο, όπου δείχνει τη σχέση μεταξύ των τιμών του μέσου χρηματικού κόστους και της μέσης καθυστέρησης μίας εφαρμογής για διάφορες συνθήκες βελτιστοποίησης. Όταν w1=1, δηλαδή η βελτιστοποίηση εστιάζει στο χρηματικό κόστος, η μέση καθυστέρηση είναι τρεις φορές μεγαλύτερη σε σχέση με την βέλτιστη τιμή της, ενώ αντίστοιχα όταν βελτιστοποιείται η καθυστέρηση (w2=1), το κόστος είναι 50% περισσότερο από τη βέλτιστη τιμή. Αντίστοιχα, αποτελέσματα φαίνονται στην Εικόνα 6-2 που δείχνει τα αποτελέσματα της βέλτιστης λύσης που προκύπτουν από το πρόβλημα γραμμικού προγραμματισμού.

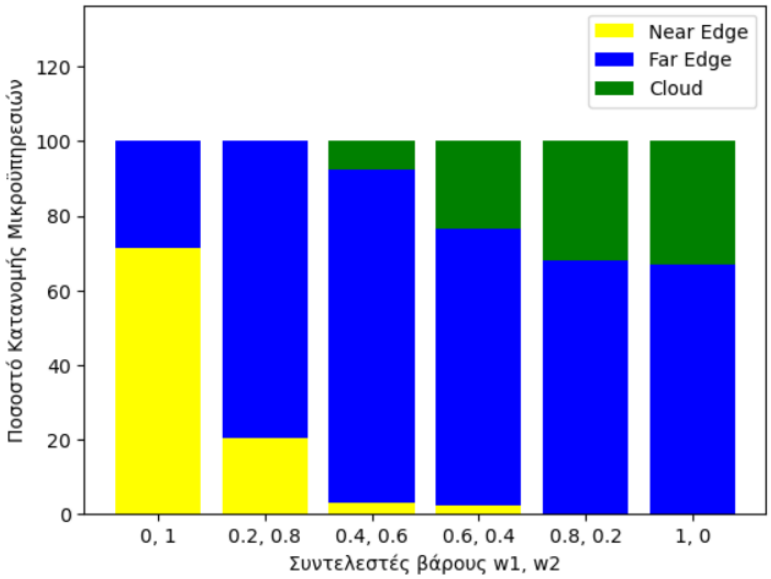



*Εικόνα 6-1Διάγραμμα Pareto για ευριστικό αλγόριθμο*



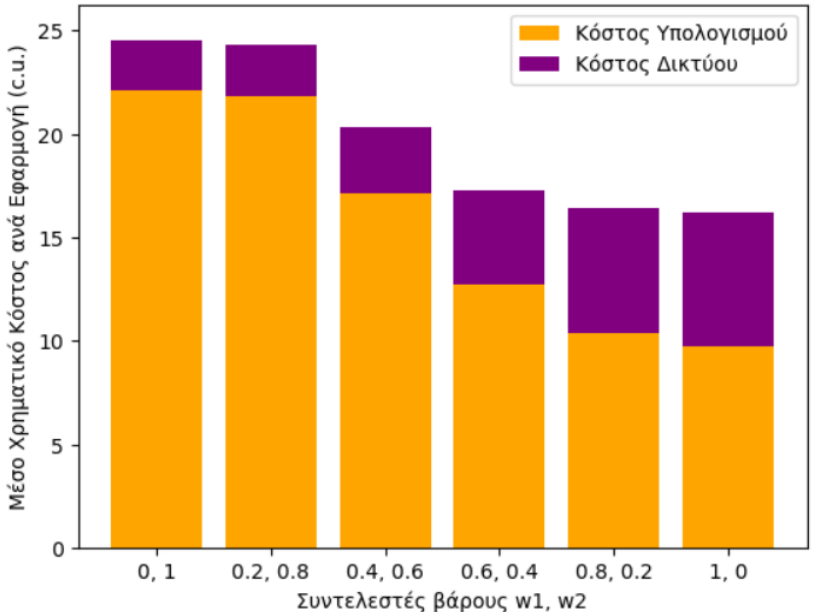

*Εικόνα 6-2 Διάγραμμα Pareto για MILP*

Στην Εικόνα 6-3 φαίνεται το ποσοστό μικροϋπηρεσιών που εξυπηρετούνται σε κάθε επίπεδο για διαφορετικές τιμές των συντελεστών βάρους w1 και w2. Στην περίπτωση που βελτιστοποιείται η καθυστέρηση των εφαρμογών προτιμάται το στρώμα του near edge που είναι κοντινότερα στις συσκευές, ωστόσο καθώς γεμίζει το near edge κάποιες μικροϋπηρεσίες εκτελούνται στο far edge. Το επίπεδο far edge φαίνεται να προτιμάται περισσότερο στις ενδιάμεσες περιπτώσεις, αφού οι κόμβοι του παρέχουν πιο ισορροπημένες τιμές κόστους χρήσης και καθυστέρησης. Τέλος, το cloud προτιμάται για την βελτιστοποίηση του κόστους, αφού παρέχει την φθηνότερη εξυπηρέτηση, ωστόσο ένα μεγάλο πλήθος εφαρμογών συνεχίζουν να χρησιμοποιούν κόμβους του far edge καθώς οι απομακρυσμένοι κόμβοι του cloud παραβιάζουν την αποδεκτή καθυστέρηση ορισμένων εφαρμογών.




*Εικόνα 6-3 Ποσοστό μικροϋπηρεσιών ανά επίπεδο για ευριστικό αλγόριθμο*

Η Εικόνα 6-4 παρουσιάζει τη συμβολή του δικτυακού και του λειτουργικού κόστους για τις διαφορετικές τιμές των συντελεστών βάρους. Όταν η βελτιστοποίηση εστιάζει στο κόστος προτιμώνται οι πόροι του far edge και του cloud, με το δικτυακό κόστος να συμβάλει κοντά στο 40% του συνολικού κόστους, καθώς το κόστος εξυπηρέτησης είναι χαμηλό ενώ το δικτυακό κόστος αυξάνεται λόγω της μεταφοράς δεδομένων σε μεγαλύτερη απόσταση. Αντίθετα, όταν ο στόχος είναι η ελαχιστοποίηση της καθυστέρησης και αξιοποιούνται κυρίως πόροι του near edge, το κόστος επεξεργασίας αποτελεί τον κύριο παράγοντα του συνολικού χρηματικού κόστους, με το δικτυακό κόστος να αντιστοιχεί μόλις στο 10% του συνολικού κόστους.

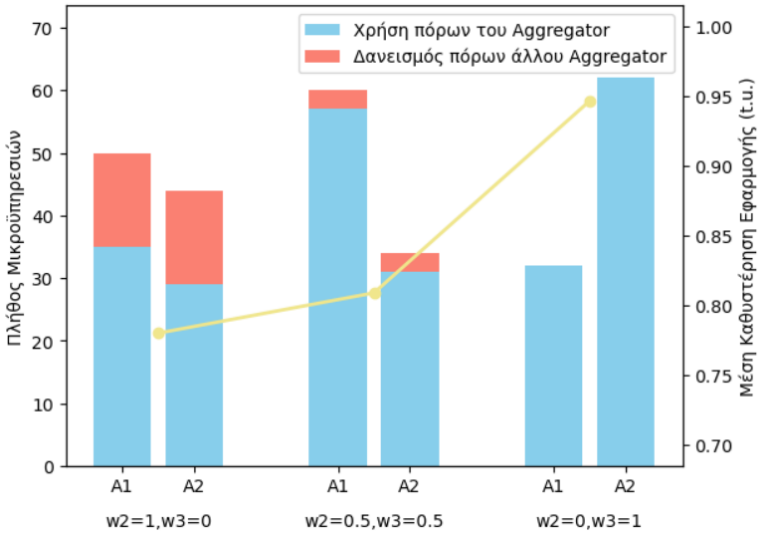



*Εικόνα 6-4 Κόστος υπολογισμού και δικτύου για διαφορετικούς συντελεστές βάρους για ευριστικό αλγόριθμο*

### 6.3.2 Επίδραση Aggregators

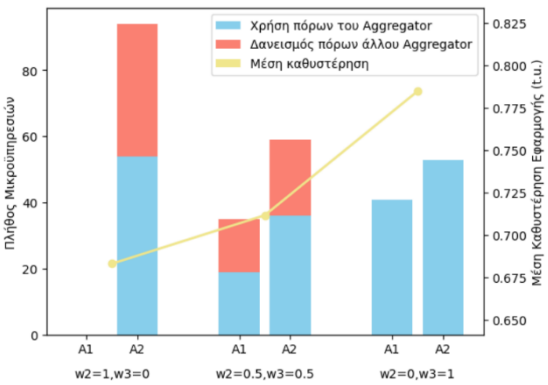
Στην υποδομή δρουν Aggregators, όπου οι συνεργασίες μεταξύ τους επηρεάζουν την κατανομή των μικροϋπηρεσιών στους υπολογιστικούς κόμβους και την τελική τιμή της συνάρτησης βελτιστοποίησης. Το φόρτο εργασίας που εξετάζεται σε αυτό το σημείο αποτελείται από 40 cloud-native εφαρμογές που επαρκούν για να γεμίσουν το near edge χωρίς να απαιτείται μεγάλη χρήση των υψηλότερων επιπέδων. Η ανάγκη για δανεισμό υπολογιστικών πόρων και η επίδραση της στον τελικό στόχο φαίνεται περισσότερο στην περίπτωση όπου οι πόροι ενός Aggregator δεν επαρκούν για την καλύτερη εξυπηρέτηση των εφαρμογών και χρειάζεται να αποκτήσουν άμεση πρόσβαση σε περισσότερους πόρους. Για αυτό, εξετάζεται η επίδραση των δανεισμών σε συνδυασμό με την ελαχιστοποίηση της συνολικής καθυστέρησης για διάφορες τιμές των συντελεστών βάρους w2 και w3. Τα βάρη w1,w4 και w5 είναι ίσα με 0 και δεν απασχολούν σε αυτό το σημείο το πρόβλημα.

Αρχικά, εξετάζεται το σενάριο που υπάρχουν δύο Aggregators οι οποίοι συνεργάζονται μεταξύ τους. Στην Εικόνα 6-5 φαίνεται η κατανομή των μικροϋπηρεσιών στους δύο Aggregators για διαφορετικές τιμές των συντελεστών βάρους w2 για την συνολική καθυστέρηση των εφαρμογών και w3 για τον δανεισμό πόρων άλλου Aggregator. Με μπλε χρώμα φαίνεται το πλήθος των μικροϋπηρεσιών που έχει αναλάβει ο κάθε Aggregator και τις εκτελεί σε δικούς του κόμβους, ενώ με κόκκινο χρώμα φαίνονται οι μικροϋπηρεσίες όπου δανείζεται κόμβους διαφορετικού Aggregator. Όταν w3=0 η χρήση πόρων άλλου Aggregator δεν επηρεάζει την συνάρτηση βελτιστοποίησης και παρατηρείται χρήση περίπου 30% πόρων διαφορετικού Aggregator, ενώ επιτυγχάνεται η ελάχιστη μέση καθυστέρηση εφαρμογής (κίτρινη γραμμή). Στην περίπτωση που w2=w3=0.5, φαίνεται λιγότερος αριθμός δανεισμών περίπου 5% με μικρή αύξηση της μέσης καθυστέρησης. Όταν w3=1 τότε στόχος είναι να ελαχιστοποιηθεί η χρήση κόμβων διαφορετικού Aggregator, όπου οδηγεί σε αύξηση της καθυστέρησης καθώς μπορεί να υπάρχουν πιο κοντινοί κόμβοι για κάποιες μικροϋπηρεσίες αλλά ανήκουν στον άλλον Aggregator.



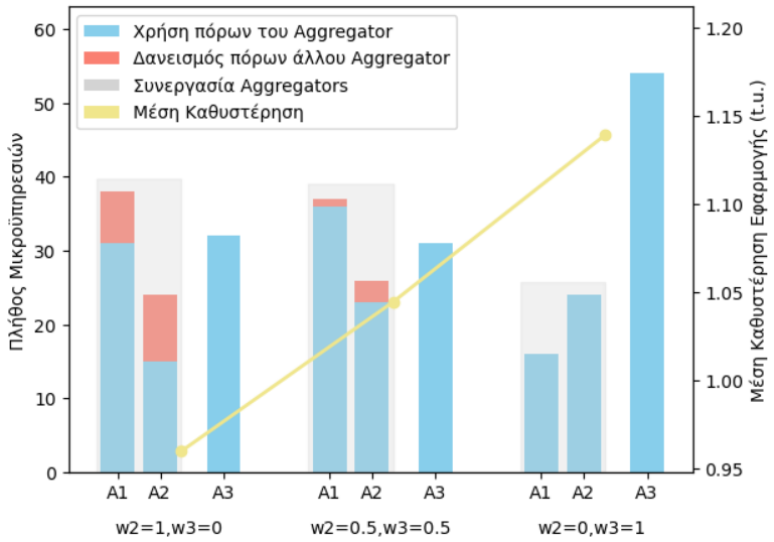

*Εικόνα 6-5 Επίδραση δανεισμού υπολογιστικών πόρων μεταξύ 2 Aggregators που συνεργάζονται*

Στην Εικόνα 6-6 φαίνονται τα αντίστοιχα αποτελέσματα για το πρόβλημα γραμμικού προγραμματισμού. Στην περίπτωση που w3 είναι κοντά στο 1 και w2 είναι πολύ μικρό, ώστε να αποφεύγονται οι δανεισμοί αλλά να ελαχιστοποιείται η καθυστέρηση, παρατηρείται αύξηση περίπου 18% της μέσης καθυστέρησης των εφαρμογών συγκριτικά με την περίπτωση όπου γίνονται ελεύθερα δανεισμοί μεταξύ των Aggregators.




*Εικόνα 6-6 Επίδραση δανεισμού υπολογιστικών πόρων για MILP*

Στη συνέχεια, εξετάζεται το σενάριο που υπάρχουν τρεις Aggregators από τους οποίους οι δύο συνεργάζονται μεταξύ τους, ενώ ο τρίτος δεν μπορεί να συνεργαστεί με κανέναν από τους άλλους δύο. Στην Εικόνα 6-7 παρατηρείται αντίστοιχη αύξηση της μέσης καθυστέρησης όσο ελαχιστοποιούνται οι δανεισμοί περίπου 20%. Επίσης, παρατηρείται συνολική αύξηση της καθυστέρησης από ότι στην περίπτωση των δύο Aggregators, αυτό οφείλεται στην δημιουργία δύο ξεχωριστών συνεργασιών Aggregators (A1 και A2, A3) που οδηγεί σε μεγαλύτερη χρήση του far edge λόγω των λιγότερων near edge κόμβων εντός της κάθε συνεργασίας.




*Εικόνα 6-7 Επίδραση δανεισμού υπολογιστικών πόρων μεταξύ 3 Aggregators που οι δύο συνεργάζονται μεταξύ τους*

### 6.3.3 Αξιολόγηση Αλγορίθμου Ανακατανομής

Για την αξιολόγηση του αλγορίθμου ανακατανομής εξετάζεται ένα φόρτο εργασίας από 60 cloud-native εφαρμογές για διάρκεια 50 χρονικών στιγμών η καθεμία που αρχίζουν και τελειώνουν ταυτόχρονα. Η κίνηση μίας εφαρμογής κατά την διάρκεια που είναι ενεργή ενδέχεται να προκαλέσει παραβίαση του ορίου καθυστέρησης της και να απαιτείσαι ανακατανομή σε νέους κόμβους.

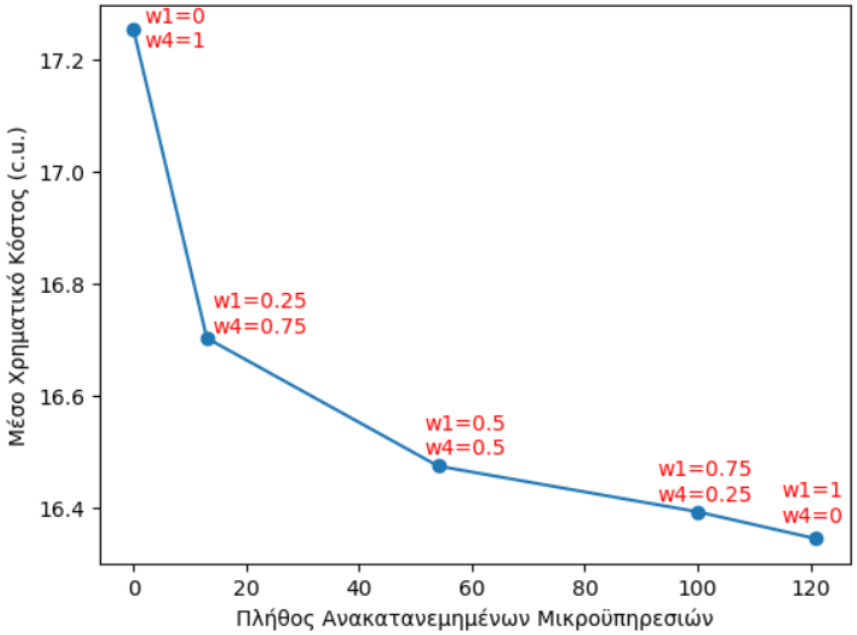
Αρχικά, στον Πίνακας 6-5 φαίνεται το πλήθος των ανακατανομών που απαιτούνται για διαφορετικές τιμές των συντελεστών βάρους w1 και w2 όταν δρα στην υποδομή ένας Aggregator. Στην περίπτωση που στόχος είναι η ελαχιστοποίηση της καθυστέρησης των εφαρμογών (w2=1) τότε οι εφαρμογές χρησιμοποιούν εξαρχής κοντινούς κόμβους, με αποτέλεσμα με την κίνηση τους να μην απομακρύνονται αρκετά ώστε να απαιτούνται ανακατανομές. Αντίθετα, στην περίπτωση που βελτιστοποιείται το κόστος (w1=1) χρησιμοποιείται περισσότερο το far edge και το cloud που βρίσκονται σε μεγαλύτερη απόσταση και κατά την διάρκεια της κίνησης τους οι συσκευές απομακρύνονται περισσότερο και οδηγούν σε περισσότερες ανακατανομές. Επίσης, παρατηρείται ότι ο αλγόριθμος ανακατανομής διατηρεί

περίπου το 25% των μικροϋπηρεσιών σε ίδιος κόμβους αντί να δρομολογεί εκ νέου όλες τις μικροϋπηρεσίες.

| | w1=1<br>w2=0 | w1=0.5<br>w2=0.5 | w1=0<br>w2=1 |
|---|---|---|---|
| Πλήθος Εφαρμογών | 10 | 5 | 1 |
| Πλήθος Μικροϋπηρεσιών που άλλαξαν κόμβο | 36 | 9 | 4 |
| Σύνολο Μικροϋπηρεσιών | 47 | 12 | 4 |

*Πίνακας 6-5 Πλήθος ανακατανομών για 1 Aggregator*

Στην Εικόνα 6-8 φαίνεται το πλήθος των ανακατανομών μικροϋπηρεσιών για το αντίστοιχο φόρτο εργασίας σε σύγκριση με το κόστος για το πρόβλημα γραμμικού προγραμματισμού. Το MILP τροποποιεί τους κόμβους των μικροϋπηρεσίων με στόχο να βελτιστοποιήσει το κόστος χωρίς να γίνεται απαραίτητα ανακατανομή μόνο όταν παραβιάζεται το όριο καθυστέρησης των εφαρμογών, όπως συμβαίνει στον ευριστικό αλγόριθμο. Από την εικόνα συμπεραίνεται η ανάγκη για ανακατανομές κατά την διάρκεια της κίνησης, καθώς ακόμα και στην βέλτιστη λύση, όταν η κίνηση είναι γνωστή εκ των προτέρων, παρατηρείται αύξηση του κόστους για να αποφευχθούν οι ανακατανομές.




*Εικόνα 6-8 Ανακατανομές μικροϋπηρεσιών για 1 Aggregator για MILP*

Στη συνέχεια, εξετάζεται ο αλγορίθμου ανακατανομής στην περίπτωση που δρουν περισσότεροι Aggregators πάνω στην υποδομή. Στον Πίνακας 6-6 φαίνονται τα αποτελέσματα των ανακατανομών για το ίδιο φόρτο εργασίας όταν υπάρχουν 3 Aggregators, όπου οι δύο συνεργάζονται μεταξύ τους ενώ ο τρίτος όχι, στην περίπτωση που βελτιστοποιείται το κόστος (w1=1). Στην περίπτωση που προτιμάται το Hard Handover παρατηρείται μικρότερο μέσο κόστος εφαρμογών, καθώς δρομολογείται εκ νέου ολόκληρη η εφαρμογή προσπαθώντας να

βελτιστοποιήσει το κόστος, ενώ στην περίπτωση του Soft Handover στόχος είναι η ελαχιστοποίηση των μικροϋπηρεσιών που αλλάζουν κόμβο με αποτέλεσμα να αυξάνεται το κόστος. Επίσης, στην περίπτωση όπου w4=0.2 και w5=0.8 φαίνεται ότι παρόλο που το πλήθος των Soft Handovers είναι μεγαλύτερο, τα Hard Handovers απαιτούνται για τις εφαρμογές με πολλές μικροϋπηρεσίες.

| | w4=1<br>w5=0 | w4=0.2<br>w5=0.8 | w4=0<br>w5=1 |
|---|---|---|---|
| Μέσο Κόστος Εφαρμογής | 16.70464 | 16.83438 | 16.86503 |
| Πλήθος Εφαρμογών | 8 | 8 | 8 |
| Hard Handovers | 8 | 3 | 0 |
| Soft Handovers | 0 | 5 | 8 |
| Πλήθος Μικροϋπηρεσιών Soft-Hard | 0 - 32 | 13 - 19 | 32 - 0 |

*Πίνακας 6-6 Πλήθος ανακατανομών για 3 Aggregators που οι 2 συνεργάζονται για w1=1*

Στον Πίνακας 6-7 φαίνονται τα αντίστοιχα αποτελέσματα για το πρόβλημα γραμμικού προγραμματισμού. Παρατηρείται μία μικρή διαφορά στο μέσο κόστος εφαρμογής, όταν προτιμάται το Hard Handover το MILP μπορεί να αλλάζει ολόκληρο Aggregator ώστε να διατηρεί την βέλτιστη κατανομή στους κόμβους κάθε χρονική στιγμή, ενώ όταν προτιμάται το Soft Handover αναγκάζεται να παραμένει στους κόμβους του ίδιου Aggregator όπου μπορεί να μην πετυχαίνει το βέλτιστο κόστος.

| | w4=0.5<br>w5=0 | w4=0.1<br>w5=0.4 | w4=0<br>w5=0.5 |
|---|---|---|---|
| Μέσο Κόστος Εφαρμογής | 16.44865 | 16.45774 | 16.46889 |
| Πλήθος Εφαρμογών | 315 | 54 | 317 |
| Hard Handovers | 300 | 12 | 17 |
| Soft Handovers | 15 | 42 | 300 |
| Πλήθος Μικροϋπηρεσιών Soft-Hard | 52 - 1260 | 131 - 43 | 1260 - 72 |

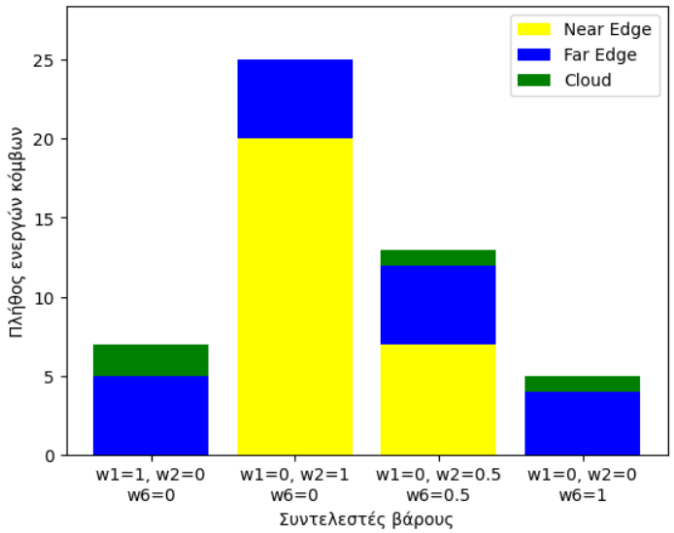
*Πίνακας 6-7 Πλήθος ανακατανομών για 3 Aggregators που οι 2 συνεργάζονται για w1=0.5 για MILP*

### 6.3.4 Επίδραση Κριτηρίου Ενεργών Κόμβων

Τέλος, εξετάζεται το κριτήριο ενεργών κόμβου του προβλήματος γραμμικού προγραμματισμού. Το φόρτο εργασίας αποτελείται από 100 cloud-native εφαρμογές που είναι ενεργές για μία χρονική στιγμή. Στην Εικόνα 6-9 παρουσιάζονται τα αποτελέσματα του MILP ως προς τον αριθμό ενεργών κόμβων σε τέσσερις διαφορετικές περιπτώσεις βελτιστοποίησης.

Στην πρώτη περίπτωση, όπου στόχος είναι αποκλειστικά η ελαχιστοποίηση του κόστους των εφαρμογών, το βέλτιστο αποτέλεσμα επιτυγχάνεται με την αξιοποίηση όλων των κόμβων του far edge και του cloud. Στην δεύτερη περίπτωση, με στόχο την ελαχιστοποίηση της καθυστέρησης των εφαρμογών, παρατηρείται ότι χρησιμοποιούνται όλοι οι κόμβοι του near edge και λόγω του μεγάλου αριθμού εφαρμογών χρησιμοποιούνται και κόμβοι του far edge. Στην

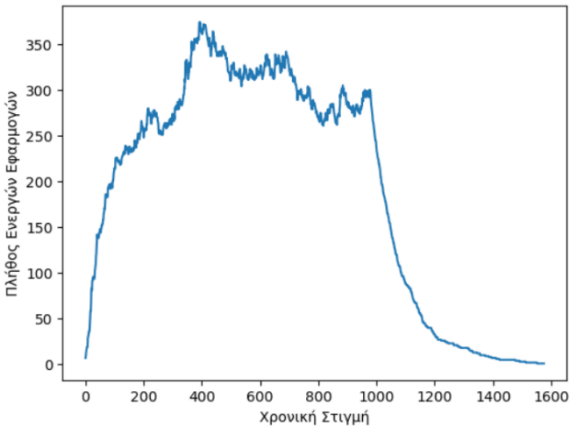
τρίτη περίπτωση, που ο στόχος συνδυάζει την ελαχιστοποίηση της καθυστέρησης των εφαρμογών και του πλήθους των ενεργών κόμβων, το far edge εξακολουθεί να χρησιμοποιείται, ωστόσο χρησιμοποιούνται λιγότεροι κόμβοι του near edge ενώ ενεργοποιείται και ο ένας κόμβος του cloud. Τέλος, στην περίπτωση που στόχος είναι μόνο η ελαχιστοποίηση των ενεργών κόμβων, τότε παρατηρείται ότι χρησιμοποιούνται λιγότεροι κόμβοι από ότι στις άλλες περιπτώσεις, συγκεκριμένα χρησιμοποιείται ένας κόμβος του cloud για τις περισσότερες εφαρμογές, ενώ για τις εφαρμογές που δεν μπορούν να εξυπηρετηθούν εντός του αποδεκτού ορίου καθυστέρησης χρησιμοποιούνται επιπλέον 4 κόμβοι του far edge.




*Εικόνα 6-9 Ενεργοί κόμβοι για MILP*

## 6.4 Αξιολόγηση Προγράμματος Σε Κανονική Λειτουργία

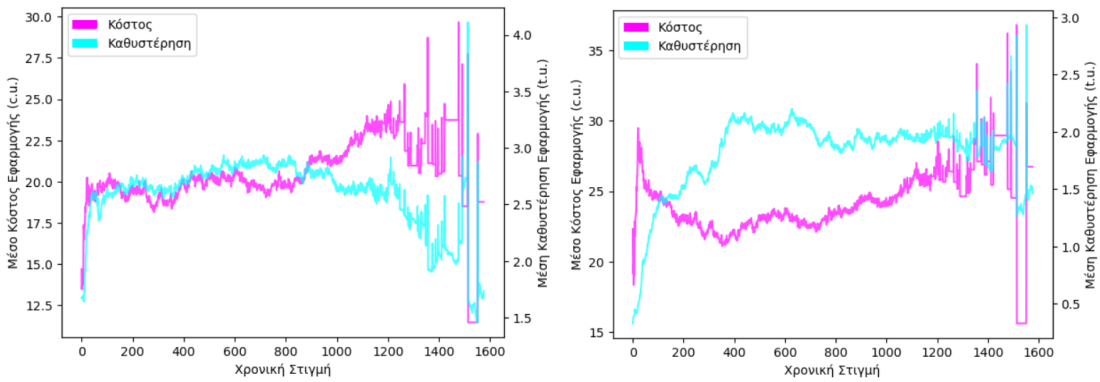
Σε αυτήν την ενότητα αξιολογείται η επεκτασιμότητα και η αποτελεσματικότητα του ευριστικού αλγορίθμου υπό συνθήκες κανονικής εκτέλεσης. Η υποδομή που χρησιμοποιείται είναι η εκτεταμένη τοπολογία, στην οποία δρουν 5 Aggregators, που μπορούν να συνεργάζονται ανά δύο. Η υποδομή καλείται να εξυπηρετήσει ένα σύνολο από 3000 cloud-native εφαρμογές. Η άφιξη των εφαρμογών ακολουθεί διαδικασία Poisson και αναπαρίσταται μέσω της εκθετικής κατανομής με παράμετρο λ ίση με 2.5, που αντιστοιχεί στο ποσοστό αφίξεων ανά μονάδα χρόνου. Η κάθε εφαρμογή έχει πιθανότητα 1% να ολοκληρωθεί για κάθε στιγμή μετά την άφιξη της. Στην Εικόνα 6-10 παρουσιάζεται το πλήθος των εφαρμογών που εκτελούνται ανά χρονική στιγμή στους κόμβους της υποδομής.




*Εικόνα 6-10 Ενεργές εφαρμογές ανά χρονική στιγμή*

Η ανάλυση εξετάζει δύο διαφορετικά σενάρια: το πρώτο επικεντρώνεται στο χρηματικό κόστος, ενώ το δεύτερο στην καθυστέρηση εκτέλεσης των εφαρμογών. Συγκεκριμένα, οι συντελεστές βάρους ορίζονται ως w1=0.5, w2=0.1 στην πρώτη περίπτωση και w1=0.1, w2=0.5 στην δεύτερη. Επιπλέον, τα υπόλοιπα κριτήρια διατηρούν τα ίδια βάρη και στα δύο σενάρια: η δυνατότητα δανεισμού υπολογιστικών πόρων έχει βάρος w3=0.2 και η ενεργοποίηση νέου κόμβου έχει βάρος w6=0.2. Τέλος, τα Handovers έχουν βάρη w4=0.2 και w5=0.8. δηλαδή προτιμάται το Soft Handover όταν απαιτείται μετακίνηση έως και τεσσάρων μικροϋπηρεσιών σε νέους κόμβους, αλλιώς προτιμάτε το Hard Handover.

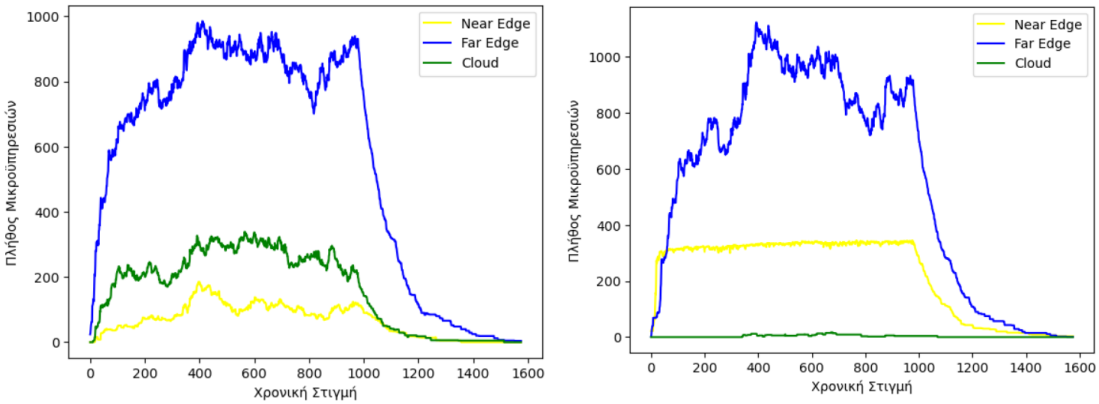
Στην Εικόνα 6-11 φαίνονται το μέσο χρηματικό κόστος και η μέση καθυστέρηση στα δύο σενάρια. Στην πρώτη περίπτωση, το μέσο κόστος και η καθυστέρηση παραμένουν σταθερά κατά την περισσότερη διάρκεια της προσομοίωσης, καθώς η πλειοψηφία των μικροϋπηρεσιών εκτελούνται σε κόμβους του far edge και του cloud, όπως φαίνεται στην Εικόνα 6-12. Στην δεύτερη περίπτωση παρατηρείται μεγαλύτερη διακύμανση των τιμών, αυτό οφείλεται στις περιορισμένες υπολογιστικές δυνατότητες του near edge, οπότε απαιτείται η χρήση ανώτερων επιπέδων για να καλύψουν τις ανάγκες των εφαρμογών.




(α) Σενάριο κόστους (β) Σενάριο καθυστέρησης

*Εικόνα 6-11 Μέσο κόστος και καθυστέρηση εφαρμογών*

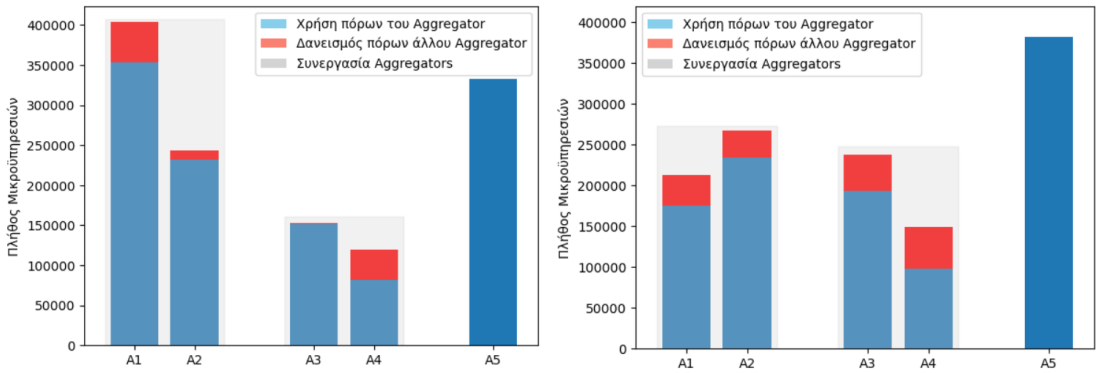
Στην Εικόνα 6-12 φαίνεται το πλήθος των μικροϋπηρεσιών και το ποσοστό πληρότητας των τριών επιπέδων αντίστοιχα. Στο σενάριο του κόστους το cloud χρησιμοποιείται, για τις εφαρμογές που είναι αποδεκτή η καθυστέρηση, ενώ φτάνει μέχρι 10% τις συνολικής πληρότητας των πόρων του, αντίθετα στο σενάριο καθυστέρησης χρησιμοποιείται αποκλειστικά σε περιπτώσεις που δεν μπορεί να χρησιμοποιηθεί άλλος κόμβος. Το far edge επίπεδο χρησιμοποιείται και στις δύο περιπτώσεις σε μεγάλο βαθμό φτάνοντας 70% πληρότητα και στις δύο, καθώς αποτελεί την μέση λύση και για τα δύο προβλήματα. Τέλος, το near edge χρησιμοποιείται πλήρως στο σενάριο καθυστέρησης, με το ποσοστό πληρότητας του συνεχώς κοντά στο 100%, ενώ στην περίπτωση του κόστους χρησιμοποιείται μόνο όταν παραβιάζεται το όριο καθυστέρησης και δεν υπάρχει διαθέσιμος κόμβος του far edge σε αρκετά κοντινή απόσταση για να λύσει το πρόβλημα.




(α) Σενάριο κόστους (β) Σενάριο καθυστέρησης

*Εικόνα 6-12 Πλήθος μικροϋπηρεσιών ανά επίπεδο*

Στην Εικόνα 6-13 φαίνεται η κατανομή των μικροϋπηρεσιών στους Aggregators και οι δανεισμοί που χρειάστηκαν μεταξύ τους. Στο σενάριο καθυστέρησης παρατηρούνται περίπου 67% περισσότεροι δανεισμοί μεταξύ των Aggregators από ότι στο σενάριο κόστους (99837 έναντι 167128 δανεισμών), καθώς οι κόμβοι near edge που χρησιμοποιούνται δεν μπορούν να εκτελέσουν όλες τις μικροϋπηρεσίες οι ίδιοι και συχνά είναι προτιμότερο να δανειστούν υπολογιστικούς πόρους από κόμβους ενός συνεργάτη Aggregator.




(α) Σενάριο κόστους (β) Σενάριο καθυστέρησης

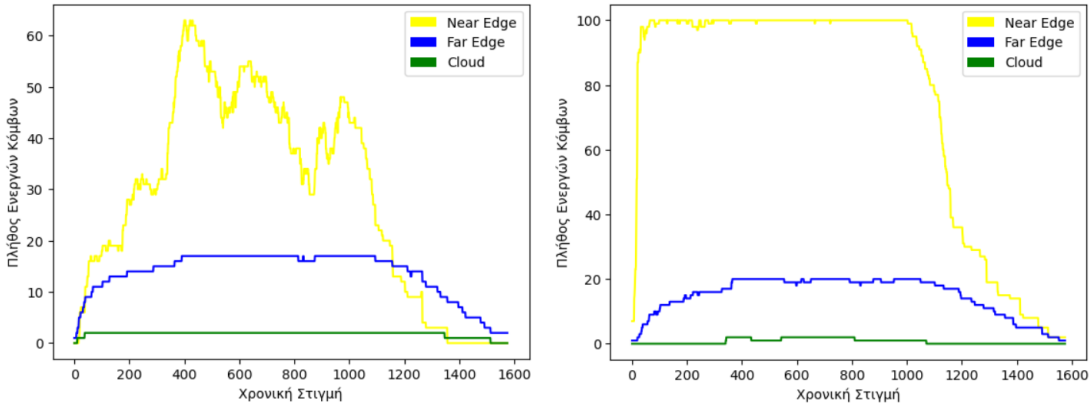
*Εικόνα 6-13 Πλήθος μικροϋπηρεσιών που εξυπηρέτησαν οι Aggregators*

Στον Πίνακας 6-8 φαίνονται η μέση τιμή του κόστους και της καθυστέρησης των εφαρμογών και οι ανακατανομές που χρειάστηκαν και στα δύο σενάρια. Αρχικά, φαίνεται η διαφορά στη μέση τιμή των εφαρμογών ανάλογα με το κριτήριο βελτιστοποίησης, στην περίπτωση του κόστους το μέσο χρηματικό κόστος ανά εφαρμογή είναι 13% λιγότερο σε σύγκριση με την περίπτωση καθυστέρησης, ενώ η καθυστέρηση είναι 46% μεγαλύτερη. Επίσης, παρατηρείται ότι στην περίπτωση του κόστους το όριο καθυστέρησης των εφαρμογών παραβιάζεται πάνω από δύο φορές πιο συχνά, καθώς προτιμώνται αρχικά κόμβοι σε μεγαλύτερη απόσταση από τις συσκευές και με την κίνηση τους αυξάνονται ακόμα περισσότερο με αποτέλεσμα να ξεπερνιέται το όριο καθυστέρησης.

| | w1=0.5<br>w2=0.1 | w1=0.1<br>w2=0.5 |
|---|---|---|
| Μέσο Κόστος Εφαρμογής | 20.09001 | 23.30894 |
| Μέση Καθυστέρηση Εφαρμογής | 2.73359 | 1.86655 |
| Πλήθος Εφαρμογών που χρειάστηκαν ανακατανομή | 469 | 228 |
| Hard Handovers | 156 | 84 |
| Soft Handovers | 313 | 144 |
| Πλήθος Μικροϋπηρεσιών Soft-Hard Handover | 558 - 979 | 258 - 492 |

*Πίνακας 6-8 Πλήθος ανακατανομών*

Στην Εικόνα 6-14 παρουσιάζεται η εξέλιξη του πλήθους ενεργών κόμβων σε κάθε χρονική στιγμή. Στο σενάριο όπου ο στόχος βελτιστοποίησης είναι η ελαχιστοποίηση της καθυστέρησης, παρατηρείται ότι ενεργοποιούνται όλοι οι κόμβοι του near edge. Η ενεργοποίησή τους είναι αναπόφευκτη προκειμένου να επιτευχθούν οι αντίστοιχες τιμές καθυστέρησης, γεγονός που οδηγεί σε αυξημένη διασπορά των εφαρμογών και κατά συνέπεια σε μεγαλύτερο αριθμό διαφορετικών κόμβων που απαιτούνται για την εξυπηρέτησή τους. Συγκεκριμένα, στη χρονική στιγμή κατά την οποία παρατηρείται το μέγιστο πλήθος ενεργών κόμβων, το σενάριο καθυστέρησης απαιτεί 48% περισσότερους διαφορετικούς κόμβους σε σχέση με το σενάριο κόστους (122 έναντι 82 ενεργών κόμβων). Επιπλέον, σε μέσο όρο, ο αριθμός ενεργών κόμβων είναι κατά 106% υψηλότερος στο σενάριο καθυστέρησης, με 90,16 ενεργούς κόμβους σε σύγκριση με 43,71 στο σενάριο κόστους. Τα αποτελέσματα αυτά καταδεικνύουν ότι η βελτιστοποίηση ως προς την καθυστέρηση συνεπάγεται σημαντικά αυξημένη χρήση υπολογιστικών κόμβων, καθώς η κατανομή των εφαρμογών απαιτεί ευρύτερη ενεργοποίηση της υποδομής για την επίτευξη χαμηλότερων τιμών καθυστέρησης.




(α) Σενάριο κόστους (β) Σενάριο καθυστέρησης

*Εικόνα 6-14 Ενεργοί κόμβοι*

# 7 Συμπεράσματα

Η παρούσα διπλωματική εργασία εστίασε στην ανάπτυξη αποδοτικών αλγορίθμων για την προσέγγιση της βέλτιστης κατανομής cloud-native εφαρμογών από κινητές συσκευές σε edge-cloud περιβάλλον. Η εργασία επιχειρεί να λύσει το πρόβλημα του κατακερματισμού των πόρων στο edge επίπεδο, σε μία κατανεμημένη και ιεραρχική υποδομή που αποτελείται από μεγάλο πλήθος υπολογιστικά περιορισμένων κόμβων στα επίπεδα near και far edge, καθώς και περιορισμένο αριθμό ισχυρών κόμβων στο cloud επίπεδο με εικονικά απεριόριστους υπολογιστικούς πόρους.

Στο πλαίσιο αυτό, μελετήθηκε η κατανομή των μικροϋπηρεσιών των εφαρμογών στους ετερογενείς κόμβους της υποδομής, με στόχο την βελτιστοποίηση διαφορετικών κριτηρίων, όπως το κόστος εξυπηρέτησης τους και την ποιότητα εξυπηρέτησης (Quality of Service - QoS). Επίσης, εξετάστηκε η κινητικότητα των συσκευών και η συνεπαγόμενη ανάγκη μετακίνησης των μικροϋπηρεσιών σε νέους κόμβους, προκειμένου να διατηρηθεί η αποδεκτή ποιότητα εξυπηρέτησης. Η εργασία προβλέπει μηχανισμούς δυναμικής ανακατανομής μικροϋπηρεσιών που διασφαλίζουν την αδιάλειπτη λειτουργία των εφαρμογών, διατηρώντας το επίπεδο της προσφερόμενης υπηρεσίας εντός αποδεκτών ορίων.

Επιπλέον, η εργασία εξετάζει την ύπαρξη διαφορετικών Aggregators, οι οποίοι ελέγχουν επιμέρους ομάδες κόμβων της υποδομής, και την επίδραση της συνεργασίας μεταξύ τους στη βελτίωση της κατανομής των μικροϋπηρεσιών και στη συνεχή ικανοποίηση των απαιτήσεων των εφαρμογών. Αναδεικνύεται η δυνατότητα δανεισμού υπολογιστικών πόρων μεταξύ Aggregators και η σημασία αυτής της συνεργασίας για τη διατήρηση υψηλής ποιότητας υπηρεσίας. Παράλληλα, διερευνάται η ανακατανομή μικροϋπηρεσιών μεταξύ κόμβων διαφορετικών Aggregators, με διάκριση μεταξύ Soft και Hard Handovers ανάλογα με το επίπεδο συνεργασίας μεταξύ τους.

Στο πλαίσιο της διπλωματικής εργασίας αναπτύχθηκε ένας ευριστικός αλγόριθμος, ο οποίος μετασχηματίζει ένα σύνθετο πρόβλημα γραμμικού προγραμματισμού σε υπολογιστικά αποδοτική μορφή. Ο αλγόριθμος επιτυγχάνει την κατανομή των εφαρμογών στους κόμβους τις υποδομής, προσεγγίζοντας την βέλτιστη λύση που προκύπτει από τον γραμμικό προγραμματισμό, αλλά με σημαντικά μειωμένο χρόνο και ικανοποιητικά αποτελέσματα ως προς την ελαχιστοποίηση τόσο του χρηματικού κόστους όσο και της καθυστέρησης επικοινωνίας. Επιπροσθέτως, διατηρεί την ποιότητα εξυπηρέτησης σε αποδεκτά επίπεδα κατά την διάρκεια της κίνησης των συσκευών, αποτρέποντας πιθανές απρόσκοπτες διακοπές της υπηρεσίας και υπερβολικές καθυστερήσεις.

Τέλος, αξιολογήθηκε η επεκτασιμότητα του αλγορίθμου σε σενάριο με μεγαλύτερο αριθμό κόμβων και εφαρμογών, αποδεικνύοντας την αποτελεσματικότητα του σε μεγάλης κλίμακας προβλήματα, όπου ο γραμμικός προγραμματισμός αδυνατεί να υπολογίσει σε ρεαλιστικούς χρόνους.

# Βιβλιογραφία


[1] Implementing edge computing for V2X use cases in automotive, by Bertrand Boisseau, https://ubuntu.com/blog/implementing-edge-computing-for-v2x-use-cases-in-automotive

[2] Abdullah MFA, Yogarayan S, Abdul Razak SF et al. Edge computing for Vehicle to Everything: a short review. F1000Research 2023, 10:1104, doi:10.12688/f1000research.73269.4

[3] "What Is Cloud Computing?", https://www.pcmag.com/how-to/what-is-cloud-computing

[4] "What is Cloud Computing", https://www.ibm.com/cloud/learn/cloud-computing

[5] Qian, L., Luo, Z., Du, Y., Guo, L. (2009). Cloud Computing: An Overview. In: Jaatun, M.G., Zhao, G., Rong, C. (eds) Cloud Computing. CloudCom 2009. Lecture Notes in Computer Science, vol 5931. Springer, Berlin, Heidelberg, doi:10.1007/978-3-642-10665-1_63

[6] Alghofaili, Yara, Albatul Albattah, Noura Alrajeh, Murad A. Rassam, and Bander Ali Saleh Al-rimy. 2021. "Secure Cloud Infrastructure: A Survey on Issues, Current Solutions, and Open Challenges" *Applied Sciences* 11, no. 19: 9005, doi:10.3390/app11199005

[7] Mell, P.; Grance, T. The NIST Definition of Cloud Computing; Nation-al Institute of Standards and Technology: Gaithersburg, MD, USA, 2011, https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf

[8] A. Gajbhiye and K. M. P. Shrivastva, "Cloud computing: Need, enabling technology, architecture, advantages and challenges," 2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence),Noida,India,2014,pp.1-7, doi:10.1109/CONFLUENCE.2014.6949224

[9] "Edge Computing VS Cloud Computing: Differences, Benefits, and Best Practices", https://www.orientsoftware.com/blog/edge-computing-vs-cloud-computing/

[10] K. Cao, Y. Liu, G. Meng and Q. Sun, "An Overview on Edge Computing Research," in IEEE Access, vol. 8, pp. 85714-85728, 2020, doi:10.1109/ACCESS.2020.2991734

[11] W. Shi and S. Dustdar, "The Promise of Edge Computing," in Computer, vol. 49, no. 5, pp. 78-81, May 2016, doi:10.1109/MC.2016.145

[12] Li, S., Xu, L.D. & Zhao, S. The internet of things: a survey. Inf Syst Front 17, 243–259 (2015). doi:10.1007/s10796-014-9492-7

[13] Kopetz, H., Steiner, W. (2022). Internet of Things. In: Real-Time Systems. Springer, Cham. doi:10.1007/978-3-031-11992-7_13

[14] “Benefits of Edge Computing: Make Data Processing Faster & Secure”, https://www.cloudpanel.io/blog/benefits-of-edge-computing/

[15] “The Rise of Edge Computing: Understanding Its Benefits and Drawbacks”, https://xailient.com/blog/the-rise-of-edge-computing-understanding-its-benefits-and-drawbacks/

[16] Bajic, Bojana, Ilija Cosic, Branko Katalinic, Slobodan Moraca, Milovan Lazarevic, and Aleksandar Rikalovic. "EDGE COMPUTING VS. CLOUD COMPUTING: CHALLENGES AND OPPORTUNITIES IN INDUSTRY 4.0." Annals of DAAAM & Proceedings 30 (2019), doi:10.2507/30th.daaam.proceedings.120

[17] “What is Cloud Native?”, https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/definition

[18] “What is a cloud-native application?”, https://www.techtarget.com/searchcloudcomputing/definition/cloud-native-application

[19] K. Gos and W. Zabierowski, "The Comparison of Microservice and Monolithic Architecture," 2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH), Lviv, Ukraine, 2020, pp. 150-153, doi:10.1109/MEMSTECH49584.2020.9109514

[20] “Introduction to Monolithic Architecture and MicroServices Architecture”, https://medium.com/koderlabs/introduction-to-

monolithic-architecture-and-microservices-architecture-b211a5955c63

[21] S. D. A. Shah, M. A. Gregory and S. Li, "Cloud-Native Network Slicing Using Software Defined Networking Based Multi-Access Edge Computing: A Survey," in IEEE Access, vol. 9, pp. 10903-10924, 2021, doi:10.1109/ACCESS.2021.3050155

[22] Carrión, C. Kubernetes as a Standard Container Orchestrator - A Bibliometric Analysis. J Grid Computing 20, 42 (2022). doi:10.1007/s10723-022-09629-8

[23] “What is a virtual machine, and why are they so useful?”, https://www.networkworld.com/article/969185/what-is-a-virtual-machine-and-why-are-they-so-useful.html

[24] “What is a Hypervisor?”, https://www.nutanix.com/info/hypervisor

[25] “What is container orchestration?”, https://www.vmware.com/topics/container-orchestration

[26] “What is a container image?”, https://www.ibm.com/think/topics/container-images

[27] “Containers vs Virtual Machines: A Detailed Comparison for Developers”, https://www.datacamp.com/blog/containers-vs-virtual-machines

[28] “Cluster Architecture: The architectural concepts behind Kubernetes.”, https://kubernetes.io/docs/concepts/architecture/

[29] “What Are Containerized Microservices?”, https://blog.dreamfactory.com/what-are-containerized-microservices

[30] “Mobile Edge Computing: What is Mobile Edge Computing?”, https://stlpartners.com/articles/edge-computing/mobile-edge-computing/

[31] “What is URLLC?”, https://inseego.com/resources/5g-glossary/what-is-urllc/

[32] “URLLC: What it is and how it works”, https://blog.antenova.com/urllc-what-it-is-and-how-it-works

[33] “What is edge AI?”, https://www.ibm.com/think/topics/edge-ai

[34] Soumplis, P., Kontos, G., Kokkinos, P. et al. Performance Optimization Across the Edge-Cloud Continuum: A Multi-agent Rollout Approach for Cloud-Native Application Workload Placement. SN COMPUT. SCI. 5, 318 (2024). doi:10.1007/s42979-024-02630-w

[35] K. Rao, G. Coviello, W. -P. Hsiung and S. Chakradhar, "ECO: Edge-Cloud Optimization of 5G applications," 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), Melbourne, Australia, 2021, pp. 649-659, doi:10.1109/CCGrid51090.2021.00078

[36] M. -T. Thai, Y. -D. Lin, Y. -C. Lai and H. -T. Chien, "Workload and Capacity Optimization for Cloud-Edge Computing Systems with Vertical and Horizontal Offloading," in IEEE Transactions on Network and Service Management, vol. 17, no. 1, pp. 227-238, March 2020, doi:10.1109/TNSM.2019.2937342

[37] T. -M. Pham and T. -T. -L. Nguyen, "Optimization of Resource Management for NFV-Enabled IoT Systems in Edge Cloud Computing," in IEEE Access, vol. 8, pp. 178217-178229, 2020, doi:10.1109/ACCESS.2020.3026711

[38] J. Zhang et al., "Joint Resource Allocation for Latency-Sensitive Services Over Mobile Edge Computing Networks With Caching," in IEEE Internet of Things Journal, vol. 6, no. 3, pp. 4283-4294, June 2019, doi:10.1109/JIOT.2018.2875917

[39] M. Chen, Y. Hao, L. Hu, M. S. Hossain and A. Ghoneim, "Edge-CoCaCo: Toward Joint Optimization of Computation, Caching, and Communication on Edge Cloud," in IEEE Wireless Communications, vol. 25, no. 3, pp. 21-27, JUNE 2018, doi:10.1109/MWC.2018.1700308

[40] J. Ren, G. Yu, Y. He and G. Y. Li, "Collaborative Cloud and Edge Computing for Latency Minimization," in IEEE Transactions on Vehicular Technology, vol. 68, no. 5, pp. 5031-5044, May 2019,

doi:10.1109/TVT.2019.2904244

[41] Pal, S.; Jhanjhi, N.Z.; Abdulbaqi, A.S.; Akila, D.; Almazroi, A.A.; Alsubaei, F.S. A Hybrid Edge-Cloud System for Networking Service Components Optimization Using the Internet of Things. Electronics 2023, 12, 649. doi:10.3390/electronics12030649

[42] A. Yousefpour, G. Ishigaki, R. Gour and J. P. Jue, "On Reducing IoT Service Delay via Fog Offloading," in IEEE Internet of Things Journal, vol. 5, no. 2, pp. 998-1010, April 2018, doi:10.1109/JIOT.2017.2788802

[43] E. El Haber, T. M. Nguyen and C. Assi, "Joint Optimization of Computational Cost and Devices Energy for Task Offloading in Multi-Tier Edge-Clouds," in IEEE Transactions on Communications, vol. 67, no. 5, pp. 3407-3421, May 2019, doi:10.1109/TCOMM.2019.2895040

[44] W. -Z. Zhang et al., "Secure and Optimized Load Balancing for Multitier IoT and Edge-Cloud Computing Systems," in IEEE Internet of Things Journal, vol. 8, no. 10, pp. 8119-8132, 15 May15, 2021, doi:10.1109/JIOT.2020.3042433

[45] Cao K, Wei T, Chen M, Li K, Weng J, Tan W. Exploring reliable edge-cloud computing for service latency optimization in sustainable cyber-physical systems. Softw Pract Exper. 2021;1–13. doi:10.1002/spe.2942

[46] L. Yang, B. Liu, J. Cao, Y. Sahni and Z. Wang, "Joint Computation Partitioning and Resource Allocation for Latency Sensitive Applications in Mobile Edge Clouds," in IEEE Transactions on Services Computing, vol. 14, no. 5, pp. 1439-1452, 1 Sept.-Oct. 2021, doi:10.1109/TSC.2018.2890603

[47] I. Sartzetakis, P. Soumplis, P. Pantazopoulos, K. V. Katsaros, V. Sourlas and E. M. Varvarigos, "Resource Allocation for Distributed Machine Learning at the Edge-Cloud Continuum," ICC 2022 - IEEE International Conference on Communications, Seoul, Korea, Republic of, 2022, pp. 5017-5022, doi:10.1109/ICC45855.2022.9838647