ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μελέτη μηχανισμών δέσμευσης πόρων για την υποστήριξη εφαρμογών νέφους σε μια ιεραρχική edge-fog-cloud τοπολογία

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Κ. Κόντος

Επιβλέπων : Εμμανουήλ Βαρβαρίγος

Καθηγητής Ε.Μ.Π.

Αθήνα,Σεπτέμβριος,2022

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μελέτη μηχανισμών δέσμευσης πόρων για την υποστήριξη εφαρμογών νέφους σε μια ιεραρχική edge-fog-cloud τοπολογία

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Κ. Κόντος

Επιβλέπων : Εμμανουήλ Βαρβαρίγος
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 2η Σεπτεμβρίου 2022.

....	....	....
Εμμανουήλ Βαρβαρίγος	Ηρακλής Αβραμόπουλος	Θεοδώρα Βαρβαρίγου
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επ. Καθηγήτρια Ε.Μ.Π.

Αθήνα,Σεπτέμβριος,2022

....

Γεώργιος Κ. Κόντος

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

Copyright © Γεώργιος Κόντος,2022
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Στο ανταγωνιστικό και ευμετάβλητο περιβάλλον του σήμερα, η cloud-native προσέγγιση για τη δόμηση εφαρμογών κερδίζει σταδιακά έδαφος έναντι του παραδοσιακού μοντέλου. Ακρογωνιαίο λίθο των εφαρμογών νέφους (cloud-native) αποτελεί η τεχνολογία των μικρό-υπηρεσιών (microservices), μικρών αυτοτελών τμημάτων της εφαρμογής που εκτελούνται αυτόνομα μέσω της ενθυλάκωσής τους σε containers. Τη διαχείριση των τελευταίων αναλαμβάνει ο ενορχηστρωτής container, μέρος της λειτουργίας του οποίου είναι η δέσμευση των απαραίτητων πόρων στα φυσικά μηχανήματα για την απρόσκοπτη λειτουργία των εφαρμογών.

Ωστόσο, καθώς η ιδέα του Ίντερνετ των Πραγμάτων ωριμάζει με την εδραίωση των κινητών δικτύων 5ης γενιάς, η ποσότητα πληροφορίας που παράγεται στα άκρα του δικτύου αυξάνεται ραγδαία, δίνοντας το έναυσμα για τη δημιουργία των κατανεμημένων αρχιτεκτονικών των fog και edge για την υποστήριξη των νέων εφαρμογών. Οι γνωστότεροι σύγχρονοι ενορχηστρωτές κρίνονται συχνά ανεπαρκείς στη διαχείριση μιας τέτοιας κατανεμημένης υποδομής οδηγώντας στην παραβίαση των προδιαγραφών που φέρουν οι εφαρμογές και στη γενικότερη υποβάθμιση της παρεχόμενης ποιότητας υπηρεσιών.

Σκοπός της παρούσας εργασίας είναι η ανεύρεση ενός μηχανισμού για την αυτοματοποίηση της δέσμευσης υπολογιστικών πόρων, με σκοπό τη βέλτιστη εξυπηρέτηση cloud-native εφαρμογών σε ένα διαστρωματωμένο edge-fog-cloud περιβάλλον. Ο μηχανισμός αυτός λαμβάνει τη μορφή ενός αλγορίθμου και ενσωματώνεται στο λογισμικό των ενορχηστρωτών. Συγκεκριμένα, αναζητούμε την κατάλληλη ανάθεση των μικρό-υπηρεσιών σε υπολογιστικά συστήματα, με σκοπό τη βελτιστοποίηση ενός βεβαρυμμένου συνδυασμού της καθυστέρησης και του κόστους εξυπηρέτησης. Αρχικά μοντελοποιούμε το παραπάνω πρόβλημα σαν πρόβλημα ακέραιου γραμμικού προγραμματισμού. Για την επίλυση αναπτύσσουμε τον προσεγγιστικό αλγόριθμο *GRAA*. Στη συνέχεια επιστρατεύουμε την τεχνική του "ξετυλίγματος" (*Rollout*), για την αποτελεσματικότερη χρήση της οποίας κατασκευάζουμε τον -πιο κατάλληλο- προσεγγιστικό αλγόριθμο *H*. Τέλος, αξιολογούμε τα αποτελέσματα των αλγορίθμων μέσω εκτενών προσομοιώσεων υπό διάφορα δεδομένα εισόδου και αποδεικνύουμε την ποιότητα των παραγόμενων υπό-βέλτιστων λύσεων μέσω σύγκρισης με την ακριβή λύση του προβλήματος.

Λέξεις κλειδιά: cloud computing, fog computing, edge computing, cloud-native, containers, ενορχηστρωτής container, microservices, δέσμευση πόρων

Abstract

In today's competitive and volatile environment, the cloud-native approach to building applications is gradually gaining ground over the traditional model. A cornerstone of cloud-native applications is the technology of microservices, small independent components of an application that run autonomously through their encapsulation in containers. The management of the latter is undertaken by the container orchestrator, part of whose operation is the allocation of the necessary resources to the physical machines for the seamless operation of the applications.

However, as the concept of the Internet of Things matures with the consolidation of 5th generation mobile networks, the amount of information generated at the edge of the network is increasing rapidly, triggering the creation of the distributed architectures of fog and edge to support the new applications. The most well-known modern orchestrators are often considered inadequate in managing such a distributed infrastructure, leading to the violation of the specifications carried by the applications and the general degradation of the quality of services provided.

The purpose of this work is to discover a mechanism to automate the allocation of computing resources, in order to optimally serve cloud-native applications in a layered edge-fog-cloud environment. This mechanism takes the form of an algorithm and is integrated into the software of the orchestrators. Specifically, we seek the appropriate assignment of microservices to computing systems, with the aim of optimizing a weighted combination of delay and service cost. We first model the above problem as an integer linear programming problem. For the solution we develop the *GRAA* approximation algorithm. Then we employ the "Rollout" technique, for the most efficient use of which we construct the -more suitable- approximation algorithm *H*. Finally, we evaluate the results of the algorithms through extensive simulations under various input data and prove the quality of the generated sub - optimal solutions by comparison with the exact solution of the problem.

Keywords: cloud computing, fog computing, edge computing, cloud-native, containers, container orchestrator, microservices, resource allocation

Ευχαριστίες

Η ολοκλήρωση της παρούσας διπλωματικής σηματοδοτεί το τέλος μιας πολυετούς προσπάθειας. Αρχικά θα ήθελα να ευχαριστήσω τον κ. Εμμανουήλ Βαρβαρίγο, για την ανάθεση ενός τόσο ενδιαφέροντος και σύγχρονου θέματος. Τις θερμότερες ευχαριστίες δίνω στον κ. Πολυζώη Σούμπλη, για την πολύτιμη καθοδήγησή του σε όλα τα στάδια εκπόνησης της εργασίας. Τέλος, ευχαριστώ τους φίλους μου για τη στήριξη, τη συμπαράσταση και τη συντροφιά καθ' όλη τη διάρκεια του ακαδημαϊκού μου ταξιδιού.

Περιεχόμενα

Περίληψη......6

Abstract......7

Ευχαριστίες......9

Περιεχόμενα......11

Περιεχόμενα Εικόνων......13

Περιεχόμενα Πινάκων......14

1 Εισαγωγή......15

1.1 Συνοπτική περιγραφή του προβλήματος......15

1.2 Σκοπός και δομή της εργασίας......16

2 Περιγραφή των edge-fog-cloud υποδομών και των χαρακτηριστικών τους......19

2.1 Cloud computing......19

2.1.1 Ορισμός του cloud computing......19

2.1.2 Πλεονεκτήματα του cloud computing......20

2.1.3 Εικονικοποίηση......20

2.1.4 Είδη υπηρεσιών υπολογιστικού νέφους......24

2.1.5 Τα είδη του νέφους......26

2.2 Edge Computing......27

2.2.1 Internet of Things......27

2.2.2 Η ανεπάρκεια του cloud......28

2.2.3 Ορισμός του προτύπου Edge Computing......29

2.2.4 Mobile Edge Computing (MEC)......30

2.3 Fog Computing......32

2.3.1 Cloudlets......33

3 Περιγραφή των cloud-native εφαρμογών......35

3.1 Η ανεπάρκεια των παραδοσιακών μονολιθικών εφαρμογών......35

3.2 Cloud-native εφαρμογές......36

3.2.1 Ορισμός του cloud-native......37

3.2.2 Βασικές χρησιμοποιούμενες τεχνολογίες......37

3.3 Cloud-native εφαρμογές σε μια ιεραρχική edge-fog-cloud τοπολογία......41

4 Σχετική Εργασία......43

5 Μεικτή βεβαρυμμένη βελτιστοποίηση καθυστέρησης-κόστους......47

5.1 Περιγραφή των στοιχείων της υποδομής και του προβλήματος......47

5.2 Περιγραφή του αλγορίθμου GRAA (Greedy Resource Allocation Algorithm) 53

5.2.1 Είσοδοι στον αλγόριθμο 53
5.2.2 Ο αλγόριθμος 54
5.3 Αποτελέσματα και αξιολόγηση 61
5.4 Rollout 66
5.4.1 Ευρετικοί και μετά-ευρετικοί αλγόριθμοι 66
5.4.2 Ο αλγόριθμος του Rollout 66
5.4.3 Επεξήγηση του αλγορίθμου και ιδιότητες 67
5.5 Ο βασικός ευρετικός H 71
5.6 Rollout με χρήση του *H* 74
5.7 Αποτελέσματα και αξιολόγηση 76
5.8 Σύγκριση των αποτελεσμάτων των GRAA, H, και Rollout με την ακριβή λύση του προβλήματος ΑΓΠ 81
5.9 Ανακεφαλαίωση, συμπεράσματα και συνολική αξιολόγηση 85
6 Επεκτάσεις – μελλοντική εργασία 86
6.1 Εξέταση του "σε σύνδεση" (online) προβλήματος 86
6.2 Χρήση γράφων για την αναπαράσταση των εφαρμογών 87
6.3 Επίγνωση τοποθεσίας – Εξατομικευμένη καθυστέρηση 87
6.4 Επίγνωση δικτύωσης 88
6.5 Διερεύνηση άλλων συναφών προβλημάτων 88
6.6 Χρήση αλγορίθμων πλειστηριασμού 88
Βιβλιογραφία 89

Περιεχόμενα Εικόνων

Εικόνα 2-1 Cloud Computing.....19
Εικόνα 2-2 Σχέση container και εικόνας container.....23
Εικόνα 2-3 Σύγκριση container και εικονικών μηχανών.....24
Εικόνα 2-4 Επίπεδα ελέγχου χρήστη [15].....26
Εικόνα 2-5 Ρυθμός ανάπτυξης IoT συσκευών [18].....28
Εικόνα 2-6 Το παραδοσιακό μοντέλο cloud.....28
Εικόνα 2-7 Το μοντέλο του Edge Computing.....29
Εικόνα 2-8 Στρώματα IoT [25].....32
Εικόνα 3-1 Μονολιθική Εφαρμογή.....35
Εικόνα 3-2 Cloud Native Εφαρμογή.....36
Εικόνα 3-3 Εφαρμογή βασισμένη σε Microservices [30].....38
Εικόνα 3-4 Τα συστατικά του Kubernetes [36].....41
Εικόνα 5-1 Αναπαράσταση της υποδομής.....48
Εικόνα 5-2 Αναπαράσταση μιας εφαρμογής υπό τη μορφή γράφου.....49
Εικόνα 5-3 Αναπαράσταση μιας τυπικής εφαρμογής του προβλήματος.....50
Εικόνα 5-5 Διάγραμμα ροής βήματος 1.....56
Εικόνα 5-6 Παράδειγμα βήματος 2.....57
Εικόνα 5-7 Διάγραμμα ροής βήματος 2.....58
Εικόνα 5-8 Παράδειγμα βήματος 3.....58
Εικόνα 5-9 Διάγραμμα ροής βήματος 3.....59
Εικόνα 5-10 Διάγραμμα ροής βήματος 4.....60
Εικόνα 5-11 Συνολικό διάγραμμα ροής για τον GRAA.....61
Εικόνα 5-12 Σχηματικά αποτελέσματα πειράματος.....63
Εικόνα 5-13 Φορτίο στους κόμβους του Fog στρώματος για βάρος w=0.4.....64
Εικόνα 5-14 Φορτίο στους κόμβους του Fog στρώματος για βάρος w=0.6.....65
Εικόνα 5-15 Τυπικό βήμα του Rollout.....68
Εικόνα 5-16 Διάγραμμα ροής για τον ευρετικό αλγόριθμο H.....74
Εικόνα 5-17 Ανεύρεση συστατικού της λύσης.....75
Εικόνα 5-18 Παράδειγμα εξαγωγής της τελικής λύσης.....76
Εικόνα 5-19 Τιμές από το Rollout για κάθε τοποθέτηση εφαρμογής, για βάρος w=0.2.....77
Εικόνα 5-20 Βελτίωση από τη χρήση του Rollout.....78
Εικόνα 5-21 Σύγκριση φορτίου κόμβων.....79
Εικόνα 5-22 Βελτίωση ανά εκτέλεση.....80
Εικόνα 5-23 Σύγκριση αποτελεσμάτων για w=0.....83
Εικόνα 5-24 Σύγκριση αποτελεσμάτων για w=0.2.....83
Εικόνα 5-25 Σύγκριση αποτελεσμάτων για w=0.8.....84
Εικόνα 5-26 Σύγκριση αποτελεσμάτων για w=1.....84

Περιεχόμενα Πινάκων

Πίνακας 4-1 Σύνοψη της σχετικής εργασίας 45
Πίνακας 5-1 Στοιχεία της πειραματικής υποδομής του GRAA 62
Πίνακας 5-2 Σχετικές καθυστερήσεις 62
Πίνακας 5-3 Στοιχεία πειραματικών εφαρμογών 62
Πίνακας 5-4 Αποτελέσματα πειράματος 63
Πίνακας 5-5 Στοιχεία της πειραματικής υποδομής 76
Πίνακας 5-6 Σχετικές καθυστερήσεις 77
Πίνακας 5-7 Στοιχεία πειραματικών εφαρμογών 77
Πίνακας 5-8 Αποτελέσματα πειράματος 78
Πίνακας 5-9 Μέση τιμή και διακύμανση βελτίωσης 80
Πίνακας 5-10 Στοιχεία της πειραματικής υποδομής 81
Πίνακας 5-11 Σχετικές καθυστερήσεις 82
Πίνακας 5-12 Στοιχεία πειραματικών εφαρμογών 82

1 Εισαγωγή

1.1 Συνοπτική περιγραφή του προβλήματος

Στην κλασσική μονολιθική προσέγγιση, όλη η λογική μιας εφαρμογής συγκεντρώνεται σε μια ενιαία οντότητα: Τα επιμέρους κομμάτια του κώδικα είναι στενά συνδεδεμένα μεταξύ τους , και απαιτείται η παρουσία όλων προκειμένου η εφαρμογή να λειτουργήσει ή να ενημερωθεί. Ακόμη, η επεξεργασία και η αποθήκευση συνιστούν έκαστη μια και μοναδική διαδικασία που συμβαίνει σε έναν εξυπηρετητή ή μια βάση δεδομένων αντίστοιχα.

Στη σύγχρονη εποχή, με τη σταδιακή εδραίωση των κινητών δικτύων 5ης γενιάς, δημιουργείται πρόσφορο έδαφος για τη δημιουργία νέων τεχνολογιών και καινοτόμων εφαρμογών που σπεύδουν να εκμεταλλευτούν τις νέες δικτυακές δυνατότητες. Προκειμένου να προσφέρουν υψηλότερης ποιότητας υπηρεσίες, οι εφαρμογές περιπλέκονται, υπάρχει διαρκής ανάγκη για αλλαγές και βελτιώσεις ενώ μια βλάβη που θα παρεμποδίσει την ορθή λειτουργία της εφαρμογής για ορισμένο διάστημα μπορεί να επιφέρει τεράστιο οικονομικό, και όχι μόνο, κόστος. Σε ένα τόσο ευμετάβλητο και ανταγωνιστικό περιβάλλον, το δύσκαμπτο μονολιθικό μοντέλο κρίνεται ανεπαρκές, δίνοντας το έναυσμα για τη δημιουργία ενός νέου παραδείγματος για τη δόμηση εφαρμογών, το cloud-native. Το τελευταίο εκμεταλλεύεται τα εγγενή χαρακτηριστικά του cloud, προσφέροντας μεταξύ άλλων στην εφαρμογή αυξημένη ανθεκτικότητα και ευκολότερη διαχείριση. Η κύρια τεχνολογία που επιτρέπει τη σχεδίαση cloud-native εφαρμογών είναι οι μικρό-υπηρεσίες (microservices), μικρά αυτοτελή τμήματα της εφαρμογής, όπου το καθένα ενσωματώνει δικό του κώδικα, εκτελείται αυτόνομα και εξυπηρετεί συγκεκριμένο σκοπό. Η εκτέλεση συμβαίνει σε αυτόνομα υπολογιστικά περιβάλλοντα, τα containers, δεσμεύοντας από το σύστημα στο οποίο εδράζονται τους απαιτούμενους πόρους.

Ωστόσο, ολοένα και περισσότερες συσκευές της καθημερινότητας εκσυγχρονίζονται και υπάγονται πλέον στην ιδέα του ίντερνετ των πραγμάτων. Το τελευταίο μπορεί να οριστεί απλοϊκά ως ένα σύστημα συνδεδεμένων συσκευών, μηχανικών και ψηφιακών συστημάτων και γενικότερα αντικειμένων που φέρουν μοναδικά αναγνωριστικά (Unique Identifiers, UID's), και έχουν την ικανότητα να μεταφέρουν πληροφορία διαμέσου ενός δικτύου χωρίς την ανθρώπινη παρέμβαση. Ωστόσο, η αλματώδης αύξηση στον αριθμό τέτοιων συνδεδεμένων αντικειμένων δημιουργεί έναν τεράστιο όγκο πληροφορίας στα άκρα του δικτύου, που συχνά χρήζει άμεσης επεξεργασίας. Συνεπώς, οι απομακρυσμένοι εξυπηρετητές του cloud εισάγουν απαγορευτικές καθυστερήσεις για εφαρμογές που φέρουν υψηλή ευαισθησία στο χρόνο απόκρισης, όπως η πλοήγηση αυτόνομων οχημάτων, η εικονική πραγματικότητα και οι εφαρμογές βίο-ιατρικής. Επιπλέον, καθώς όλος ο πληροφοριακός όγκος πρέπει να διακομιστεί στο cloud, τα ενδιάμεσα δικτυακά στρώματα (Μητροπολιτικά Δίκτυα - MAN's, Δίκτυα ευρείας περιοχής – WAN's) υποφέρουν από φαινόμενα συμφόρησης (bottleneck). Τα παραπάνω, μείζονος σημασίας, προβλήματα δημιουργούν την ανάγκη για την τοποθέτηση δικτυακών και υπολογιστικών πόρων πλησιέστερα στην πηγή παραγωγής δεδομένων. Σε πρώτο

επίπεδο, η αρχιτεκτονική του fog επεκτείνει το παράδειγμα του cloud προς τα άκρα του δικτύου, προσφέροντας ισχυρούς πόρους σε μια ενδιάμεση απόσταση και ανακουφίζοντας τα δίκτυα ευρείας περιοχής. Σε δεύτερο επίπεδο και ακόμη πλησιέστερα στην πηγή παραγωγής συναντούμε το edge στρώμα που απαρτίζεται από χαμηλότερης δυναμικής συστήματα, και σκοπό έχει την εκτέλεση χρόνο-ευαίσθητων εργασιών και τη μείωση της δικτυακής κίνησης (traffic) που διακομίζεται στο cloud. Μπορούμε να πούμε πως αμφότερα τα edge και fog συνιστούν κατανεμημένα αρχιτεκτονικά μοντέλα που σκοπεύουν στην αποκεντροποίηση της δικτύωσης, της αποθήκευσης και της επεξεργασίας, ανταλλάσσοντας δυναμική -από άποψη πόρων- για εγγύτητα. Ωστόσο, η ελαφριά φύση των container επιτρέπει την εκτέλεσή τους ακόμη και στα φτωχότερα συστήματα του edge, επιτρέποντας έτσι την υποστήριξη cloud-native εφαρμογών, η τμημάτων αυτών, στα άκρα του δικτύου.

Το πρόβλημα που εξετάζει η παρούσα εργασία είναι η ανεύρεση μηχανισμών για την κατάλληλη κατανομή και διαχείριση των πόρων των διαθέσιμων συστημάτων στα διάφορα στρώματα της τοπολογίας edge-fog-cloud, για την υποστήριξη του εισερχόμενου φόρτου εργασίας από τα microservices. Σκοπός είναι η βελτιστοποίηση ενός συνδυασμού παραγόντων όπως η μέση καθυστέρηση και το μέσο κόστος εξυπηρέτησης, με σεβασμό στα ιδιαίτερα χαρακτηριστικά κάθε microservice, όπως η ανάγκη για καθυστέρηση που εγγυημένα δεν ξεπερνά ένα άνω όριο.

Αν αποστειρώσουμε το πρόβλημα από τις εξειδικευμένες λεπτομέρειες , όπως οι χρησιμοποιούμενες τεχνολογίες και τα συστήματα της υποδομής, θα συνειδητοποιήσουμε πως πρόκειται για ένα πολύ γενικότερο πρόβλημα: Έχοντας συγκεκριμένους διαθέσιμους πόρους, αναζητούμε την κατάλληλη ανάθεσή τους για την βελτιστοποίηση κάποιου παράγοντα οικονομικής, χρονικής, ενεργειακής κλπ. φύσεως. Αυτή η αφηρημένη διατύπωση συνιστά το γενικό πρόβλημα της δέσμευσης πόρων, που συναντάται σε ένα ευρύ πεδίο εφαρμογών όπως η επιλογή του αριθμού και της τοποθεσίας των νέων εγκαταστάσεων μιας επιχείρησης, η ανάθεση εργασιών στο ανθρώπινο δυναμικό, η δημιουργία μιας αποτελεσματικής γραμμής παραγωγής εργοστασίου κλπ. Ωστόσο, αν λάβουμε υπόψη τα συγκεκριμένα χαρακτηριστικά κάθε προβλήματος μπορούμε να σχεδιάσουμε πολύ πιο αποτελεσματικές λύσεις για κάθε ένα από αυτά αντί να αναζητήσουμε λύση με μια ολιστική προσέγγιση.

1.2 Σκοπός και δομή της εργασίας

Στην παρούσα εργασία θα προσπαθήσουμε να ενσωματώσουμε στη θεώρηση του προβλήματος τα απαραίτητα τεχνικά χαρακτηριστικά που θα μας βοηθήσουν στην αναζήτηση μιας καλύτερης και πιο συγκεκριμένης λύσης, χωρίς ωστόσο να εντρυφήσουμε ενδελεχώς στις χρησιμοποιούμενες τεχνολογίες και ιδιαίτερα σε αυτές που αφορούν το κομμάτι του σχεδιασμού και της δικτύωσης. Για μια πληρέστερη εικόνα, παρακινούμε τον αναγνώστη να εξερευνήσει περισσότερο τις αναφερθείσες έννοιες των κεφαλαίων της θεωρίας μέσω των παραπομπών. Η προσέγγιση μας θα διατηρήσει έναν ελαφρώς γενικευμένο χαρακτήρα, προσφέροντας ένα πλαίσιο ικανό να τροποποιηθεί κατάλληλα και να προσαρμοστεί στις ανάγκες του εκάστοτε ενδιαφερόμενου. Για το σκοπό αυτό θα εστιάσουμε στη

μαθηματική – αλγοριθμική πλευρά του προβλήματος. Το πρόβλημα της δέσμευσης πόρων για την υποστήριξη cloud-native εφαρμογών είναι στενά συνυφασμένο με την έννοια του ενορχηστρωτή (orchestrator), ένα λογισμικό που μεταξύ άλλων αναλαμβάνει, μέσω ενός ενσωματωμένου μηχανισμού, την αυτόματη δέσμευση των απαραίτητων πόρων για την υποστήριξη του φόρτου εργασίας (workload), που εν προκειμένω αφορά microservices εφαρμογών που ενθυλακώνονται σε containers. Ωστόσο, οι περισσότεροι σύγχρονοι ενορχηστρωτές δεν είναι κατάλληλα σχεδιασμένοι για μια γεωγραφικά διεσπαρμένη υποδομή, ενώ συχνά αποτυγχάνουν στην τήρηση των προδιαγραφών που εισάγουν οι εφαρμογές, όπως περιορισμοί στην καθυστέρηση ή ανάγκες για χαμηλό κόστος εξυπηρέτησης. Το έργο μας εστιάζει στη δημιουργία ενός καινοτόμου τέτοιου μηχανισμού, που θα αναλαμβάνει την κατάλληλη αντιστοίχιση των containers με υπολογιστικά συστήματα, μεριμνώντας για την ικανοποίηση των περιορισμών και τη γενικότερη ευημερία των υποστηριζόμενων εφαρμογών. Συγκεκριμένα εξετάζουμε το πρόβλημα της "μεικτής βεβαρυμμένης βελτιστοποίησης καθυστέρησης-κόστους", όπου ανάλογα με το "βάρος" αναζητούμε μια συνδυαστική βελτιστοποίηση της μέσης καθυστέρησης και του μέσου κόστους ανά microservice. Για την επίλυση αρχικά παρουσιάζουμε έναν ισχυρό προσεγγιστικό αλγόριθμο, ενώ στη συνέχεια κατασκευάζουμε έναν απλούστερο αλλά πολύ γρηγορότερο προσεγγιστικό αλγόριθμο ο οποίος αξιοποιείται από την τεχνική του ξετυλίγματος (Rollout) για περεταίρω βελτιστοποίηση.

Ακολουθεί μια σύνοψη της δομής της εργασίας :

Στο κεφάλαιο 2 θα αναλύσουμε αρχικά την έννοια του cloud computing, παρουσιάζοντας τα χαρακτηριστικά που το διέπουν και την κύρια τεχνολογία που επέτρεψε τη δημιουργία του, την εικονικοποίηση. Στη συνέχεια θα εξετάσουμε τα παραδείγματα του fog και edge computing, εστιάζοντας στα ειδοποιά στοιχεία του καθενός και στις μεταξύ τους διαφορές, καθώς και στους παράγοντες που οδήγησαν στην καθιέρωσή τους, όπως το Ίντερνετ των πραγμάτων και τα δίκτυα 5ης γεννιάς.

Στο κεφάλαιο 3 θα παρουσιάσουμε την cloud-native προσέγγιση για τη δημιουργία εφαρμογών, αναδεικνύοντας τα πλεονεκτήματά της έναντι της μονολιθικής και εν συνεχεία εξετάζοντας τις τεχνολογίες που χρησιμοποιεί. Ακόμη, θα αναδείξουμε την ανάγκη για την ύπαρξη μιας κατανεμημένης ιεραρχικής τοπολογίας και θα κάνουμε μια εισαγωγή στο γενικό πρόβλημα της δέσμευσης υπολογιστικών πόρων σε μια τέτοια τοπολογία.

Στο 4ο κεφάλαιο θα κάνουμε μια σύντομη βιβλιογραφική επισκόπηση συναφών προβλημάτων, κατηγοριοποιώντας τις λύσεις σύμφωνα με τις μαθηματικές και λογικές προσεγγίσεις.

Στο κεφάλαιο 5 περιγράφουμε λεπτομερώς το πρόβλημα της μεικτής βεβαρυμμένης βελτιστοποίησης καθυστέρησης- κόστους και το εκφράζουμε σαν πρόβλημα ακέραιου γραμμικού προγραμματισμού (ΑΓΠ). Στη συνέχεια παρουσιάζουμε τον πρώτο άπληστο προσεγγιστικό αλγόριθμο ονόματι Greedy Resource Allocation Algorithm (GRAA), εξετάζουμε τα αποτελέσματα της χρήσης του και εντοπίζουμε τις αδυναμίες του. Έπειτα αναλύουμε τον μηχανισμό του ξετυλίγματος (Rollout) , κατασκευάζουμε τον -πιο κατάλληλο- άπληστο προσεγγιστικό αλγόριθμο H και

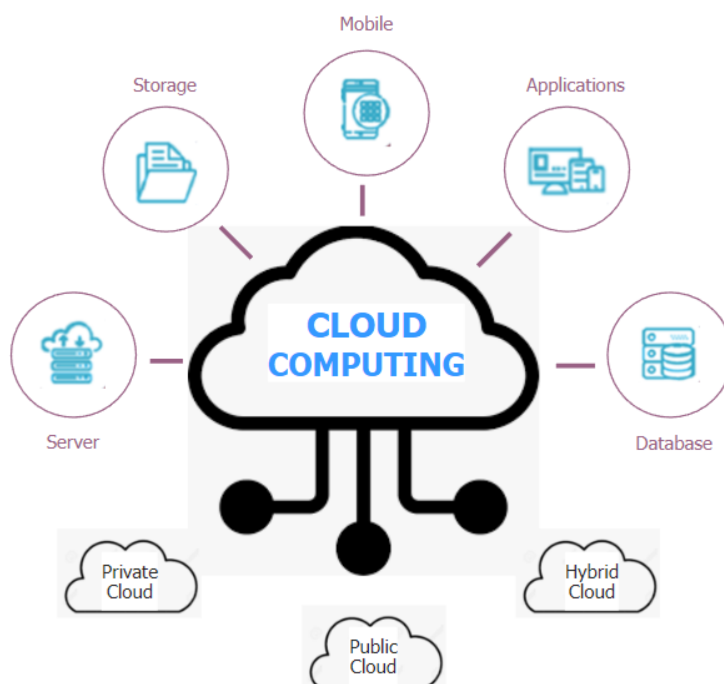
τέλος εφαρμόζουμε το Rollout με χρήση του H και αξιολογούμε τα αποτελέσματα των προσομοιώσεων.

Τέλος, στο 6^{ο} κεφάλαιο γίνεται μια νύξη σε πιθανές προεκτάσεις και μελλοντική εργασία.

2 Περιγραφή των edge-fog-cloud υποδομών και των χαρακτηριστικών τους

2.1 Cloud computing

2.1.1 Ορισμός του cloud computing



Εικόνα 2-1 Cloud Computing

Ο όρος νέφος (Cloud) προκύπτει ιστορικά από τα "νηπιακά" στάδια του διαδικτύου, όταν αυτό αναπαρίστανto ως ένα σύννεφο. [1],[2] Ο αόριστος αυτός συμβολισμός αυτός είχε σκοπό την απλοποίηση των σχετικών διαγραμμάτων και παρουσιάσεων, καθώς οι χιλιάδες των εσωτερικών συνδέσεων στο διαδίκτυο ήταν αμελητέας σημασίας για την κατανόηση του νοήματός τους.

Σύμφωνα με την επίσημη ιστοσελίδα της IBM [3] , το υπολογιστικό νέφος είναι η κατά παραγγελία (on demand) διάθεση, μέσω του διαδικτύου, υπολογιστικών πόρων, εφαρμογών, εξυπηρετητών (servers), αποθηκευτικού χώρου και άλλων, τα οποία φιλοξενούνται σε απομακρυσμένα κέντρα δεδομένων. Τα κέντρα αυτά διαχειρίζονται οι πάροχοι υπηρεσιών νέφους (Cloud Service Providers – CSP's).

Σε μια πιο γενικευμένη προσέγγιση, οι Ling Qian et al. [4], ορίζουν το υπολογιστικό νέφος σαν ένα είδος υπολογιστικής τεχνικής όπου οι υπηρεσίες πληροφορικής παρέχονται από τεράστιες στο πλήθος , χαμηλού κόστους υπολογιστικές μονάδες συνδεδεμένες με δίκτυα IP. Το υπολογιστικό νέφος πρέπει επίσης να φέρει τα εξής τεχνικά χαρακτηριστικά:

- Μεγάλης κλίμακας υπολογιστικοί πόροι
- Εύκολη επεκτασιμότητα και ευελιξία
- Κοινής πρόσβασης "δεξαμενή" πόρων (φυσικών και εικονικών)

- Δυναμικός προγραμματισμός για τη διάθεση πόρων

Απλοϊκά, υπολογιστικό νέφος είναι η απομακρυσμένη πρόσβαση σε υπολογιστικούς πόρους, δεδομένα και προγράμματα, αλλά και η αποθήκευση αυτών, στο διαδίκτυο αντί για τον σκληρό δίσκο του υπολογιστή.

2.1.2 Πλεονεκτήματα του cloud computing

Το 2021, η συνολική αξία της αγοράς του cloud computing άγγιζε τα 372 δισεκατομμύρια δολάρια, ενώ προβλέπεται ετήσια αύξηση της τάξεως του 17.5% [5]Ακόμη, η χρησιμοποίηση του cloud-computing από μεγάλες επιχειρήσεις αυξήθηκε το 2021 κατά 7%, φτάνοντας συνολικά το 72% [6]. Πρόκειται δηλαδή, για μια ευρέως χρησιμοποιούμενη τεχνολογία που διαρκώς κερδίζει έδαφος και χρησιμοποιείται σε ολοένα και περισσότερους τομείς. Η παγκόσμια αυτή αναγνώριση πηγάζει από τα πλεονεκτήματα που προσφέρει το cloud computing έναντι των ιδιόκτητων υποδομών και των παραδοσιακών φυσικών υπολογιστικών συστημάτων, μερικά από τα οποία είναι [7]:

- Ευκολότερη επαναφορά πληροφορίας. Οι εξυπηρετητές του cloud φροντίζουν για τη δημιουργία πολλαπλών αντιγράφων ασφαλείας (back-up data), εξασφαλίζοντας την απρόσκοπτη επαναφορά σε περίπτωση ανάγκης.
- Εξαιρετική προσβασιμότητα. Η πληροφορία που εναποτίθεται στο cloud είναι προσβάσιμη από όπου υπάρχει σύνδεση στο διαδίκτυο.
- Πρακτικά αστείρευτοι πόροι αποθήκευσης. Οι εξυπηρετητές του cloud μπορούν να αποθηκεύσουν έναν πελώριο όγκο πληροφορίας. Σε ατομικό επίπεδο, κάθε χρήστης μπορεί να αποθηκεύσει έγγραφα, εικόνες, βίντεο και οτιδήποτε άλλο επιθυμεί στο cloud, ελευθερώνοντας πόρους αποθήκευσης από τις συσκευές του. Σε επίπεδο επιχειρήσεων, εφαρμογές και καταγραφές συχνά απαιτούν τεράστιο αποθηκευτικό χώρο, καθιστώντας πολλές φορές το cloud τη μόνη βιώσιμη οικονομικά λύση.
- Πρωτότυπες δυνατότητες συνεργασίας και διαχείρισης. Τα περιβάλλοντα cloud προσφέρουν τη δυνατότητα επικοινωνίας μεταξύ των διαφόρων συνεργαζόμενων μελών για την περάτωση κάποιας εργασίας, καθώς όλοι ενεργούν στην ίδια υποδομή, μπορούν να εργαστούν παράλληλα και να έχουν διαυγή εικόνα για το συνολικό έργο κάθε στιγμή.
- Επεκτασιμότητα και ευελιξία. Η χρήση του υπολογιστικού νέφους είναι ιδανική για εταιρείες με αναπτυσσόμενη ή κυμαινόμενη ζήτηση σε πόρους, καθώς μπορούν να ελέγξουν τους διαθέσιμους πόρους από το νέφος χωρίς να επενδύσουν σε ιδιόκτητη φυσική υποδομή.

Η βασική τεχνολογία που καθιστά δυνατή αυτού του είδους την υπολογιστική τεχνική είναι η εικονικοποίηση (Virtualization), την οποία θα εξετάσουμε παρακάτω.

2.1.3 Εικονικοποίηση

Στην επιστήμη των υπολογιστών, εικονικοποίηση ονομάζεται η δημιουργία μιας εικονικής εκδοχής κάποιας φυσικής οντότητας, όπως πλατφόρμες υλικού

υπολογιστών (computer hardware platforms), συσκευές αποθήκευσης και πόροι υπολογιστικών δικτύων (δρομολογητές, μεταγωγείς κλπ.). [8] Πιο συγκεκριμένα, είναι η τεχνολογία που δημιουργεί ένα επίπεδο αφαίρεσης πάνω από το υλικό ενός υπολογιστή και επιτρέπει την κατανομή των πόρων του με τη δημιουργία εικονικών μηχανών. Οι εφαρμογές που "τρέχουν" σε μια τέτοια εικονική μηχανή, δεν αντιλαμβάνονται τα κατώτερα στρώματα και θεωρούν πως βρίσκονται στο δικό τους αφιερωμένο σύστημα με το κατάλληλο λειτουργικό και βιβλιοθήκες.[9]

Η εικονικοποίηση βρίσκει ένα ευρύ φάσμα εφαρμογών. Για τους οικιακούς χρήστες μπορεί να χρησιμοποιηθεί για τη χρήση ενός προγράμματος που προορίζεται για διαφορετικό λειτουργικό σύστημα από αυτό του υπολογιστή τους. Για τους διαχειριστές εξυπηρετητών (server admins), παρέχει τη δυνατότητα μεγαλύτερης χρησιμοποίησης των πόρων του εξυπηρετητή , αφού οι πόροι μπορούν να κατανεμηθούν σε πληθώρα εικονικών μηχανών και να χρησιμοποιηθούν από εφαρμογές με διαφορετικές ανάγκες και προδιαγραφές. Ακόμη, η εικονικοποίηση προσφέρει τη δυνατότητα απομόνωσης των εφαρμογών που "τρέχουν" παράλληλα, αφού οι εικονικές μηχανές που βρίσκονται στον ίδιο οικοδεσπότη (host) λειτουργούν αυτόνομα και δεν αλληλοεπιδρούν. Παρακάτω θα εξετάσουμε τρεις βασικές έννοιες που αφορούν την τεχνολογία της εικονικοποίησης.

2.1.3.1 Hypervisors

Ο hypervisor είναι λογισμικό που επιτρέπει τη δημιουργία και τη διαχείριση εικονικών μηχανών[10]. Το "φυσικό υλικό" (hardware) που χρησιμοποιείται από τον hypervisor ονομάζεται οικοδεσπότης (host), ενώ οι εικονικές μηχανές που χρησιμοποιούν τους πόρους αυτού του υλικού ονομάζονται φιλοξενούμενοι (guests).

Ο hypervisor χειρίζεται τους πόρους - όπως ο επεξεργαστής, η μνήμη RAM και ο αποθηκευτικός χώρος - σαν μια δεξαμενή (pool) από όπου μπορεί εύκολα να αντλήσει και να ανακατανείμει τους πόρους σύμφωνα με τις ανάγκες των υπαρχόντων ή νέων εικονικών μηχανών. Συνεπώς, πολλά λειτουργικά συστήματα μπορούν να "τρέχουν" παράλληλα και να μοιράζονται κοινούς φυσικούς πόρους, αναδεικνύοντας έτσι ένα από τα μεγαλύτερα πλεονεκτήματα της εικονικοποίησης.

Παραδοσιακά, οι hypervisors διακρίνονται σε τύπου 1 και τύπου 2. Οι τύπου 1 (native or bare metal hypervisors), δρουν άμεσα στο υλικό του οικοδεσπότη για την υποστήριξη των φιλοξενούμενων συστημάτων και διαχειρίζονται αυτόνομα τους πόρους προς ανάθεση. Προσφέρουν κατά κανόνα υψηλότερης ποιότητας αποτελέσματα και είναι η συνήθης επιλογή των μεγάλων κέντρων δεδομένων. Από την άλλη, οι τύπου 2 (hosted hypervisors) δραστηριοποιούνται σε επίπεδο λογισμικού ή εφαρμογής, και τα φιλοξενούμενα συστήματα αντιμετωπίζονται ως διαδικασίες του οικοδεσπότη δεσμεύοντας τους πόρους του.

2.1.3.2 Εικονικές μηχανές

Η εικονική μηχανή είναι μια εικονικοποιημένη εκδοχή ενός υπολογιστικού συστήματος που τρέχει "πάνω" από ένα άλλο υπολογιστικό σύστημα. Μια τέτοια

μηχανή αποκτά πρόσβαση σε πόρους του οικοδεσπότη όπως υπολογιστική ισχύς, μνήμη RAM και σκληρός δίσκος, ενώ ακόμη μπορεί να φέρει εικονικές ή φυσικές κάρτες δικτύου καθώς και να συνδεθεί σε περιφερειακές συσκευές.

Λόγω της ευελιξίας και της φορητής φύσης τους, οι εικονικές μηχανές αποκτούν διάφορα πλεονεκτήματα έναντι των παραδοσιακών φυσικών υπολογιστών όπως:[11]

• Μειωμένο κόστος. Συντρέχοντας πολλαπλά εικονικά περιβάλλοντα στο ίδιο φυσικό μηχάνημα, αυξάνεται ο βαθμός χρησιμοποίησης του τελευταίου και μειώνεται η ανάγκη για την αγορά επιπλέον αφιερωμένων φυσικών συστημάτων.

• Ταχύτητα και αποτελεσματικότητα. Η εύκολη και γρήγορη δημιουργία μιας εικονικής μηχανής υπερτερεί για την ανάπτυξη και χρήση εφαρμογών που ειδάλλως θα απαιτούσαν ένα δικό τους αφιερωμένο μηχάνημα.

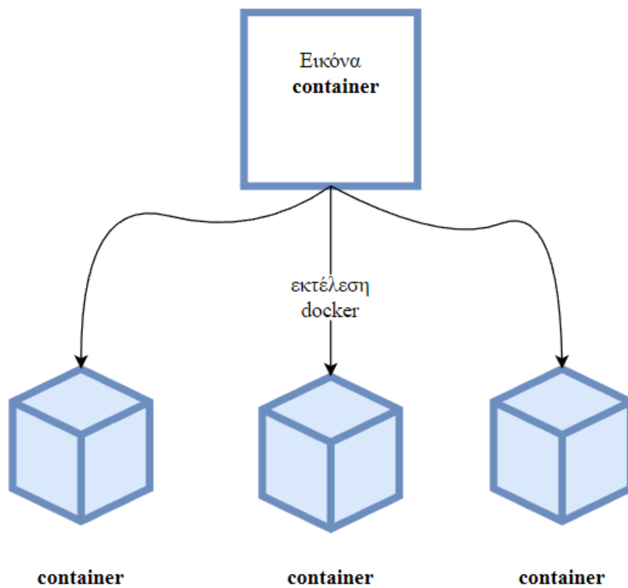
• Μειωμένος "νεκρός χρόνος" και αντοχή σε καταστροφές . Στην περίπτωση βλάβης ενός οικοδεσπότη, η εικονική μηχανή μπορεί άμεσα να μεταφερθεί από έναν hypervisor σε έναν άλλο που τρέχει σε ένα υγιές σύστημα.

• Επεκτασιμότητα. Οι δεσμευόμενοι πόροι επεκτείνονται σύμφωνα με τις ανάγκες των εφαρμογών, ενώ οι διεργασίες μπορούν να κατανεμηθούν σε περισσότερες εικονικές μηχανές.

• Ασφάλεια. Καθώς οι εικονικές μηχανές λειτουργούν αυτόματα και απομονωμένα, ένα επιβλαβές λογισμικό που θα προσβάλλει μια εικονική μηχανή δε θα επηρεάσει τις υπόλοιπες αλλά ούτε και τον οικοδεσπότη.

2.1.3.3 Containers

Τα containers είναι πρότυπες μονάδες λογισμικού που συνδυάζουν κώδικα μαζί με όλες τις απαραίτητες εξαρτήσεις και βιβλιοθήκες ώστε να εξασφαλίσουν τη γρήγορη και απρόσκοπτη λειτουργία μιας εφαρμογής ανεξάρτητα από το υπολογιστικό περιβάλλον [9]. Για να το επιτύχουν, χρησιμοποιούν την εικονικοποίηση επιπέδου λειτουργικού συστήματος, μοιράζονται τον ίδιο πυρήνα και τις βιβλιοθήκες του οικοδεσπότη και δημιουργούν ξεχωριστές απομονωμένες διαδικασίες, κάθε μια εκ των οποίων χρησιμοποιεί πόρους όπως υπολογιστική ισχύς, μνήμη RAM και αποθηκευτικός χώρος. Το βασικό δομικό στοιχείο ονομάζεται "εικόνα container", και μετατρέπεται σε container κατά την εκτέλεση, με τη βοήθεια μιας "μηχανής container".

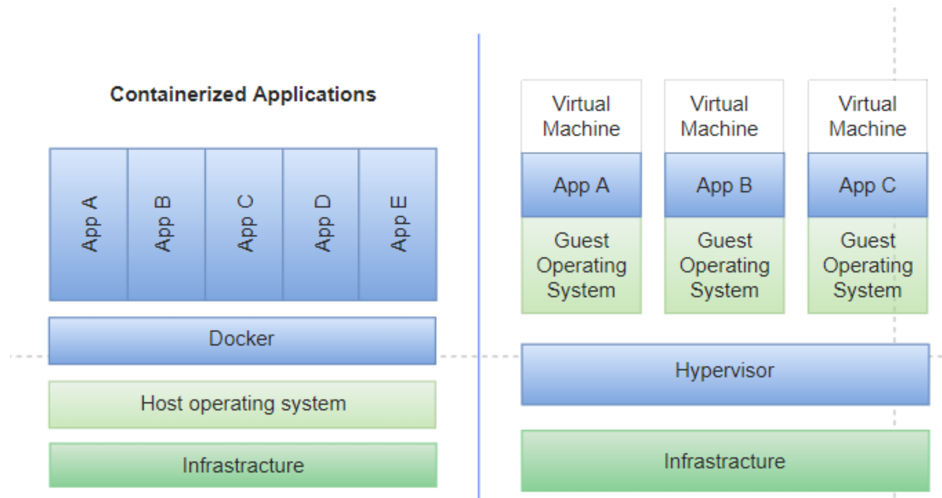


Εικόνα 2-2 Σχέση container και εικόνας container

Οι «εικόνες container» είναι πλήρεις, στατικές και εκτελέσιμες εκδόσεις μιας εφαρμογής ή υπηρεσίας. Συνήθως χρησιμοποιούν μια αρχιτεκτονική διαστρωμάτωσης, όπου τα ανώτερα στρώματα, τα οποία δύναται να διαβαστούν και να επεξεργασθούν (readable/writable), υπερτίθενται στο βασικό στρώμα, το οποίο περιλαμβάνει τις απαραίτητες εξαρτήσεις για την εκτέλεση του κώδικα, και είναι σταθερό και επαναχρησιμοποιήσιμο. Μια εφαρμογή που δομείται με τη χρήση container (containerized application), μπορεί να αποτελείται από πολλές εικόνες, οι οποίες εκτελούνται ανεξάρτητα. Ακόμη, πολλαπλά στιγμιότυπα της ίδιας εικόνας μπορούν να εκτελούνται ταυτόχρονα, ενώ ένα δυσλειτουργικό στιγμιότυπο αντικαθίσταται από ένα νέο χωρίς διακοπή της λειτουργίας της εφαρμογής. Το γεγονός ότι η εικόνα περιέχει όλη την απαραίτητη πληροφορία για την εκτέλεσή της προσφέρει στην τεχνολογία των container τεράστια προσαρμοστικότητα και φορητότητα, καθώς η ίδια εικόνα μπορεί να εκτελεστεί σε μια απομακρυσμένη εικονική μηχανή και μετέπειτα σε ένα ιδιόκτητο υπολογιστικό σύστημα χωρίς καμία αλλαγή στον ενσωματωμένο κώδικα.

Η εκτέλεση εφαρμογών με χρήση containers δημιουργεί την ανάγκη για τη χρήση ενός κεντρικού διαχειριστή, που θα αναλαμβάνει τη δημιουργία και την καταστροφή container, τη δέσμευση των απαραίτητων πόρων και γενικότερα το συντονισμό των δομικών μονάδων για την ορθή λειτουργία της εφαρμογής. Το ρόλο αυτό έχει ο ενορχηστρωτής container, που θα εξετάσουμε αναλυτικότερα σε επόμενο κεφάλαιο καθώς αφορά άμεσα το πρόβλημα που εξετάζει η παρούσα εργασία. Συγκεκριμένα, θα εστιάσουμε στον αλγόριθμο που ενσωματώνουν τα συστήματα ενορχηστρωτών για τη δέσμευση πόρων, την αντιστοίχιση δηλαδή των εικόνων container σε φυσικά ή εικονικά μηχανήματα που θα αναλάβουν την εκτέλεσή τους, και θα εντοπίσουμε τις αδυναμίες των ήδη υπαρχόντων ενορχηστρωτών, προτού αναπτύξουμε τη δική μας προσέγγιση.

2.1.3.4 Σύγκριση container και εικονικών μηχανών



Εικόνα 2-3 Σύγκριση container και εικονικών μηχανών

Containers: Τα containers δημιουργούν αφαίρεση στο επίπεδο εφαρμογής, δηλαδή η εικονικοποίηση συμβαίνει στο στρώμα λογισμικού πάνω από το λειτουργικό σύστημα του οικοδεσπότη. Πολλά container μπορούν να εκτελούνται συγχρόνως καθώς μοιράζονται τον ίδιο πυρήνα, με το καθένα να παρουσιάζεται ως ξεχωριστή διαδικασία στο πεδίο του χρήστη. Είναι γενικά πολύ ελαφρύτερα (μικρότερου μεγέθους, της τάξεως των δεκάδων MB) και γρηγορότερα από τις εικονικές μηχανές, ενώ δεν υπάρχει ανάγκη για την ύπαρξη πολλαπλών λειτουργικών συστημάτων.

Εικονικές μηχανές: Οι εικονικές μηχανές δημιουργούν αφαίρεση στο επίπεδο φυσικού υλικού, δηλαδή η εικονικοποίηση συμβαίνει μέχρι τα κατώτερα στρώματα του υλικού (hardware). O hypervisor επιτρέπει πολλές εικονικές μηχανές να τρέχουν σε μια φυσική. Κάθε εικονική μηχανή είναι μια πλήρης εξομοίωση λειτουργικού συστήματος μαζί με βιβλιοθήκες και δυαδικά αρχεία, καταναλώνοντας αποθηκευτικό χώρο της τάξεως των πολλών GB, συνιστώντας άρα μια πιο εύρωστη αλλά δύσκαμπτη τεχνολογία.

2.1.4 Είδη υπηρεσιών υπολογιστικού νέφους

Οι περισσότερες υπηρεσίες νέφους εμπίπτουν σε μια εκ των τριών κατηγοριών: Υποδομή ως Υπηρεσία (Infrastructure as a Service or IaaS), Πλατφόρμα ως υπηρεσία (Platform as a Service or PaaS) και Λογισμικό ως Υπηρεσία (Software as a Service or SaaS). Πολλές φορές οι κατηγορίες αυτές ονομάζονται "στοίβα υπολογιστικού νέφους" διότι περιλαμβάνουν διαφορετικά, διαδοχικά επίπεδα εικονικοποίησης.

Παρακάτω θα εξετάσουμε ενδελεχώς κάθε μια κατηγορία καθώς συχνά υπάρχει σύγχυση μεταξύ των χαρακτηριστικών τους.

2.1.4.1 Υποδομή ως Υπηρεσία (IaaS)

Η Υποδομή ως Υπηρεσία προσφέρει κατά παραγγελία πρόσβαση σε θεμελιώδεις πόρους όπως φυσικούς ή εικονικούς servers, δικτυακές συσκευές και αποθηκευτικό χώρο μέσω του διαδικτύου, σε ένα διανεμητικό (pay-as-you-go) μοντέλο [13]. Παρατηρούμε ότι η περιγραφή αυτή προσομοιάζει σε μεγάλο βαθμό αυτή του ίδιου του cloud computing, καθώς ήταν το πρώτο και για πολλά χρόνια (έως το 2010) το επικρατέστερο μοντέλο. Με την υιοθεσία της Υποδομής ως Υπηρεσία, οι χρήστες έχουν τη δυνατότητα να μεγεθύνουν ή να συρρικνώσουν τους χρησιμοποιούμενους πόρους σύμφωνα με τις ανάγκες τους, μειώνοντας έτσι τα μεγάλα εναρκτήρια κόστη και το ρίσκο που αυτά επιφέρουν. Η ευελιξία του μοντέλου είναι τέτοια, που ακόμη και σε περιόδους απότομης αύξησης της ζήτησης σε πόρους (spikes), δεν απαιτείται κάποια επιπλέον ιδιόκτητη υποδομή. Η βασική τεχνολογία που χρησιμοποιείται σε αυτό το μοντέλο είναι οι εικονικές μηχανές. Όπως θα δούμε παρακάτω, η Υποδομή ως Υπηρεσία παρέχει στους χρήστες το κατώτατο (μεγαλύτερο) επίπεδο ελέγχου.

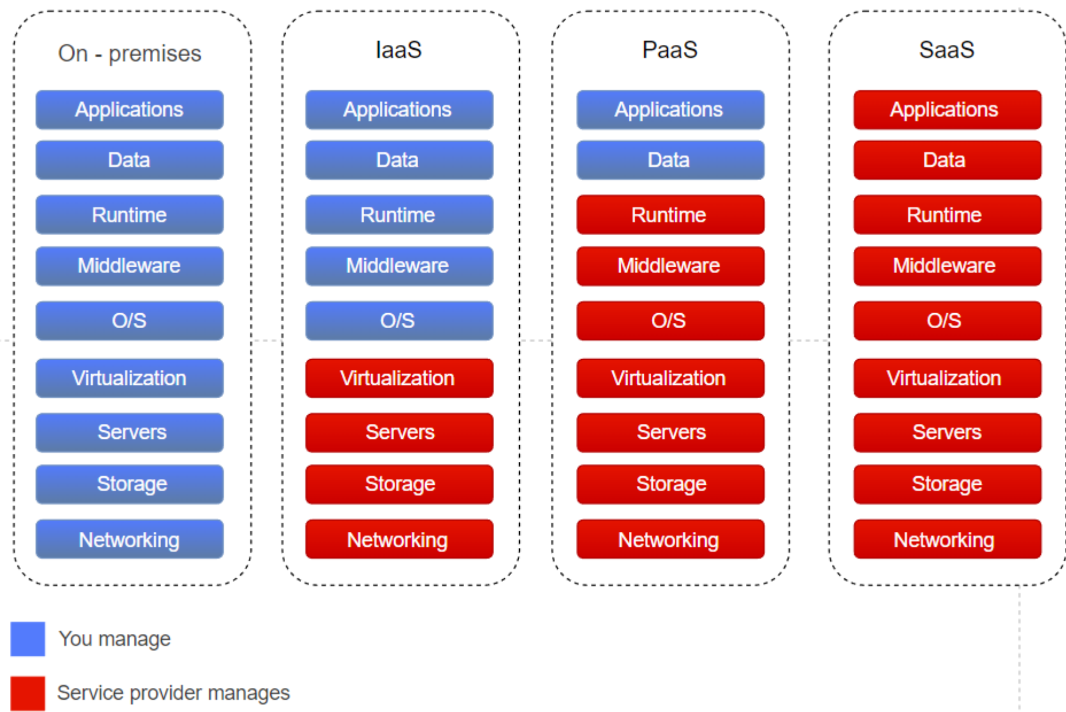
2.1.4.2 Πλατφόρμα ως Υπηρεσία (PaaS)

Η Πλατφόρμα ως Υπηρεσία είναι ένα πλήρως εξοπλισμένο περιβάλλον για προγραμματισμό και ανάπτυξη εφαρμογών.[14] Όπως και στην Υποδομή ως Υπηρεσία, συμπεριλαμβάνονται όλοι οι βασικοί πόροι όπως servers, αποθηκευτικός χώρος και δικτυακές συσκευές, αλλά επιπλέον παρέχονται είδη μεσαίου λογισμικού (middleware), εργαλεία προγραμματισμού, συστήματα διαχείρισης βάσεων δεδομένων κλπ. , προσβάσιμα μέσω του διαδικτύου. Όλα τα παραπάνω ανήκουν στον πάροχο υπηρεσιών νέφους και ελέγχονται από αυτόν, ενώ οι χρήστες απλώς επιλέγουν από ένα κατάλογο τους servers και το περιβάλλον για να σχεδιάσουν, να αναπτύξουν, να τεστάρουν, να διατηρήσουν και να επικαιροποιήσουν τις εφαρμογές τους. Η βασική τεχνολογία που χρησιμοποιείται σε αυτό το μοντέλο είναι τα containers.

2.1.4.3 Λογισμικό ως Υπηρεσία (SaaS)

Το Λογισμικό ως Υπηρεσία, γνωστό και ως λογισμικό με βάση το νέφος (cloud-based software), είναι λογισμικό εφαρμογών που φιλοξενείται στο νέφος και είναι προσβάσιμο από τους χρήστες μέσω ενός προγράμματος περιηγητή ή μιας διεπαφής προγραμματισμού εφαρμογών (API) που ενσωματώνεται στο λειτουργικό σύστημα του υπολογιστή ή της κινητής συσκευής του χρήστη. Όλη η υποκείμενη υποδομή, το λογισμικό της εφαρμογής και τα δεδομένα βρίσκονται στη βάση του παρόχου, και έτσι ο χρήστης απαλλάσσεται από την ανάγκη επικαιροποίησης της εφαρμογής (updates) και την αποθήκευση δεδομένων σε δικιά του συσκευή. Παραδείγματα του Λογισμικού ως Υπηρεσία είναι οι περισσότερες των γνωστών εμπορικών εφαρμογών σήμερα, όπως υπηρεσίες ηλεκτρονικού ταχυδρομείου, δικτυακές εφαρμογές Office αλλά και το Netflix.

2.1.4.4 Σύγκριση επιπέδου ελέγχου από το χρήστη μεταξύ των υπηρεσιών νέφους



Εικόνα 2-4 Επίπεδα ελέγχου χρήστη [15]

Όπως παρατηρούμε στην εικόνα, στα ιδιόκτητα συστήματα ο χρήστης προφανώς έχει τον πλήρη έλεγχο σε όλα τα επίπεδα. Στην Υποδομή ως Υπηρεσία, οι εξυπηρετητές, ο αποθηκευτικός χώρος, η δικτύωση και η εικονικοποίηση αυτών επαφίεται στον πάροχο υπηρεσιών νέφους. Στην Πλατφόρμα ως Υπηρεσία, ο χρήστης έχει έλεγχο μόνο στις εφαρμογές που προγραμματίζει και στα δεδομένα που αποθηκεύει/παράγει, ενώ στο Λογισμικό ως Υπηρεσία ο πάροχος έχει τον πλήρη έλεγχο σε όλα τα επίπεδα με το χρήστη απλώς να χρησιμοποιεί την εκάστοτε εφαρμογή.

2.1.5 Τα είδη του νέφους

2.1.5.1 Δημόσιο Νέφος

Στο Δημόσιο Νέφος ο πάροχος υπηρεσιών νέφους διαθέτει τους πόρους του - οτιδήποτε από Λογισμικό ως Υπηρεσία, εικονικές μηχανές και φυσικό υπολογιστικό υλικό- μέσω του δημοσίου ίντερνετ [16]. Ο πάροχος διατηρεί την κυριότητα και αναλαμβάνει τη διαχείριση και τη διατήρηση των υποδομών, ενώ συχνά "εξοπλίζει" τους πελάτες με υψηλής ταχύτητας συνδέσεις για τη γρήγορη και απρόσκοπτη προσπέλαση των δεδομένων και των εφαρμογών τους. Πρόκειται για ένα περιβάλλον όπου "συνυπάρχουν" πολλοί και ετερόκλητοι πελάτες που μοιράζονται τους πόρους του παρόχου, με αποτέλεσμα να εγείρονται ζητήματα ασφάλειας και ακεραιότητας, γεγονός που οδήγησε στη δημιουργία των δύο άλλων μοντέλων που θα εξετάσουμε παρακάτω.

2.1.5.2 Ιδιωτικό Νέφος

Η υποδομή του Ιδιωτικού Νέφους λειτουργεί αποκλειστικά για έναν οργανισμό, είτε εδράζεται και διαχειρίζεται "εκ των έσω", είτε από κάποιον "τρίτο" (third party). Συνήθως επιλέγεται από πελάτες που αποθηκεύουν και διαχειρίζονται ευαίσθητα δεδομένα, και άρα η ασφάλεια τους είναι πρωταρχικής σημασίας. Το μοντέλο αυτό έχει δεχτεί κριτική στην περίπτωση που δημιουργείται εντός των υποδομών μιας επιχείρησης, καθώς στερείται το βασικό οικονομικό πλεονέκτημα του υπολογιστικού νέφους, που είναι η εξωτερική ανάθεση για τη διαχείριση και συντήρηση των πόρων.

2.1.5.3 Υβριδικό Νέφος

Το υβριδικό νέφος, όπως μαρτυρά ο όρος, είναι ένα οργανωτικό μοντέλο που συνδυάζει το δημόσιο και το ιδιωτικό νέφος, τα οποία παραμένουν ξεχωριστές οντότητες αλλά υπάρχει ένα επίπεδο ενορχήστρωσης ανάμεσά τους, που δίνει στον εκάστοτε οργανισμό την ευελιξία να επιλέξει το κατάλληλο νέφος για την κάθε εφαρμογή ή ακόμα και να μεταφέρει φόρτο εργασίας από το ένα περιβάλλον στο άλλο. Για παράδειγμα, μια εταιρεία μπορεί να αποθηκεύει και να επεξεργάζεται τα ευαίσθητα δεδομένα στο ιδιωτικό νέφος των υποδομών της και παράλληλα να χρησιμοποιεί εφαρμογές δημοσίου νέφους για άλλες λειτουργίες. Επιπλέον, μια τεχνική ονόματι "cloud bursting" εκμεταλλεύεται την αρχιτεκτονική του υβριδικού νέφους ως εξής: Χρησιμοποιεί τις ιδιόκτητες υποδομές με το ιδιωτικό νέφος σε συνθήκες κανονικής λειτουργίας, και μεταφέρει φόρτο εργασίας σε δημόσιο νέφος σε περιόδους αυξημένης ζήτησης.

2.2 Edge Computing

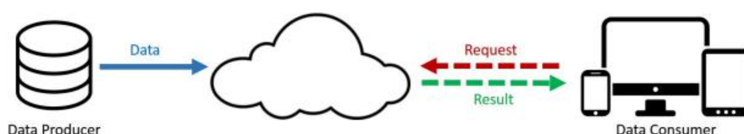
2.2.1 Internet of Things

Ο όρος του "'Ιντερνετ των Πραγμάτων" εισήχθη στην κοινότητα το 1999 για τη διαχείριση της εφοδιαστικής αλυσίδας επιχειρήσεων, αλλά σύντομα η έννοια διευρύνθηκε και υιοθετήθηκε από πολλά πεδία όπως η ιατρική, η τεχνολογία περιβάλλοντος, οι μεταφορές, ο οικισμός κλπ.[17] Στις μέρες μας πρόκειται για μια σύνθετη και συγκεχυμένη έννοια, θα μπορούσαμε όμως να πούμε πως το "'Ιντερνετ των Πραγμάτων" περιγράφει φυσικά αντικείμενα (ή ομάδες φυσικών αντικειμένων) με ενσωματωμένους αισθητήρες, επεξεργαστές, λογισμικό και άλλες τεχνολογίες που επιτρέπουν την επικοινωνία με άλλες συσκευές και συστήματα μέσω του ίντερνετ, χωρίς την ανθρώπινη παρέμβαση. Για παράδειγμα, ένας "έξυπνος" θερμοστάτης (με τον όρο έξυπνος να υποδηλώνει πως πρόκειται για IoT συσκευή) θα λαμβάνει δεδομένα για την τοποθεσία του κατοίκου από το αυτοκίνητό του, και θα ρυθμίζει το σύστημα θέρμανσης ούτως ώστε κατά τη στιγμή της άφιξης το σπίτι να έχει την επιθυμητή θερμοκρασία.

2.2.2 Η ανεπάρκεια του cloud

Εικόνα 2-5 Ρυθμός ανάπτυξης IoT συσκευών [18]

Με την αλματώδη τεχνολογική πρόοδο, ολοένα και περισσότερες ηλεκτρονικές συσκευές της καθημερινότητας εκσυγχρονίζονται και υπάγονται πλέον στο παράδειγμα του Ίντερνετ των Πραγμάτων. Ο αριθμός τους ήδη κυμαίνεται στις δεκάδες δισεκατομμυρίων και πληθαίνει συνεχώς, ενώ ο παραγόμενος όγκος δεδομένων υπολογίζεται πως θα προσεγγίσει τα 175 zettabytes [19], σημειώνοντας αύξηση 61%. Η εναπόθεση όλου του υπολογιστικού φόρτου στο νέφος έχει αποδειχθεί στο παρελθόν μια αποτελεσματική μέθοδος, αφού η ισχύς των βάσεων του νέφους είναι μακράν μεγαλύτερη από την υπολογιστική ισχύ των συσκευών στα άκρα.



Εικόνα 2-6 Το παραδοσιακό μοντέλο cloud

Ωστόσο, καθώς ο παραγόμενος όγκος αυξάνεται με εκθετικούς ρυθμούς, προκύπτουν οι εξής, μείζονος σημασίας, περιοριστικοί παράγοντες:

• Εύρος ζώνης (Bandwidth). Το εύρος ζώνης ορίζεται ως η ποσότητα πληροφορίας που μπορεί ένα δίκτυο να μεταφέρει στη μονάδα του χρόνου, συνήθως εκφραζόμενο σε bps. Όταν όλη η πληροφορία μεταφέρεται στο στρώμα του cloud, δημιουργούνται φαινόμενα συμφόρησης (network bottleneck) στα ανώτερα δικτυακά στρώματα (MAN,WAN) εισάγοντας επιπρόσθετες καθυστερήσεις ή ακόμη και απώλεια σύνδεσης, που οδηγεί στην απόρριψη αιτημάτων (blocking).

• Καθυστέρηση (Latency). Η καθυστέρηση ορίζεται ως ο απαιτούμενος χρόνος για τη μεταφορά δεδομένων μεταξύ δύο σημείων ενός δικτύου. Ο χρόνος αυτός προσαυξάνεται για κάθε άλμα (hop) μεταξύ δικτυακών συσκευών που παρεμβάλλονται (π.χ. δρομολογητές) καθώς και από αστάθμητους παράγοντες

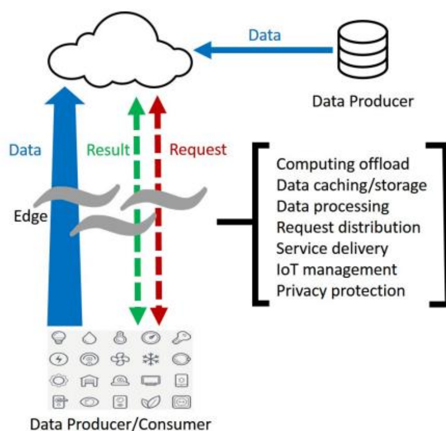
όπως η δικτυακή συμφόρηση ή ακόμα και η απώλεια σύνδεσης. Επομένως, η καθυστέρηση για τη μεταφορά όλων των δεδομένων στους απομακρυσμένους εξυπηρετητές του νέφους δυσχεραίνει τη λειτουργία των εφαρμογών που απαιτούν λήψη αποφάσεων σε πραγματικό χρόνο.

Για παράδειγμα, ένα αυτόνομο όχημα παράγει δεδομένα της τάξεως του 1Gb/sec, μέρος των οποίων απαιτεί άμεση επεξεργασία και άρα η μεταφορά του στο νέφος θα παρήγαγε απαγορευτικούς χρόνους απόκρισης, ενώ η συμφόρηση του δικτύου σε περιοχές μεγάλης κυκλοφοριακής πυκνότητας θα προκαλούσε σημαντική υποβάθμιση της παρεχόμενης ποιότητας υπηρεσιών (Quality of Service – QoS).

Σημειώνεται πως οι εφαρμογές του IoT δεν είναι οι μόνες που επιβαρύνουν τα άκρα του δικτύου. Ακόμη και εφαρμογές που απευθύνονται στο μέσο χρήστη, όπως η εικονική ή η επαυξημένη πραγματικότητα και η ζωντανή μετάδοση βίντεο υψηλής ποιότητας, παρουσιάζουν μεγάλη ευαισθησία στο χρόνο απόκρισης και δημιουργούν τεράστια δικτυακή κίνηση. Σε τέτοιες περιπτώσεις όπου κάποιος πληροφοριακός όγκος χρήζει άμεσης επεξεργασίας κρίνεται απαραίτητη η ανάπτυξη μιας νέας αρχιτεκτονικής, όπου τα δεδομένα επεξεργάζονται στο άκρο του δικτύου και άρα οι χρόνοι απόκρισης συρρικνώνονται, ενώ μόνο η πληροφορία που χρειάζεται μεγαλύτερη επεξεργασία, για παράδειγμα για την διεξαγωγή στατιστικής μελέτης, διακομίζεται στο cloud. Συνεπώς, με την μερική επεξεργασία στα άκρα, το δίκτυο ανακουφίζεται και η ποιότητα των παρεχόμενων υπηρεσιών βελτιώνεται.

2.2.3 Ορισμός του προτύπου Edge Computing

Το edge computing **[20]**είναι ένα κατανεμημένο υπολογιστικό παράδειγμα που, σε αντίθεση με το cloud computing, φέρνει την επεξεργασία και την αποθήκευση κοντινότερα στην πηγή παραγωγής των δεδομένων, εκμεταλλευόμενο τις συσκευές στα άκρα όπως τα κινητά τηλέφωνα, συσκευές IoT ή ακόμη και δικτυακές πύλες ώστε να περατώσει εργασίες και να προσφέρει υπηρεσίες. Πρόκειται περισσότερο για μια λογική αρχιτεκτονικής παρά για κάποια συγκεκριμένη τεχνολογία.



Εικόνα 2-7 Το μοντέλο του Edge Computing

Η εικόνα παρουσιάζει τα αμφίδρομα επικοινωνιακά κανάλια στο edge computing. Σε αντίθεση με το παράδειγμα του cloud, οι IoT συσκευές δρουν παράλληλα ως καταναλωτές και παραγωγοί δεδομένων, ζητώντας υπηρεσίες και περιεχόμενο από το cloud αλλά συγχρόνως εκτελώντας εργασίες για αυτό. Οι συσκευές στα άκρα μπορούν να κάνουν υπολογιστική εκφόρτωση (computing offloading) στο νέφος, να αποθηκεύσουν (σε δίσκο ή και σε μνήμη cache) και να επεξεργαστούν δεδομένα, να επωμισθούν μέρος των αιτημάτων του χρήστη και των υπηρεσιών προς αυτόν κλπ.

Κατά αυτόν τον τρόπο, αξιοποιείται στο έπακρο το εύρος ζώνης των τοπικών δικτύων (LAN's) από τις τοπικές IoT συσκευές, οδηγώντας στην αποσυμφόρηση των υπόλοιπων τμημάτων του δικτύου και ελαχιστοποιώντας την καθυστέρηση. Η τοπική αποθήκευση "προστατεύει" την ακατέργαστη πληροφορία από την πιθανότητα απώλειας και επιτρέπει την εν μέρει επεξεργασία στα άκρα, μειώνοντας έτσι και την ποσότητα δεδομένων που τελικά μεταφέρεται στο cloud. Επιπρόσθετα, η απλή μεταφορά δεδομένων από την πηγή παραγωγής στο cloud πολλές φορές εγείρει νομικά ζητήματα και προβλήματα ασφαλείας, ειδικά όταν διασχίζονται εθνικά όρια. Το edge computing μπορεί να αναλάβει την επισκίαση (obscuring) ή την κρυπτογράφηση της πληροφορίας πρώτού αυτή σταλθεί στο απομακρυσμένο κέντρο δεδομένων.

2.2.4 Mobile Edge Computing (MEC)

2.2.4.1 Υπηρεσίες Δικτύων 5ης γενιάς (5G)

Το 5G είναι η αρχιτεκτονική 5ης γενιάς για την κινητή επικοινωνία βασισμένη στην τεχνολογία κυψελών.[21] Το κίνητρο για τη δημιουργία του 5G μπορεί να αναζητηθεί σε πληθώρα παραγόντων, ο κυριότερος εκ των οποίων είναι η αδυναμία διαχείρισης της διαρκώς αυξανόμενης δικτυακής κίνησης από την προηγούμενη τεχνολογία του 4G και η ανάγκη για υψηλότερες ταχύτητες και μειωμένη καθυστέρηση. Το 5G επιτρέπει έως και 10 φορές ταχύτερη ροή δεδομένων (σε σχέση με τον προκάτοχό του 4G), υποστηρίζει τεράστια πυκνότητα χρηστών (κυψέλης), προσφέρει βελτιωμένη ποιότητα υπηρεσιών (QoS) ενώ ταυτόχρονα οι χρήστες απολαμβάνουν σημαντικά μειωμένη καθυστέρηση. Οι παρεχόμενες υπηρεσίες μπορούν να ενταχθούν σε μια εκ των κάτωθι τριών κατηγοριών:

- Ακραία κινητή ευρυζωνική σύνδεση (extreme mobile broadband – eMBB). Αφορά το μέσο χρήστη, στον οποίο προσφέρεται υψηλότερη ταχύτητα και εύρος ζώνης, με μέτρια καθυστέρηση. Καθίσταται δυνατή η ζωντανή μετάδοση σε υψηλή ανάλυση (UltraHD streaming), η χρήση τεχνολογιών επαυξημένης ή εικονικής πραγματικότητας (AR-VR media) και πολλά άλλα.

- Τεράστια επικοινωνία μηχανών (massive machine type communication – MMTC). Παρέχεται μεγάλου εύρους επικοινωνία μεταξύ μηχανών (που φέρουν υπολογιστικά συστήματα, σένσορες, ενεργοποιητές κλπ.) με μειωμένο κόστος και βελτιωμένη ενεργειακή κατανάλωση.

- Πολύ αξιόπιστη επικοινωνία χαμηλής καθυστέρησης (ultra-reliable low latency communication – URLLC). Αφορά ευαίσθητες εφαρμογές πραγματικού χρόνου, όπως η τηλέ-χειρουργική και η επικοινωνία οχημάτων (V2V).

Παρέχεται χαμηλή καθυστέρηση και μεγάλη αξιοπιστία, δίνοντας έτσι βελτιωμένη ποιότητα υπηρεσιών (QoS) σε σχέση με το παραδοσιακό μοντέλο κινητής δικτύωσης.

Παρατηρούμε ότι μεγάλο μέρος των υπηρεσιών του 5G αφορά στη βελτίωση της ποιότητας υπηρεσιών, που με άλλα λόγια σημαίνει την ικανοποίηση των προδιαγραφών των συγκεκριμένων εφαρμογών, ιδίως στον τομέα της καθυστέρησης (latency) και της αξιοπιστίας. Καθίσταται λοιπόν σαφής ο εξέχοντας ρόλος του edge computing, που εκπροσωπείται επίσημα στην αρχιτεκτονική των δικτύων 5G από το Mobile edge computing, μια τεχνολογία-κλειδί για τη συνολική ευημερία του IoT περιβάλλοντος σε συνδυασμό με τις ανάγκες των χρηστών.

2.2.4.2 Το πρότυπο του MEC

Σε αντίθεση με το edge computing, το Mobile Edge Computing, αν και συναφές στον ορισμό, είναι μια σαφώς καθορισμένη και προτυποποιημένη από τον Ευρωπαϊκό Οργανισμό Τηλεπικοινωνιών (ETSI) τεχνολογία. Σύμφωνα με τον τελευταίο [22], το MEC προσφέρει ένα προγραμματιστικό περιβάλλον καθώς και δυνατότητες cloud computing στα άκρα, εντός του εκάστοτε δικτύου ασύρματης πρόσβασης (RAN) και σε εγγύτητα με τους συνδρομητές κινητής επικοινωνίας. Βασισμένο σε μια εικονικοποιημένη πλατφόρμα, το MEC αναγνωρίζεται από το ευρωπαϊκό 5GPPP (5G Public Private Partnership) ως μια από τις τεχνολογίες-πυλώνες του 5G.

Είναι δηλαδή η πραγμάτωση της ιδέας του edge computing (τουλάχιστον σε μια έκφανσή της) , που θα προσδώσει δυνατότητες επεξεργασίας, αποθήκευσης και δικτύωσης στους σταθμούς βάσης, που μέχρι πρότινος απλώς μεταβίβαζαν τα αιτήματα των χρηστών μέσω του βασικού δικτύου (backbone) στους απομακρυσμένους cloud servers.

Παρακινούμενο από την προαναφερθείσα ανεπάρκεια του cloud , σκοπός του MEC είναι να αναδιαρθρώσει τη δικτύωση , προσφέροντας μείωση του χρόνου απόκρισης, αυτοματοποίηση, κλιμάκωση, εγγύηση αποτελεσματικής δικτύωσης και παροχής υπηρεσιών, και συνολική βελτίωση της εμπειρίας του χρήστη.

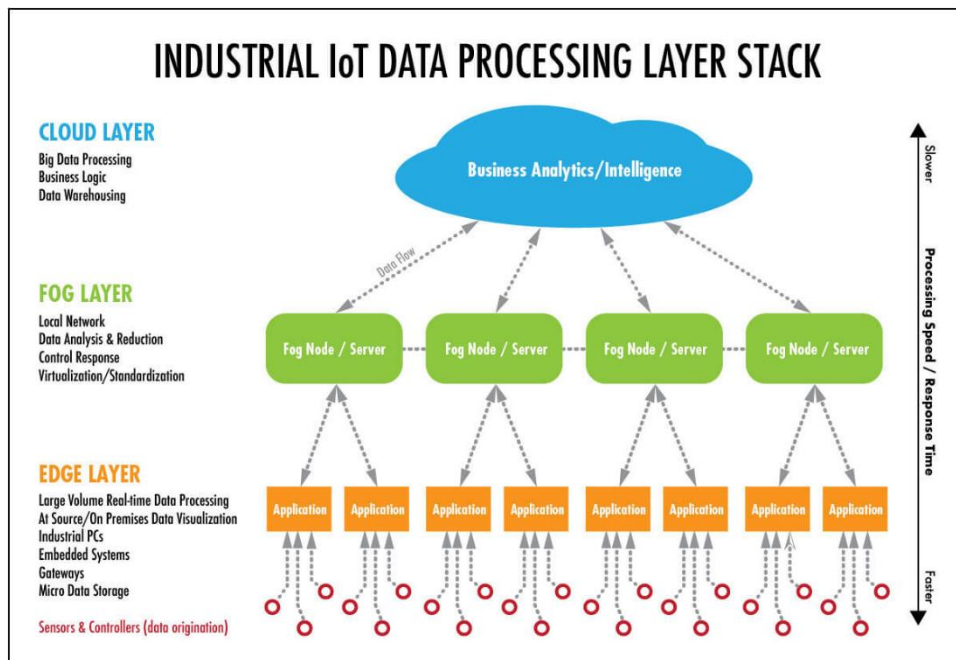
Τα κύρια χαρακτηριστικά του MEC είναι τα εξής:

- Εγγύτητα. Οι MEC εξυπηρετητές τοποθετούνται στην πλησιέστερη τοποθεσία σε σχέση με τους τελικούς χρήστες (εντός κυψέλης).
- Χαμηλή καθυστέρηση. Τα δεδομένα χρόνο-κρίσιμων εφαρμογών δεν προωθούνται στο δίκτυο κορμού, και άρα οι χρήστες ή τα IoT συστήματα απολαμβάνουν ελάχιστη καθυστέρηση και υψηλή ταχύτητα.
- Επίγνωση τοποθεσίας. Οι κόμβοι του MEC λαμβάνουν πληροφορία από κοντινές συσκευές στα άκρα και γνωρίζουν τη γεωγραφική θέση τους.
- Ενημέρωση κατάστασης του δικτύου. Με χρήση του MEC, λαμβάνονται οι απαραίτητες πληροφορίες ώστε να προβλεφθεί και να αντιμετωπιστεί η ενδο-

κυψελική συμφόρηση , καθιστώντας το μια ιδιαίτερα χρήσιμη τεχνολογία για τους τηλεπικοινωνιακούς παρόχους.

2.3 Fog Computing

Η έννοια του fog computing συχνά συγχέεται με αυτή του edge computing, καθώς και τα δύο παραδείγματα φέρουν τον ίδιο "αποκεντρωτικό" χαρακτήρα, αξιοποιώντας τα άκρα του δικτύου για την επεξεργασία, την αποθήκευση και τη δικτύωση. Θα μπορούσαμε να πούμε πως το fog computing εδράζεται ανάμεσα στο cloud και το edge (εντός των ευρύτερων δικτύων MAN/WAN), προσφέροντας υποδομές για την υποστήριξη υπηρεσιών κοντινότερα στα άκρα του δικτύου ονόματι κόμβοι ομίχλης (fog nodes). Οι κόμβοι αυτοί μπορεί να είναι μηχανήματα δικτύωσης όπως δρομολογητές, μεταγωγείς, σταθμοί βάσης και σημεία πρόσβασης (access points) αλλά και ισχυροί υπολογιστές όπως τα cloudlets [23]. Ακόμη, συχνά συναντώνται ειδικού σκοπού μηχανήματα για την "επιτάχυνση υλικού" (hardware acceleration)[24], όπως επεξεργαστές γραφικών (GPUs) και ολοκληρωμένα συστήματα ειδικών εφαρμογών.(ASICs)



Εικόνα 2-8 Στρώματα IoT [25]

Από την οπτική της Cisco [26], το fog computing θεωρείται μια επέκταση του cloud computing στα άκρα του δικτύου (εξού και η ονομασία fog, καθώς η ομίχλη είναι ένα σύννεφο κοντινότερα στο έδαφος). Είναι μια ως επί το πλείστο εικονικοποιημένη αρχιτεκτονική που παρέχει επεξεργαστική ισχύ, αποθηκευτικό χώρο και υπηρεσίες δικτύωσης ανάμεσα στο cloud και τις τερματικές συσκευές. Το fog δεν υποκαθιστά το cloud , αλλά κυρίως το συμπληρώνει, καθώς οι δυο αυτές τεχνολογίες συχνά ενορχηστρώνονται μεταξύ τους και συνεργάζονται για την περάτωση εργασιών.

Τα βασικότερα χαρακτηριστικά του fog computing - εστιάζοντας στις διαφορές από

το cloud και αναδεικνύοντας τα πλεονεκτήματα της δράσης του στα άκρα του δικτύου- είναι τα εξής: [27]

• Χαμηλή καθυστέρηση και επίγνωση τοποθεσίας. Το παράδειγμα του fog computing οικοδομήθηκε για να υποστηρίξει απαιτητικές (από άποψη κατανάλωσης πόρων) εφαρμογές στα άκρα, που συχνά απαιτούν χαμηλή καθυστέρηση (διαδικτυακά παιχνίδια, ζωντανή ροή βίντεο, εικονική πραγματικότητα κλπ.).

• Ευρεία γεωγραφική κατανομή. Σε οξεία αντίθεση με την κεντρικοποιημένη προσέγγιση του cloud, οι εφαρμογές που υποστηρίζει το fog απαιτούν ευρέως κατανεμημένες πηγές πόρων. Για παράδειγμα, στην περίπτωση των «έξυπνων» οχημάτων, διακομιστές μεσολάβησης (proxy servers) και σημεία πρόσβασης (access points) θα τοποθετούνται ανά τακτά διαστήματα στο οδικό δίκτυο.

• Ευκινησία. Σε πολλές εφαρμογές κρίνεται απαραίτητη η άμεση επικοινωνία με τις τερματικές συσκευές, και άρα υιοθετούνται ανάλογες τεχνικές , όπως το πρωτόκολλο LISP μέσω του οποίου αποπροσαρτάται η ταυτότητα της συσκευής από την τοποθεσία της (τη θέση της στο δίκτυο).

• Εξέχοντας ρόλος της ασύρματης διασύνδεσης. Στην επερχόμενη IoT πραγματικότητα, συσκευές και σένσορες θα τοποθετούνται σε ποικίλα περιβάλλοντα, πολλές φορές δυσπρόσιτα και αφιλόξενα για καλωδίωση. Συνεπώς, η πλειονότητα της δικτυακής κίνησης θα συμβαίνει ασύρματα.

• Ανομοιογένεια ως προς τις υπηρεσίες. Οι fog κόμβοι έχουν ποικιλία μορφών και δυνατοτήτων ώστε να ανταπεξέλθουν στις ετερόκλητες ανάγκες των εφαρμογών σε ποικίλα περιβάλλοντα.

• Υποστήριξη on-line ανάλυσης και αλληλεπίδρασης με το cloud. Πολλές εφαρμογές απαιτούν τόσο την άμεση επεξεργασία με τη χρήση του edge και του fog, όσο και αναλυτικότερη μελέτη για την εξαγωγή στατιστικών και "BIG DATA" με τη βοήθεια του πολύ ισχυρότερου cloud , Συνεπώς πρέπει να υπάρχει διαρκής και απρόσκοπτη επικοινωνία των δύο υποδομών.

2.3.1 Cloudlets

Το cloudlet είναι ένα αποκεντρωμένο, μικρής κλίμακας κέντρο δεδομένων με ενισχυμένη κινητικότητα.**[28]** Κύριος σκοπός του είναι η υποστήριξη εφαρμογών που απαιτούν τόσο την δια-δραστικότητα σε πραγματικό χρόνο όσο και την έντονη επεξεργασία, προσφέροντας ισχυρούς πόρους στις κινητές συσκευές σε συνδυασμό με χαμηλή καθυστέρηση. Θα μπορούσαμε να πούμε πως αρχιτεκτονικά εδράζεται στο ευρύτερο πεδίο του fog, αποσυμφορίζοντας τα δίκτυα ευρείας Ζώνης (WAN's). Όπως και στα παραδοσιακά κέντρα δεδομένων του νέφους, η κύρια τεχνολογία που χρησιμοποιείται για την περάτωση εργασιών είναι οι εικονικές μηχανές, ωστόσο τα cloudlets παρουσιάζουν τις εξής διαφορές :

• Η εξυπηρέτηση συμβαίνει δυναμικά και εξαρτάται από την κινητικότητα και την τοποθεσία των χρηστών, οπότε οι εικονικές μηχανές πρέπει να προσαρμόζονται και να ενεργοποιούνται πολύ ταχύτερα.

• Όταν ένας χρήστης απομακρύνεται από την περιοχή κάλυψης του cloudlet, η ποιότητα των υπηρεσιών μειώνεται σταδιακά καθώς αυξάνεται η λογική απόσταση του δικτύου. Για την αντιμετώπιση του προβλήματος αυτού, τα cloudlets ενορχηστρώνονται για την βέλτιστη εξυπηρέτηση των πελατών, υιοθετώντας τεχνικές "μετανάστευσης" εικονικών μηχανών στο WAN δίκτυο. (VM hand-off).

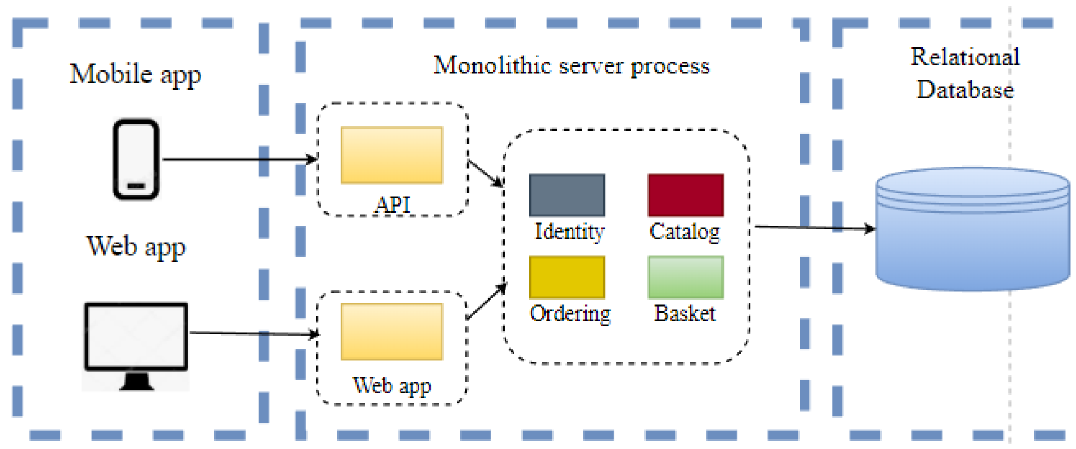
3 Περιγραφή των cloud-native εφαρμογών

3.1 Η ανεπάρκεια των παραδοσιακών μονολιθικών εφαρμογών

Για πολλά χρόνια, το κύριο μοντέλο δόμησης μιας εφαρμογής ήταν αυτό που σήμερα ονομάζουμε "μονολιθικό".**[29]** Μια μονολιθική εφαρμογή δομείται σαν μια ενιαία οντότητα, αποτελούμενη από τα εξής:

- Βάση δεδομένων. Για την αποθήκευση πληροφορίας, συνήθως με τη χρήση κάποιου συστήματος διαχείρισης σχεσιακών βάσεων δεδομένων (π.χ. MySQL, MariaDB).
- Διεπαφή χρήστη "στην πλευρά του πελάτη" (client-side). Συνήθως με τη μορφή HTML διαδικτυακών σελίδων.
- Εφαρμογή "στην πλευρά του εξυπηρετητή" (server-side). Ο εξυπηρετητής αναλαμβάνει τις διαδικαστικές λειτουργίες, όπως η διαχείριση HTTP αιτημάτων και η ενημέρωση της βάσης δεδομένων.

Παρακάτω θα εξετάσουμε το παράδειγμα μιας εταιρικής εφαρμογής και θα αναδείξουμε τη διαφορά μεταξύ της μονολιθικής και της cloud-native προσέγγισης.



Εικόνα 3-1 Μονολιθική Εφαρμογή

Πρόκειται για μια τυπική εφαρμογή για την πώληση προϊόντων και υπηρεσιών. Όπως παρατηρούμε στην εικόνα, όλη η λογική της εφαρμογής συγκεντρώνεται ενιαία, συμπεριλαμβάνοντας ενότητες όπως "Ταυτότητα", "Κατάλογος", "Παραγγελίες" κλπ., που επικοινωνούν απευθείας μεταξύ τους μέσω μίας μοναδικής διαδικασίας του server. Όλες οι ενότητες μοιράζονται την ίδια βάση δεδομένων, και η εφαρμογή αποκτά χρηστική αξία μέσω μιας HTML σελίδας ή ενός API σε μια εφαρμογή κινητού.

Καθώς η εφαρμογή αποκτά απήχηση και γιγαντώνεται, η μονολιθική κατασκευή παρουσιάζει διάφορα δισεπίλυτα προβλήματα όπως :

- Κάθε μικρή αλλαγή μπορεί να επιφέρει ανεπιθύμητα αποτελέσματα για όλη την εφαρμογή .

• Κάθε αλλαγή απαιτεί την επανέκδοση όλης της εφαρμογής .

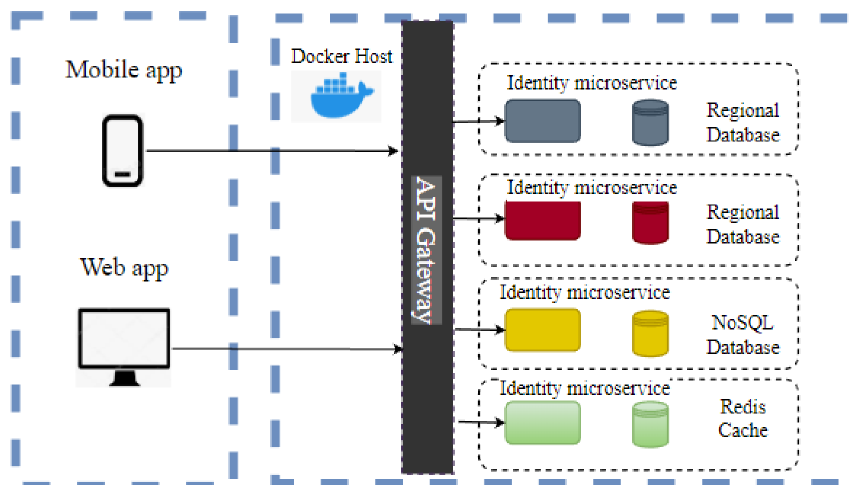
• Ένα δυσλειτουργικό στοιχείο μπορεί να επιφέρει την κατάρρευση όλης της εφαρμογής

• Η εισαγωγή νέων τεχνολογιών είναι δύσκολη και πολύπλοκη διαδικασία.

• Η εφαρμογή έχει κατασκευαστεί σε ιδιόκτητες υποδομές (on premises), και άρα στερείται των πλεονεκτημάτων του cloud που εξετάσαμε στο προηγούμενο κεφάλαιο.

Για την αντιμετώπιση των παραπάνω, και πολλών άλλων, προβλημάτων που προκύπτουν από τη δυσκαμψία του μονολιθικού μοντέλου, πολλοί οργανισμοί καταφεύγουν στην cloud-native προσέγγιση που θα αναλύσουμε παρακάτω.

3.2 Cloud-native εφαρμογές



Εικόνα 3-2 Cloud Native Εφαρμογή

Η ίδια εφαρμογή τώρα χρησιμοποιεί την cloud-native προσέγγιση. Παρατηρούμε πως οι ενότητες αποδομούνται σε μικρές απομονωμένες διαδικασίες ονόματι microservices, ενώ η αποθήκευση χρησιμοποιεί διάφορα είδη μνημών και βάσεων δεδομένων. Αλλά το σημαντικότερο, η εφαρμογή πλέον εκμεταλλεύεται την ελαστικότητα, την επεκτασιμότητα και την υψηλή διαθεσιμότητα μιας σύγχρονης πλατφόρμας νέφους. Για παράδειγμα, σε περίπτωση παροξυσμού της ζήτησης, καθίσταται δυνατό (με τις τεχνολογίες που θα δούμε παρακάτω) να αυξηθούν οι πόροι μόνο στα microservices που επηρεάζονται από την αυξημένη κίνηση (π.χ. Ordering, Basket), αντί μιας ολικής κλιμάκωσης. Ακόμη, μια αλλαγή στον κατάλογο δεν θα επιφέρει την επανέκδοση ολόκληρης της εφαρμογής αλλά μόνο του συγκεκριμένου Catalog microservice. Συνεπώς, η cloud-native δόμηση δύναται να προσφέρει στην εφαρμογή σημαντικά μειωμένο ενεργειακό, χρονικό και φυσικά χρηματικό κόστος, καθώς και μια πολύ πιο αξιόπιστη και ευκολότερα διαχειρίσιμη εφαρμογή.

3.2.1 Ορισμός του cloud-native

Η αρχιτεκτονική και οι υιοθετούμενες τεχνολογίες του cloud-native είναι μια νέα προσέγγιση για το σχεδιασμό , την κατασκευή και τη διαχείριση εφαρμογών που λαμβάνουν χώρα σε περιβάλλοντα νέφους και εκμεταλλεύονται πλήρως τα πλεονεκτήματα αυτού του υπολογιστικού μοντέλου.[30]

Σύμφωνα με το ίδρυμα cloud-native computing (Cloud Native Computing Foundation)[26], με το cloud-native αναπτύσσονται ευέλικτες και επεκτάσιμες εφαρμογές σε μοντέρνα δυναμικά περιβάλλοντα όπως το δημόσιο, το ιδιωτικό και το υβριδικό νέφος, χρησιμοποιώντας λογισμικό "ανοιχτού κώδικα" που δίνουν στην εφαρμογή τα κάτωθι χαρακτηριστικά:

- Τοποθέτηση σε containers. Κάθε μέρος της εφαρμογής (κώδικας, διαδικασίες, βιβλιοθήκες κλπ.) προσαρτάται σε δικό της container, εξασφαλίζοντας αναπαραγωγικότητα, διαφάνεια και την απομόνωση των χρησιμοποιούμενων πόρων.

- Δυναμική ενορχήστρωση. Τα containers δημιουργούνται, καταστρέφονται , προγραμματίζονται και διαχειρίζονται δυναμικά από έναν ενορχηστρωτή.

- Χρήση μικρό-υπηρεσιών (microservices). Μέσω αυτών αυξάνεται σημαντικά η ευελιξία και η συντηρησιμότητα της εφαρμογής.

3.2.2 Βασικές χρησιμοποιούμενες τεχνολογίες

3.2.2.1 Cloud υποδομές

Οι cloud-native εφαρμογές σχεδιάζονται ώστε να ευδοκιμήσουν σε περιβάλλοντα νέφους, αξιοποιώντας στο έπακρο τις παρεχόμενες υπηρεσίες. Μεταχειρίζονται την υποκείμενη υποδομή σαν αναλώσιμη, ικανή να δημιουργηθεί σε διάστημα λεπτών και να επεκταθεί ή να καταστραφεί κατά βούληση.

Σε ένα παραδοσιακό κέντρο δεδομένων, οι servers είναι φυσικά μηχανήματα, και άρα η επεκτασιμότητα εξασφαλίζεται με την πρόσθεση πόρων στα υπάρχοντα μηχανήματα. Αν κάποιο αποκτήσει βλάβη, το μηχάνημα "περιθάλπεται" και η βλάβη αποκαθίσταται, ωστόσο η έλλειψη διαθεσιμότητας γίνεται άμεσα αισθητή.

Εν αντιθέσει, στο cloud όλα παρέχονται από εικονικές μηχανές και containers. Η επέκταση γίνεται με τη δημιουργία περισσότερων, ενώ η περίπτωση δυσλειτουργίας αντιμετωπίζεται άμεσα με τη δημιουργία νέων, "υγιών" συστημάτων, αυτόματα.

3.2.2.2 Microservices

Η cloud-native αρχιτεκτονική χρησιμοποιεί μια νέα δημοφιλή μέθοδο για την κατασκευή σύγχρονων εφαρμογών, αυτή των microservices.[32]Σχεδιασμένα σαν ένα κατανεμημένο σύνολο από μικρές, αυτόνομες υπηρεσίες που αλληλοεπιδρούν με χρήση πρωτοκόλλων του διαδικτύου, τα microservices φέρουν τα παρακάτω χαρακτηριστικά :

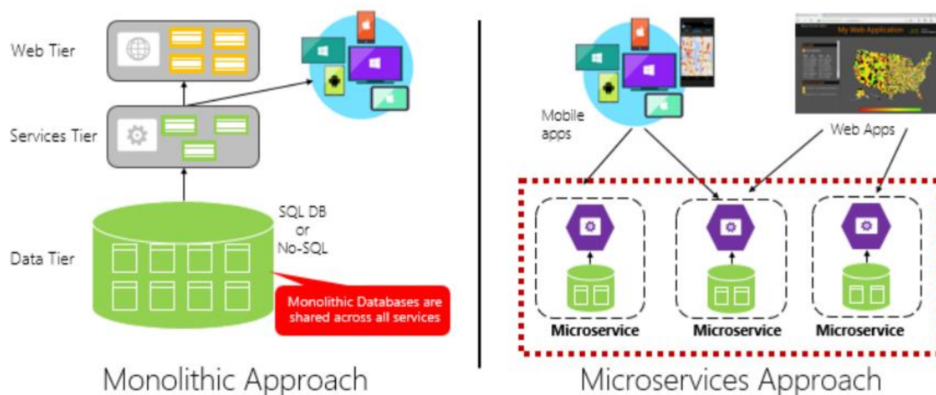
• Το καθένα αναπτύσσεται αυτόνομα και μπορεί να "ξεκινήσει" ανεξάρτητα από τα υπόλοιπα.

• Το καθένα επιτελεί συγκεκριμένη λειτουργία επί του συνολικού έργου .

• Το καθένα προσαρτάται σε δικό του container, φέροντας ξεχωριστή τεχνολογία αποθήκευσης, όλες τις απαραίτητες εξαρτήσεις και προγραμματιστική πλατφόρμα.

• Το καθένα "τρέχει" τη δική του διαδικασία και επικοινωνεί με τα υπόλοιπα με χρήση καθιερωμένων πρωτοκόλλων όπως το HTTP/HTTPS, gRPC, WebSockets κλπ.

• Το καθένα πρέπει να έχει πρόσβαση σε συγκεκριμένες κλάσεις των υπολοίπων, μειώνοντας την μεταξύ τους διάδραση στο ελάχιστο αναγκαίο χωρίς να στερεί την απαραίτητη ανταλλαγή πληροφοριών. .



Εικόνα 3-3 Εφαρμογή βασισμένη σε Microservices [30]

Στην εικόνα παρατηρείται μια σύγκριση της μονολιθικής προσέγγισης με αυτή των microservices. Στην πρώτη διακρίνεται μια διαστρωματωμένη αρχιτεκτονική, η οποία χρησιμοποιείται για την εκτέλεση μιας μοναδικής διαδικασίας, συνήθως καταναλώνοντας μια σχεσιακή βάση δεδομένων (relational data base). Στη δεύτερη περίπτωση, η εφαρμογή κατακερματίζεται σε αυτόνομες υπηρεσίες, κάθε μια με τη δική της λογική, κατάσταση (state), δεδομένα και αποθηκευτικό μέσο.

Η αρχιτεκτονική των microservices παρέχει σημαντικά πλεονεκτήματα έναντι της μονολιθικής:

• Κάθε microservice έχει αυτόνομο "κύκλο ζωής" και μπορεί να εξελιχθεί ανεξάρτητα. Κατά αυτόν τον τρόπο, μπορεί να ενημερωθεί ένα μικρό μέρος της εφαρμογής χωρίς τον κίνδυνο της αποσταθεροποίησης του συστήματος, ενώ η έκδοση της ενημέρωσης μπορεί να συμβεί "ζωντανά" και απρόσκοπτα.

• Κάθε microservice μπορεί να επεκταθεί ανεξάρτητα. Αντί να αυξηθούν οι πόροι σε όλη την εφαρμογή σαν μια ενιαία οντότητα, αυξάνονται μόνο στα microservices που χρειάζονται επιπλέον ισχύ για την περάτωση των εργασιών τους και τη συμφωνία με τις προδιαγραφές απόδοσης της εφαρμογής, μειώνοντας έτσι σημαντικά τα συνολικά κόστη επεκτασιμότητας.

3.2.2.3 Τεμαχισμός Δικτύου (Network slicing)

Η τεχνολογία αυτή δεν είναι εξ ορισμού συνυφασμένη με το cloud-native, αλλά θεωρείται απαραίτητη για την υιοθέτηση του cloud-native παραδείγματος σε 5G περιβάλλοντα.[33] Ο τεμαχισμός δικτύου είναι μια μέθοδος για τη δημιουργία πολλαπλών εικονικών δικτύων πάνω από μια κοινή υποδομή, με κάθε εικονικό υπό-δίκτυο (τεμάχιο) να είναι λογικά διαχωρισμένο από τα υπόλοιπα, και άρα να μπορεί να παραμετροποιηθεί ξεχωριστά και να λάβει συγκεκριμένους πόρους. Κατά αυτόν τον τρόπο, κάθε τεμάχιο μπορεί να βελτιστοποιηθεί για την υποστήριξη συγκεκριμένων εφαρμογών (ή τμημάτων εφαρμογών-microservices) , υπηρεσιών, ομάδων χρηστών κλπ., δηλαδή να παραλάβει συγκεκριμένους δικτυακούς πόρους όπως εύρος ζώνης και δικτυακές λειτουργίες (network functions). Οι κύριες τεχνολογίες που επιτρέπουν τη δημιουργία ευέλικτων και επεκτάσιμων δικτυακών τεμαχίων είναι η δικτύωση καθοριζόμενη από λογισμικό (Software Defined Networking - SDN) και η εικονικοποίηση δικτυακών λειτουργιών (Network Functions Virtualization - NFV). Θα μπορούσαμε να πούμε πως η τοποθέτηση σε containers εξασφαλίζει μια εξατομικευμένη προσέγγιση σε κάθε microservice για την παροχή υπολογιστικών πόρων, ενώ ο τεμαχισμός αφορά, με αντίστοιχη λογική, τη δικτύωση. Καθώς η παρούσα εργασία θα εστιάσει στη δέσμευση υπολογιστικών πόρων για τα containers, οι δικτυακές λειτουργίες δε θα αναλυθούν περισσότερο αν και πρόκειται για ένα εξίσου σημαντικό κομμάτι. Ωστόσο, οι αλγοριθμικές τεχνικές που παρουσιάζουμε πιθανόν να μπορούν να αξιοποιηθούν , να τροποποιηθούν ή να επεκταθούν για τη δημιουργία μιας συνολικής "λύσης" στο πρόβλημα της δέσμευσης πόρων.

3.2.2.4 Containers και Ενορχηστρωτές

Τα containers παρέχουν στην εφαρμογή φορητότητα και εγγυημένη διαθεσιμότητα. Όντας ελαφρύτερα (μικρότερα σε μέγεθος) από τις εικονικές μηχανές, ενδείκνυνται για την ενθυλάκωση microservices αφού πληθαίνει ο αριθμός των παράλληλων συστημάτων που μπορούν να τρέχουν στον ίδιο οικοδεσπότη. Υπενθυμίζοντας, ο κώδικας του microservice, μαζί με τις απαραίτητες εξαρτήσεις προσαρτώνται σε ένα δυαδικό αρχείο ονόματι "εικόνα container". Κατά την έναρξη, η εικόνα μετατρέπεται σε "στιγμιότυπο container" , το οποίο μπορεί να τρέξει ανεξαρτήτως του περιβάλλοντος και του λειτουργικού συστήματος στο οποίο βρίσκεται, ενώ μπορούν να υπάρξουν οσαδήποτε στιγμιότυπα της ίδιας εικόνας.

Ο κατακερματισμός της εφαρμογής σε microservices και η ενθυλάκωση αυτών σε containers μπορεί να οδηγήσει σε εκατοντάδες ή χιλιάδες containers που λειτουργούν παράλληλα, δημιουργώντας άρα την ανάγκη για την ύπαρξη ενός προγράμματος για την αυτοματοποίηση του συντονισμού και της διαχείρισής τους. Τέτοια λογισμικά ονομάζονται "ενορχηστρωτές container" και είναι υπεύθυνα για την παρατήρηση της ορθής λειτουργίας των container, την επανεκκίνηση σε περίπτωση μη ορθής εκτέλεσης και το σημαντικότερο : τη δέσμευση και την κλιμάκωση των υπολογιστικών πόρων για την ανταπόκριση στις ανάγκες του εισερχόμενου φόρτου εργασίας (workload) από εφαρμογές και υπηρεσίες, προσφέροντας, γενικά, τα κάτωθι πλεονεκτήματα [34]:

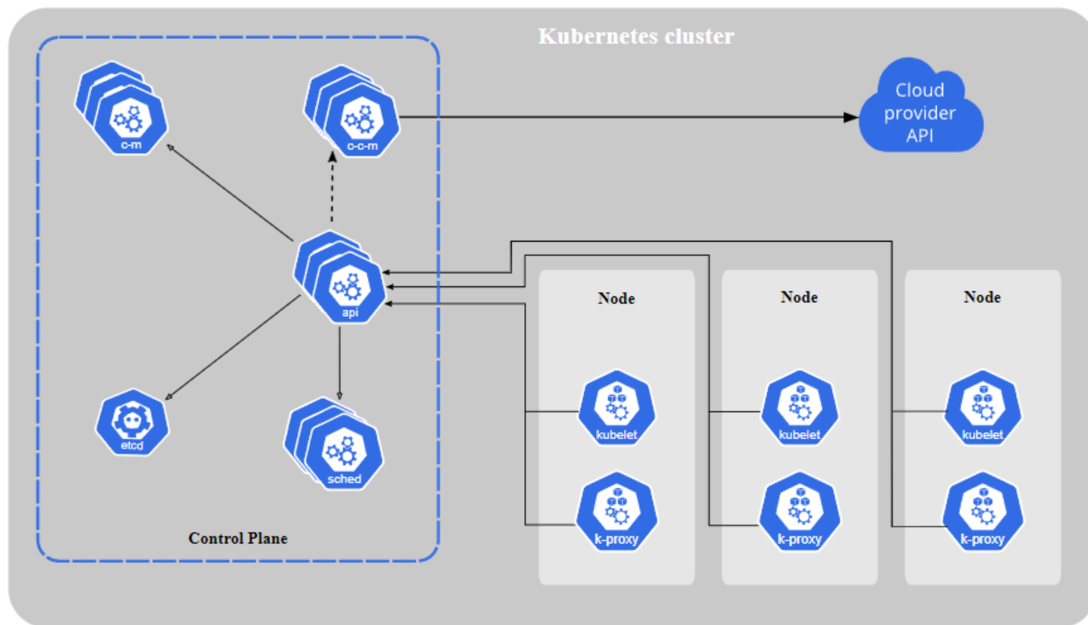
- Αυτοματοποιημένη διαχείριση. Πρόκειται για το βασικότερο πλεονέκτημα και τον κυριότερο παράγοντα για την υιοθέτηση των ενορχητρωτών. Η πολυπλοκότητα που εισάγεται από την συνύπαρξη τεράστιου πλήθους container για τη λειτουργία μιας εφαρμογής είναι τέτοια που καθιστά αδύνατη τη διαχείριση της υποδομής χωρίς κάποια κεντρική, αυτοματοποιημένη προσέγγιση.
- Ανθεκτικότητα. Τα εργαλεία που παρέχουν οι ενορχηστρωτές επιτρέπουν την αυτόματη επανεκκίνηση ή την κλιμάκωση των πόρων ενός container ή μιας συστάδας (cluster) container, παρέχοντας αυξημένη αντοχή σε βλάβες και άμεση ανταπόκριση στην περίπτωση παροξυσμού (spike) της ζήτησης σε πόρους.
- Ασφάλεια. Ο μεγάλος βαθμός αυτοματοποίησης μειώνει τον κίνδυνο από ενδεχόμενα "ανθρώπινα λάθη".

Ο πιο διαδεδομένος ενορχηστρωτής σήμερα είναι ο Κυβερνητής (Kubernetes - K8S) [35], [36], ένα ανοιχτού κώδικα λογισμικό αρχικά σχεδιασμένο από την Google, το οποίο διαχειρίζεται πλέον το Ίδρυμα Cloud-Native (Cloud-Native Computing Foundation). Όταν εκτελείται ο Κυβερνητής, δημιουργείται μια συστάδα (cluster) αποτελούμενη από ένα σύνολο κόμβων-εργατών (worker nodes) , κάθε ένας εκ των οποίων αποτελείται από τα κάτωθι "συστατικά":

- Kubelet. Ένας πράκτορας (agent) υπεύθυνος για την ορθή λειτουργία (σύμφωνα με τις προδιαγραφές) των container που βρίσκονται εντός ενός Pod, που είναι ένα σύνολο από ένα ή περισσότερα containers εντός του ίδιου κόμβου και αποτελεί την βασική διαχειρίσιμη μονάδα του Κυβερνητή.
- Kube-proxy. Είναι ένας δικτυακός μεσολαβητής (network proxy) υπεύθυνος για τη δικτύωση (π.χ. τη δρομολόγηση της κίνησης) των container.
- Μηχανή Container (Container Runtime). Το απαραίτητο λογισμικό για τη δημιουργία container (π.χ. Docker, Containerd)

Ωστόσο μεγαλύτερο ενδιαφέρον για την παρούσα εργασία παρουσιάζει ο κόμβος ελέγχου της συστάδας, ονόματι Master κόμβος, που φέρει τα εξής συστατικά:

- etcd. Πρόκειται για μια κατανεμημένη μέθοδο αποθήκευσης κλειδιού-στοιχείου (key-value), που αποθηκεύει την παραμετροποίηση της συστάδας και αντικατοπτρίζει την συνολική κατάστασή της σε κάθε χρονική στιγμή.
- kube-controller-manager. Χρησιμοποιείται για την εκτέλεση διαδικασιών ελέγχου. Ο Κυβερνητής λειτουργεί με τη λογική της "ποθητής κατάστασης" (desired state) που παραμετροποιείται από το χρήστη μέσω του API (Application Programming Interface – Διεπαφή Προγραμματισμού Εφαρμογών). Οι controllers επιτηρούν τη συστάδα για την επίτευξη και τη διατήρηση της ποθητής κατάστασης, εκτελώντας διαδικασίες όπως η δημιουργία και η καταστροφή Pods.
- kube-scheduler. Είναι το συστατικό στο οποίο εστιάζει περισσότερο η εργασία μας, και είναι υπεύθυνο για την επιλογή του κατάλληλου κόμβου για την τοποθέτηση νεοσύστατων Pods σύμφωνα με την διαθεσιμότητα των πόρων. Για το σκοπό αυτό, ο scheduler πρέπει να έχει γνώση της κατάστασης των πόρων στους κόμβους και άλλων παραμέτρων εισαγόμενων από τις εφαρμογές όπως η ζητούμενη ποιότητα υπηρεσίας (QoS).



Εικόνα 3-4 Τα συστατικά του Kubernetes [36]

Ωστόσο, ο Κυβερνητής και άλλοι παρόμοιοι διαχειριστές εστιάζουν σε μεγάλα κέντρα δεδομένων και δεν είναι σχεδιασμένοι για μια γεωγραφικά διεσπαρμένη υποδομή. Επιπλέον, συχνά αποτυγχάνουν στην υιοθέτηση μιας πολιτικής που να εγγυάται τη χαμηλή καθυστέρηση που αιτούνται συγκεκριμένες εφαρμογές. Συνεπώς, δημιουργείται η ανάγκη για τη δημιουργία ενός καινοτόμου διαχειριστή πόρων (ή μιας ιεραρχίας διαχειριστών) ικανού να ελέγξει μια σύγχρονη κατανεμημένη υποκείμενη υποδομή και να αναθέσει κατάλληλα τους υπολογιστικούς πόρους. Η αναζήτηση τέτοιων μηχανισμών είναι το θέμα της παρούσας εργασίας καθώς πρόκειται για ένα ανοιχτό ερευνητικό αντικείμενο που βρίσκεται ακόμη σε αρκετά πρώιμα στάδια.

3.3 Cloud-native εφαρμογές σε μια ιεραρχική edge-fog-cloud τοπολογία

Καθώς η τεχνολογία του 5G ωριμάζει και εδραιώνεται, δημιουργείται πρόσφορο έδαφος για την ανάπτυξη περισσότερων IoT εφαρμογών, καθώς και καινοτόμων εφαρμογών για τους χρήστες. Για την υποστήριξή τους δεν επαρκεί η κεντρική υποδομή του cloud, γι' αυτό θα επιστρατευτούν οι προαναφερθείσες τεχνολογίες των edge και fog computing, που θα προσφέρουν το ποθητό επίπεδο ποιότητας υπηρεσιών. Αξίζει να τονιστεί ο ρόλος των containers, αφού η πολύ "ελαφρύτερη" φύση τους σε σχέση με τις εικονικές μηχανές επιτρέπει τη δημιουργία και την εκτέλεσή τους ακόμη και στους περιορισμένων δυνατοτήτων κόμβων του fog αλλά και στις edge συσκευές. Κατά αυτόν τον τρόπο είναι δυνατή η δημιουργία κατανεμημένων υποδομών, που εκτείνονται από τους απομακρυσμένους cloud εξυπηρετητές μέχρι τις κοντινές edge συσκευές.

Ωστόσο, η μεγαλύτερη πρόκληση που προκύπτει είναι η εύρεση του κατάλληλου μηχανισμού για τη δέσμευση πόρων και την ανάθεση εργασιών, τόσο μεταξύ των στοιχείων του ιδίου επιπέδου (π.χ. στο στρώμα του fog), όσο και μεταξύ των διαφορετικών επιπέδων στην ιεραρχία. Για παράδειγμα, έχοντας ένα microservice

με συγκεκριμένες ανάγκες σε επεξεργαστική ισχύ και αποθηκευτικό χώρο αλλά και απαίτηση της εφαρμογής για μια μέγιστη αποδεκτή καθυστέρηση, πρέπει να αναζητηθεί το καταλληλότερο επίπεδο (ή στρώμα) της υποδομής , και εν συνεχεία τα επιμέρους στοιχεία του επιπέδου που θα αναλάβουν την εκτέλεση του container. Στην περίπτωση που το edge στρώμα δεν είναι ικανό για την περάτωση λόγω έλλειψης των απαιτούμενων πόρων, το microservice θα πρέπει προωθηθεί στο ανώτερο στρώμα του fog και εν συνεχεία στο cloud, πάντοτε σε προσπάθεια για την τήρηση των προδιαγραφών.

Στα πλαίσια της εργασίας δε θα αναλυθούν οι αρχιτεκτονικές και οι σχεδιαστικές παράμετροι που επιτρέπουν το συντονισμό και την επικοινωνία των επιμέρους στοιχείων, οπότε ο μηχανισμός αυτός μπορεί να λάβει τη μορφή ενός απλού αλγορίθμου και να ενσωματωθεί στα συστήματα διαχείρισης και παρακολούθησης πόρων των ενορχηστρωτών, όπως ο Κυβερνητής αλλά και άλλοι ενορχηστρωτές προορισμένοι για κατανεμημένα περιβάλλοντα (π.χ. KubeEdge).

Στη συνέχεια θα εξετάσουμε παρόμοιους μηχανισμούς δέσμευσης πόρων στη βιβλιογραφία, κατηγοριοποιώντας τους σύμφωνα με τις μεθόδους που χρησιμοποιούν και τους παράγοντες προς βελτιστοποίηση που εστιάζουν.

4 Σχετική Εργασία

Η ανεύρεση μηχανισμών για τη δέσμευση πόρων είναι ένα πολυδιάστατο ανοικτό ζήτημα που προσελκύει το ενδιαφέρον πληθώρας ερευνητικών ομάδων ανά τον κόσμο. Η μοντελοποίηση του προβλήματος ποικίλει σύμφωνα με την τοπολογία που εξετάζεται και τις τεχνολογίες που υιοθετούνται, ενώ οι προτεινόμενες λύσεις επιστρατεύουν τεχνικές από την ευρύτερη σφαίρα των μαθηματικών και της επιστήμης των υπολογιστών.

Παρόλο που ειδικότερα η ανάπτυξη μηχανισμών για τη δέσμευση πόρων για την υποστήριξη cloud-native εφαρμογών σε διαστρωματωμένο edge-fog-cloud περιβάλλον δεν έχει εξεταστεί ενδελεχώς στη βιβλιογραφία, είναι σκόπιμη η μελέτη του ευρύτερου προβλήματος της δέσμευσης πόρων σε περιβάλλοντα νέφους καθώς συχνά το υπό μελέτη πρόβλημα και οι υιοθετούμενες τεχνικές παρουσιάζουν μεγάλη συνάφεια και μπορούν να αποτελέσουν πολύτιμη πηγή έμπνευσης και γνώσης.

Οι Xin Li et al. [37] εξετάζουν την τοποθέτηση εικονικών μηχανών στα φυσικά συστήματα ενός εξυπηρετητή νέφους για την πραγματοποίηση Big-Data ανάλυσης από τα δεδομένα IoT συσκευών, με σκοπό την βέλτιστη χρησιμοποίηση των πόρων δικτύωσης και την αποφυγή συμφορήσεων. Η μοντελοποίηση γίνεται με χρήση γράφου, όπου οι κόμβοι αντιπροσωπεύουν τις εικονικές μηχανές και οι σύνδεσμοι τη μεταξύ τους δικτυακή κίνηση , ενώ σκοπός είναι η ελαχιστοποίηση της μέγιστης χρησιμοποίησης των συνδέσμων. Παρουσιάζεται ένας ευρετικός αλγόριθμος η βασική ιδέα του οποίου είναι η τοποθέτηση όσο το δυνατόν περισσότερων εικονικών μηχανών που αλληλοεπιδρούν στο ίδιο φυσικό μηχάνημα προς μείωση του (δικτυακού) κόστους επικοινωνίας.

Στο [38], οι συγγραφείς μελετούν τη δημιουργία ενός μηχανισμού για τη δέσμευση πόρων για την εκτέλεση network slices, τα οποία υποδιαιρούνται στα μικρότερα μ-slices. Η υποδομή οπτικοποιείται με χρήση γράφου, όπου οι κόμβοι είναι μηχανήματα με επεξεργαστική ισχύ (flops/s) και οι σύνδεσμοι αντιπροσωπεύουν τη χωρητικότητα δικτύου (bits/s). Ο αλγόριθμος που αναπτύσσεται χρησιμοποιεί βάρη ώστε να επιτρέψει την ανταλλαγή μεταξύ "δικαιοσύνης" στη δικτύωση ή στην επεξεργασία (traffic/computing fairness) ανάλογα με την ισχύ των υποδομών και τον εκάστοτε σκοπό.

Οι Nahida Kiran et al. [39]ασχολούνται με τη δέσμευση πόρων για την εκτέλεση εικονικοποιημένων δικτυακών λειτουργιών (NFV's) σε αποκεντρωμένους κόμβους MEC, που χειρίζονται από έναν κεντρικό SDN χειριστή. Εάν το ζητούμενο VNF και οι απαραίτητοι πόροι είναι διαθέσιμοι στον MEC κόμβο, η εκτέλεση συμβαίνει εκεί αλλιώς εξετάζονται τεχνικές όπως η κλωνοποίηση του VNF από γειτονικούς κόμβους ή από το απομακρυσμένο cloud με τη βοήθεια του SDN χειριστή. Γίνεται και πάλι χρήση γράφου όπου οι κόμβοι είναι οι κόμβοι του MEC ενώ σύνδεσμοι οι μεταξύ τους συνδέσεις, ενώ στη συνέχεια παρουσιάζεται ένας γενετικός ευρετικός αλγόριθμος.

Εστιάζοντας τώρα στις πιο κοντινές από άποψη τοπολογίας και τεχνολογιών μελέτες, οι συγγραφείς στο [40]παρουσιάζουν το foggy, μια αρχιτεκτονική δομή

βασισμένη σε ανοικτού κώδικα εργαλεία που αναλαμβάνει την ανάθεση εργασιών και την αντίστοιχη δέσμευση πόρων σε ένα διαστρωματωμένο, ετερογενές fog-cloud περιβάλλον. Η αρχιτεκτονική βασίζεται σε ένα σύστημα διαπραγμάτευσης (negotiator) , σε έναν ενορχηστρωτή (orchestrator) και σε ένα σύστημα παρακολούθησης εργασιών και πόρων (inventory). Γίνεται προσπάθεια αντιστοίχισης των microservices που καταφθάνουν σαν ουρά FIFO στους κόμβους οι οποίοι κατατάσσονται τόσο σύμφωνα με την επεξεργαστική ισχύ τους, όσο και σύμφωνα με τη δικτύωση με τον τελικό προορισμό.

Στο [41]υλοποιείται ένα σύστημα τοποθέτησης microservices σε μια πολύ-επίπεδη fog τοπολογία. Όλες οι λειτουργίες εκτελούνται αποκεντρωμένα από τους ίδιους τους κόμβους (load balancing, service discovery), οι οποίοι επικοινωνούν τόσο με τους κόμβου του ιδίου επιπέδου (cluster) όσο και με τα ανώτερα επίπεδα. Γίνεται και πάλι χρήση γράφου, ενώ η βασική ιδέα του αλγορίθμου είναι πως αν το ζητούμενο microservice υπάρχει ήδη στον κόμβο , γίνεται επέκταση με προσθήκη πόρων , ειδάλλως γίνεται προσπάθεια για την αντιγραφή του εντός του cluster ή το αίτημα μεταφέρεται σε ανώτερο επίπεδο.

Οι Joseph & Chandrasekaran **[42]**παρατηρούν ότι η αλληλεπίδραση μεταξύ των microservices που συνθέτουν μια εφαρμογή είναι καθοριστικός παράγοντας της αύξησης του χρόνου απόκρισης και συνεπώς επιδιώκουν την τοποθέτηση όσο το δυνατόν περισσότερων αλληλοεπιδρώντων microservices στον ίδιο κόμβο. Το πρόβλημα μονετελοποιείται αρχικά ως δυαδικό τετραγωνικού προγραμματισμού (BQPP) και στη συνέχεια παρουσιάζεται ο ευρετικός αλγόριθμος intMA που βασίζεται στη χρήση διπλής κατεύθυνσης γράφου με βάρη. .

Οι Silva & Fonseca [43]δίνουν μια πρωτότυπη προσέγγιση στο ζήτημα της δέσμευσης πόρων σε ένα fog-cloud περιβάλλον αναπτύσσοντας έναν αλγόριθμο για την πρόβλεψη της ζήτησης προς αποφυγή των απορρίψεων αιτημάτων, ειδικά αυτών που είναι ευαίσθητα στο χρόνο. Ο αλγόριθμος βασίζεται στην οπισθοδρόμηση γκαουσιανής διαδικασίας (GPR) και αναλύει τα παρελθοντικά αιτήματα ώστε να εξασφαλίζει επάρκεια στο επίπεδο fog για την υποστήριξη μελλοντικών χρόνο-κρίσιμων αιτημάτων.

Ερευνητική εργασία	Προσέγγιση λύσης	Χρήση γράφου	Edge-fog-cloud τοπολογία	Χρήση microservices
[37]	ΑΓΠ/ευρετικός αλγόριθμος	Ναι	Όχι	Όχι
[38]	Αλγόριθμος	Ναι	Όχι	Όχι
[39]	ΜΓΠ/Γενετικός ευρετικός αλγόριθμος	Ναι	Ναι	Όχι
[40]	Πρόγραμμα Λογισμικού	Όχι	Ναι	Ναι
[41]	Ευρετικός αλγόριθμος	Ναι	Ναι	Ναι
[42]	ΔΤΠ/ευρετικός αλγόριθμος	Ναι	Ναι	Ναι
[43]	αλγόριθμος	Όχι	Ναι	Όχι

Πίνακας 4-1 Σύνοψη της σχετικής εργασίας

ΑΓΠ: Ακέραιος Γραμμικός Προγραμματισμός
ΜΓΠ: Μικτός Γραμμικός Προγραμματισμός
ΔΤΠ: Δυαδικός Τετραγωνικός Προγραμματισμός

Τέλος θα εξετάσουμε μια εργασία που χρησιμοποιεί έναν αλγόριθμο πλειστηριασμού για τη δέσμευση πόρων, καθώς η χρήση τέτοιων αλγορίθμων παρουσιάζει ιδιαίτερο ενδιαφέρον και συμπεριλαμβάνεται στους στόχους μας για μελλοντική εργασία. Οι συγγραφείς στο [44] εξετάζουν τη δέσμευση πόρων για ένα δίκτυο MEC. Θεωρείται σκόπιμη η ανάπτυξη ενός on-line μηχανισμού που λαμβάνει υπόψη τα ετερογενή από άποψης χρόνου αιτήματα. Συγκεκριμένα, ο χρόνος κατακερματίζεται σε χρονικές περιόδους T σε κάθε μια από τις οποίες εκτελείται μια ξεχωριστή διαδικασία δέσμευσης πόρων. Η διαδικασία αυτή λαμβάνει τη μορφή συνδυαστικού πλειστηριασμού, όπου ο σταθμός βάσης (BS) είναι ο υπέυθυνος για τον πλειστηριασμό , οι πόροι υπολογισμού και μνήμης τα αντικείμενα και οι χρήστες οι ενδιαφερόμενοι αγοραστές. Οι "νικητές" κάθε γύρου πληρώνουν το αντίτιμο στο τέλος του γύρου και καταλαμβάνουν τους αντίστοιχους πόρους για το χρονικό διάστημα που ζητούν, ενώ οι υπόλοιποι μπορούν να συμμετάσχουν στον πλειστηριασμό της επόμενης χρονικής περιόδου μαζί με νέο-αφιχθέντες ενδιαφερόμενους. Τα αιτήματα κατηγοριοποιούνται σύμφωνα με το προφίλ τους (ανάγκες σε χρόνο και καθυστέρηση), και η τιμολόγηση είναι διαφορετική για κάθε τέτοια "ομάδα" αιτημάτων. Η μοντελοποίηση του προβλήματος βελτιστοποίησης λαμβάνει τη μορφή ενός πολυδιάστατου προβλήματος σακιδίου (knapsack problem),

όπου τα αιτήματα λαμβάνουν τη μορφή κύβου με τις διαστάσεις να αντιπροσωπεύουν το κόστος επικοινωνίας, το υπολογιστικό κόστος και το χρόνο.

Όλες οι παραπάνω προσεγγίσεις της βιβλιογραφίας φέρουν περιορισμούς στην αρχιτεκτονική της υποδομής, στη φύση του φόρτου εργασίας (workload), στα συστήματα που χρησιμοποιούνται και στις υιοθετούμενες τεχνολογίες. Σκοπός της παρούσας εργασίας είναι να απαλλάξουμε τη λύση από όσο το δυνατόν περισσότερες τεχνικές εξειδικεύσεις και υποθέσεις, ώστε να δώσουμε μια "γενικού σκοπού" προσέγγιση, ικανή να ενσωματωθεί, με κατάλληλες τροποποιήσεις, στο εκάστοτε πρόβλημα και τις ανάγκες του. Ακόμη, η τεχνική του Rollout που θα χρησιμοποιήσουμε είναι μια πολύ χρήσιμη και πρωτότυπη μέθοδος τόσο για θέματα βελτιστοποίησης όσο και για την κατασκευή συστημάτων μηχανικής μάθησης. Τέλος, οι περισσότερες εργασίες εστιάζουν κυρίως στον πάροχο, μελετώντας την οικονομική και ενεργειακή ευημερία του, συχνά αμελώντας τις ανάγκες των "πελατών" της υποδομής, όπως τα microservices. Για το λόγο αυτό το εξετασθέν πρόβλημα θα εστιάσει ακριβώς στην τήρηση των προδιαγραφών που θέτουν οι εφαρμογές και στη συνολική χρονική και οικονομική βελτιστοποίηση της εξυπηρέτησης των microservices.

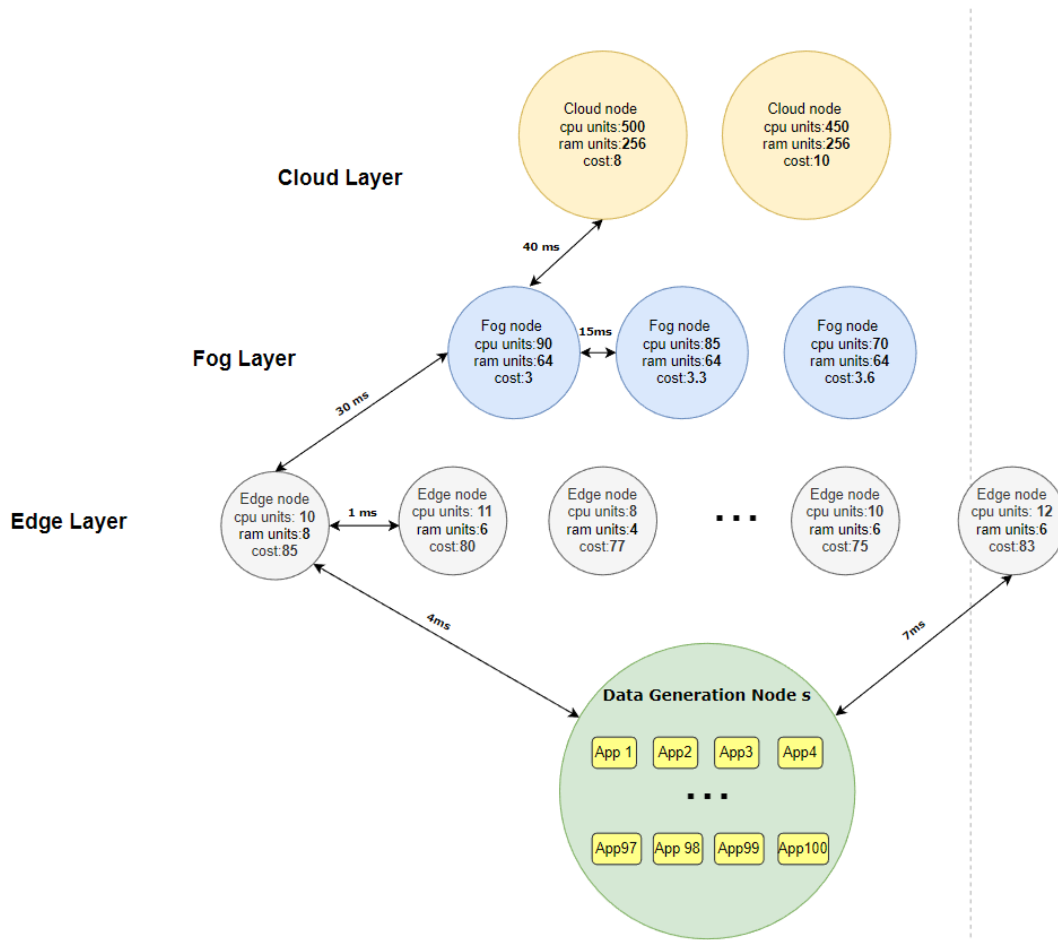
5 Μεικτή βεβαρυμμένη βελτιστοποίηση καθυστέρησης-κόστους

5.1 Περιγραφή των στοιχείων της υποδομής και του προβλήματος

Θεωρούμε μια ιεραρχική edge-fog-cloud υποδομή για την εξυπηρέτηση του εισερχόμενου φόρτου εργασίας (workload). Η υποδομή μπορεί να θεωρηθεί ως ένα διαστρωματωμένο σύνολο κόμβων, όπου σε κάθε τέτοιο κόμβο είναι τοποθετημένα ένα ή περισσότερα υπολογιστικά συστήματα κάποιας μορφής. Όπως είναι φυσικό, η δυναμική των συστημάτων αυτών ποικίλει και γενικά αυξάνεται καθώς "ανεβαίνουμε" στα στρώματα της τοπολογίας. Για παράδειγμα, στο edge στρώμα συναντούμε μηχανήματα περιορισμένων πόρων όπως laptops, raspberry pies, desktop GPUs, κινητά τηλέφωνα κλπ., στο fog ισχυρότερα όπως τα cloudlets και τέλος φτάνουμε στους μεγάλους και πρακτικά αστείρευτους – από άποψη πόρων - εξυπηρετητές του cloud. Επιπλέον, η καθυστέρηση (latency) $l(i,s)$ ενός κόμβου *i* ορίζεται ως η χρονική καθυστέρηση για τη μεταφορά πληροφορίας από και προς τον κόμβο *s*, στον οποίο θεωρούμε πως συγκεντρώνεται ο φόρτος εργασίας. Είναι ελάχιστη για το edge στρώμα που βρίσκεται εντός του δικτύου πρόσβασης το οποίο απαρτίζεται από κόμβους τοποθετημένους «κοντά» στην πηγή παραγωγής δεδομένων, και αυξάνεται σταδιακά κατά την άνοδο στην ιεραρχία της υποδομής, αφού μεγαλώνει η φυσική απόσταση των κόμβων. Σύμφωνα με το μοντέλο των εφαρμογών που θα χρησιμοποιήσουμε, κρίνεται σκόπιμο να ορίσουμε και την σχετική καθυστέρηση $l(i_1,i_2)=l(i_2,i_1)$ για κάθε ζεύγος κόμβων (i_1,i_2), που εκφράζει την καθυστέρηση για τη μεταφορά πληροφορίας από τον κόμβο i_1 στον κόμβο i_2 και αντιστρόφως. Η σχετική καθυστέρηση είναι μικρή για κόμβους που βρίσκονται εντός του ιδίου στρώματος και ειδικά για τους κόμβους του edge, που θεωρούμε πως είναι περισσότεροι στο πλήθος και έχουν μεγαλύτερη γεωγραφική πυκνότητα, ενώ μεγαλώνει για κόμβους διαφορετικών στρωμάτων. Τέλος, το κόστος επεξεργασίας $cs(i)$ ενός κόμβου *i*, είναι το χρηματικό κόστος που προκύπτει από τα έξοδα για την αγορά και τη λειτουργία των υπολογιστικών συστημάτων του κόμβου, και μετακυλίεται στους χρήστες της υποδομής σαν κόστος εξυπηρέτησης. Είναι ελάχιστο για το στρώμα του cloud, καθώς επιτυγχάνει οικονομίες κλίμακας στοχεύοντας στην μαζική εξυπηρέτηση από τα ισχυρά συστήματά του, και αυξάνεται σταδιακά μέχρι το edge στρώμα, για το οποίο θεωρούμε μεγαλύτερο κόστος εξαιτίας των περιορισμένων πόρων του (που πολλές φορές τοποθετούνται σε δυσπρόσιτα γεωγραφικά σημεία) και της ταχύτερης εξυπηρέτησης που προσφέρει.

Τελικά, η υποδομή μπορεί να λάβει τη μορφή ενός συνεκτικού μη-κατευθυνόμενου γράφου με βάρη $\boldsymbol{G^P}=(\boldsymbol{V^p},\boldsymbol{E^p},\boldsymbol{C^p},\boldsymbol{R^p},\boldsymbol{CS^p},\boldsymbol{L^p})$. Κάθε κόμβος $i\in V^p$ χαρακτηρίζεται από την επεξεργαστική χωρητικότητα C_i^P, τη χωρητικότητα μνήμης RAM R_i^P καθώς και το κόστος CS_i^P. Από την άλλη, κάθε ακμή μεταξύ 2 κόμβων i_1,i_2 φέρει ένα βάρος $L_{i_1,i_2}^P=L_{i_2,i_1}^P$ που εκφράζει τη σχετική χρονική καθυστέρηση για τη μεταφορά πληροφορίας από τον κόμβο i_1 στον κόμβο i_2 και αντιστρόφως. Ακόμη, ο γράφος περιλαμβάνει έναν ειδικό κόμβο παραγωγής δεδομένων $s\in V^p$, όπου εκεί θεωρούμε

πως συγκεντρώνεται ο φόρτος εργασίας (workload). Ο κόμβος αυτός περιέχει τα πιο χαμηλής δυναμικής συστήματα (για παράδειγμα, τις ίδιες τις IoT συσκευές). Θα θεωρήσουμε πως όποιες εργασίες μπορούν να γίνουν τοπικά στον κόμβο παραγωγής δεδομένων, διεκπεραιώνονται πριν τη διαδικασία εξυπηρέτησης, και άρα τελικά η υποδομή αναλαμβάνει τον εναπομείναντα φόρτο εργασίας. Επομένως, για τον κόμβο παραγωγής θα ισχύει $C_s^P = R_s^P = CS_s^P = 0$. Σύμφωνα με τα παραπάνω, η καθυστέρηση κάθε κόμβου $i \in V^p$ ορίζεται ως η καθυστέρηση από και προς τον κόμβο παραγωγής δεδομένων $L_{i,s}^P$, ενώ η σχετική καθυστέρηση μεταξύ δύο κόμβων i_1, i_2 είναι L_{i_1,i_2}^P.

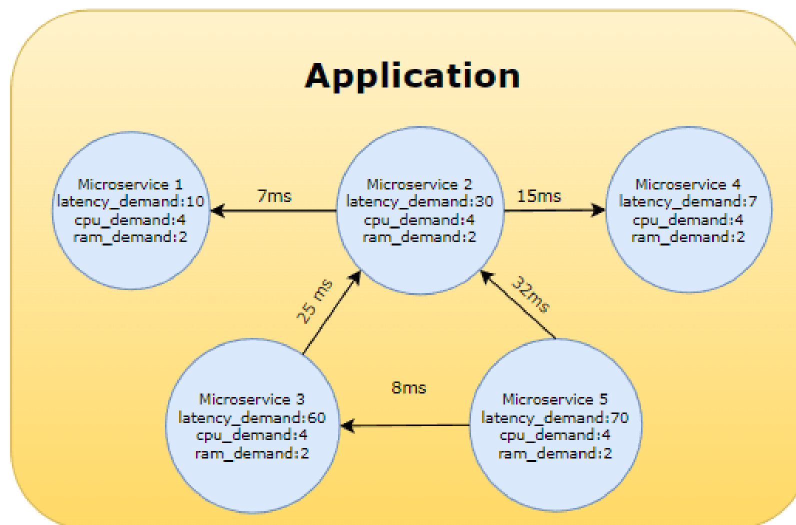


Εικόνα 5-1 Αναπαράσταση της υποδομής

Ο φόρτος εργασίας αφορά εν προκειμένω cloud-native εφαρμογές που χρησιμοποιούν την τεχνολογία των microservices. Κάθε τέτοια εφαρμογή διαιρείται σε έναν αριθμό microservices, καθένα εκ των οποίων φέρει συγκεκριμένες ανάγκες σε πόρους. Ακόμη, κάθε microservice χαρακτηρίζεται από ένα άνω όριο καθυστέρησης, που εκφράζει τη μέγιστη αποδεκτή καθυστέρηση από και προς τον κόμβο εξυπηρέτησής του (που θα αναλάβει δηλαδή την εκτέλεση του container στο οποίο θα ενθυλακωθεί). Χάριν απλότητας, μπορούμε να θεωρήσουμε πως οι εφαρμογές προς εξυπηρέτηση συγκεντρώνονται σε μια FIFO (First-In, First-Out) ουρά σε ένα συγκεκριμένο χωρικό σημείο (στον κόμβο παραγωγής δεδομένων *s*), από το οποίο μετράμε τις καθυστερήσεις προς τους επιμέρους κόμβους της υποδομής. Τέλος, κάθε microservice φέρει ένα άνω όριο σχετικής καθυστέρησης με

ένα ή περισσότερα άλλα microservices της ίδιας εφαρμογής, που εκφράζει τη μέγιστη αποδεκτή καθυστέρηση μεταξύ των κόμβων εξυπηρέτησης των microservices αυτών. Το μέγεθος αυτό είναι ένα δείγμα για το βαθμό συσχέτισης μεταξύ των microservices. Όταν κάποια από τα microservices μιας εφαρμογής παρουσιάζουν υψηλό βαθμό συσχέτισης, αναμένεται συχνή μεταξύ τους επικοινωνία και άρα κρίνεται σκόπιμη η τοποθέτησή τους σε γειτονικούς κόμβους (δηλαδή φέρουν χαμηλό άνω όριο σχετικής καθυστέρησης μεταξύ τους). Επιπλέον, συχνά χρησιμοποιούνται ομοιώματα (duplicates) των microservices για την ανταπόκριση στην αυξημένη ζήτηση (οριζόντια κλιμάκωση – horizontal scaling) ή για λόγους ανθεκτικότητας της εφαρμογής σε βλάβες. Στις περιπτώσεις αυτές συνήθως επιλέγονται και πάλι κόμβοι εύκολα προσπελάσιμοι μεταξύ τους, και άρα σε κοντινή φυσική απόσταση.

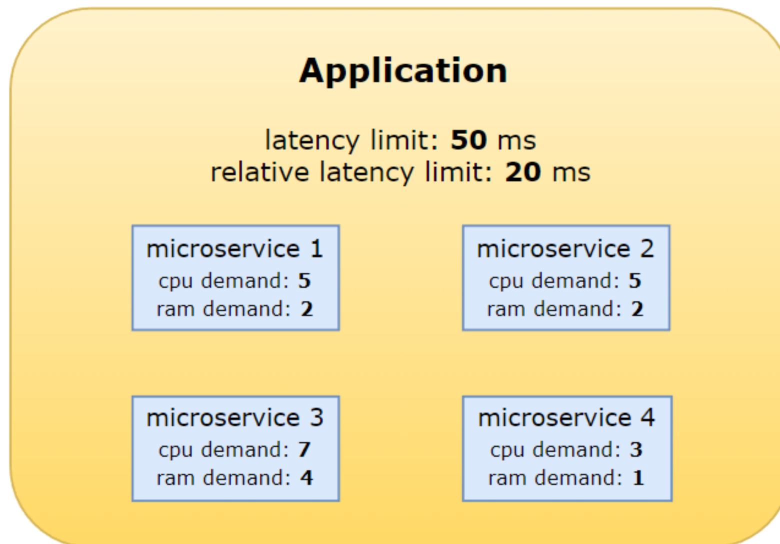
Η κάθε εφαρμογή μπορεί να λάβει τη μορφή ενός γράφου με βάρη $\boldsymbol{G^a} = (\boldsymbol{V^a}, \boldsymbol{E^a}, \boldsymbol{C^a}, \boldsymbol{R^a}, \boldsymbol{L^a}, \boldsymbol{RL^a})$. Κάθε κόμβος $k \in \boldsymbol{V^a}$ αποτελεί ένα microservice της εφαρμογής και χαρακτηρίζεται από την ζήτησή του σε επεξεργαστική ισχύ C_k^a, τη ζήτηση σε μνήμη RAM R_k^a και το άνω όριο καθυστέρησης $L_k^a \leq L_{i,s}^P$, όπου *i* ο κόμβος εξυπηρέτησης της υποδομής. Ακόμη, κάθε ακμή μεταξύ 2 κόμβων k_1, k_2 εκφράζει το άνω όριο σχετικής καθυστέρησης $RL_{k_1,k_2}^a \leq L_{i_1,i_2}^P$, όπου i_1 ο κόμβος εξυπηρέτησης της υποδομής για το microservice k_1, και αντίστοιχα i_2 ο κόμβος εξυπηρέτησης για το microservice k_2.



Εικόνα 5-2 Αναπαράσταση μιας εφαρμογής υπό τη μορφή γράφου

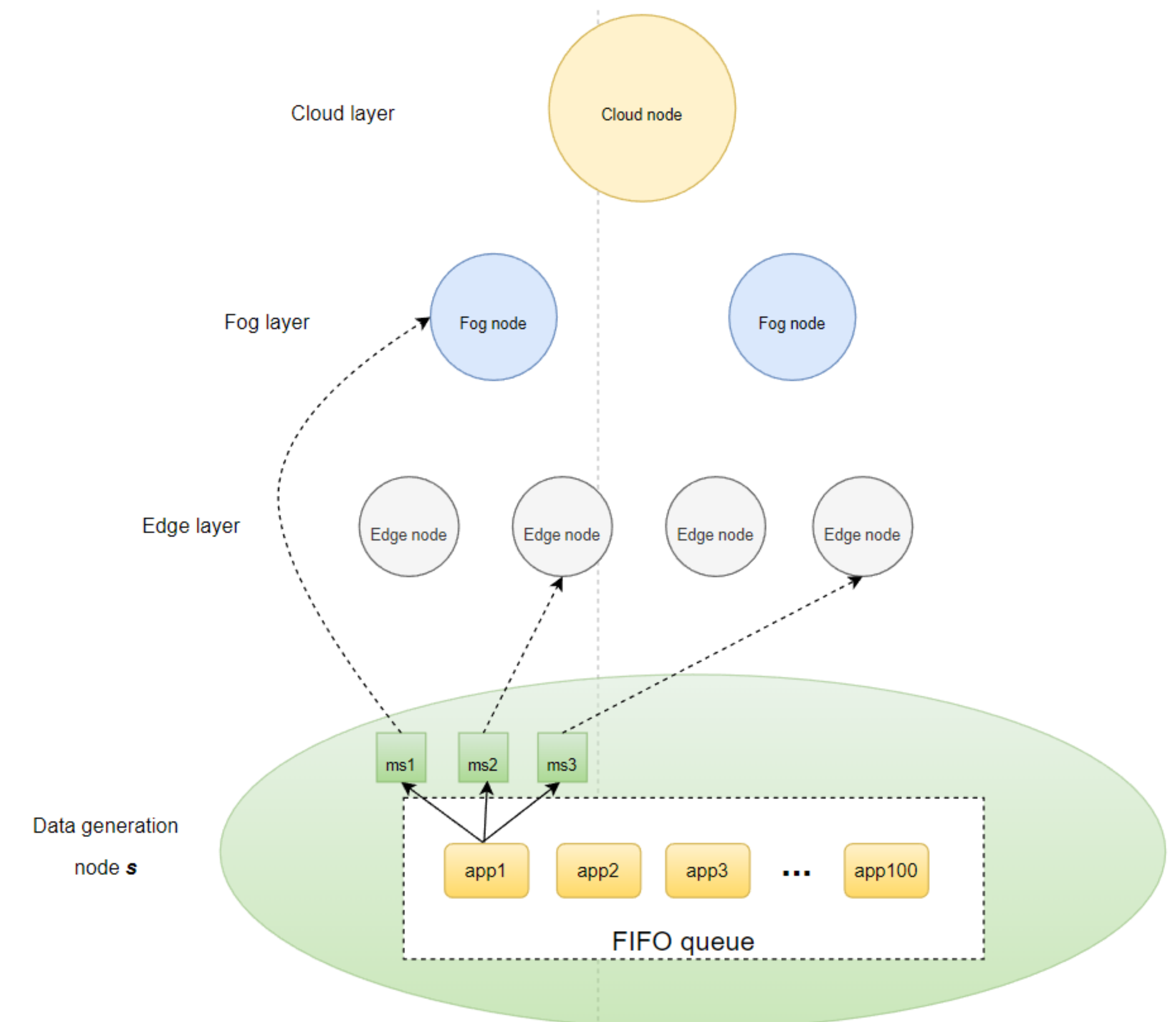
Ωστόσο, για την ευκολότερη αντιμετώπιση του προβλήματος μπορούμε να συμπτύξουμε τα άνω όρια καθυστέρησης των microservices, καθώς και τα άνω όρια σχετικής καθυστέρησης μεταξύ τους, σε δύο τελικά μεγέθη που αφορούν ολόκληρη την εφαρμογή (και άρα κάθε microservice αυτής). Καταλήγουμε δηλαδή σε ένα συνολικό άνω όριο καθυστέρησης $\boldsymbol{L_j}$ για κάθε εφαρμογή *j*, που εκφράζει τη μέγιστη αποδεκτή καθυστέρηση καθενός εκ των κόμβων της υποδομής που θα αναλάβουν microservices της εφαρμογής αυτής σε σχέση με τον κόμβο παραγωγής δεδομένων, καθώς και σε ένα συνολικό άνω όριο σχετικής καθυστέρησης $\boldsymbol{RL_j}$, που εκφράζει τη

μέγιστη αποδεκτή καθυστέρηση μεταξύ των κόμβων που θα αναλάβουν microservices της εφαρμογής αυτής. Η τελική αναπαράσταση των εφαρμογών που θα θεωρηθούν στο πρόβλημα φαίνεται παρακάτω:



Εικόνα 5-3 Αναπαράσταση μιας τυπικής εφαρμογής του προβλήματος

Επομένως, κάθε εφαρμογή j χαρακτηρίζεται από το διάνυσμα $[L_j, RL_j, K_j]$, όπου L_j το άνω όριο καθυστέρησης, RL_j το άνω όριο σχετικής καθυστέρησης και K_j ο αριθμός των microservices της εφαρμογής. Ακόμη, κάθε microservice k της εφαρμογής j χαρακτηρίζεται από το διάνυσμα $[mc_{jk}, mr_{jk}]$, όπου mc_{jk} η ζήτησή του σε επεξεργαστική ισχύ, και mr_{jk} η ζήτησή του σε μνήμη RAM mr_{jk}.



Εικόνα 5-4 Παράδειγμα αντιστοίχισης microservices σε κόμβους της τοπολογίας. Οι εφαρμογές συγκεντρώνονται σε FIFO ουρά στον κόμβο παραγωγής δεδομένων. Τα microservices της 1ης εφαρμογής στην ουρά αντιστοιχίζονται σε κόμβους της τοπολογίας, σύμφωνα με τη συνάρτηση προς βελτιστοποίηση και με σεβασμό στους περιορισμούς του προβλήματος.

Το ζητούμενο είναι η αντιστοίχιση όλων των microservices των εφαρμογών σε κόμβους , με σκοπό την ελαχιστοποίηση ενός βεβαρυμμένου συνδυασμού της καθυστέρησης και του κόστους, με σεβασμό τόσο στον περιορισμό της χωρητικότητας των κόμβων σε πόρους, όσο και στο άνω όριο καθυστέρησης και στο άνω όριο σχετικής καθυστέρησης των εφαρμογών. Ως καθυστέρηση για το microservice k λογίζεται η καθυστέρηση του κόμβου στον οποίο θα τοποθετηθεί , και αντίστοιχα ως κόστος λογίζεται το κόστος του κόμβου τοποθέτησης. Το πρόβλημα αναπαρίσταται μαθηματικά από το ακόλουθο πρόβλημα ακέραιου γραμμικού προγραμματισμού:

$$minimize \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{K_j} w * CS_i^p * x_{ijk} + (1-w) * L_{i,s}^p * x_{ijk}$$

$$s.t. \quad L_{i,s}^p * x_{ijk} \le L_j \qquad \forall\, i = 1,\dots,N \;, j = 1,..,M \;, k = 1,\dots,K_j$$

$$L_{i_1,i_2}^p * x_{i_1jk} + L_{i_1,i_2}^p * x_{i_2jk'} \le RL_j + L_{i_1,i_2}^p \qquad \forall\, i1 = 1,\dots,N \quad i2 = 1,\dots,N \quad \forall k,k' = 1,\dots,K_j\; j = 1,\dots,M$$

$$\sum_{j=1}^{M}\sum_{k=1}^{K_j} mc_{jk} * x_{ijk} \le C_i^p \quad \forall\, i = 1,\dots,N$$

$$\sum_{j=1}^{M}\sum_{k=1}^{K_j} mr_{jk} * x_{ijk} \le R_i^p \quad \forall\, i = 1,\dots,N$$

$$\sum_{i=1}^{N} x_{ijk} = 1 \qquad \forall\, j = 1,..,M \;, k = 1,\dots,K_j$$

Όπου:

- $\boldsymbol{x_{ijk}}$ η δυαδική (binary) μεταβλητή απόφασης που ισούται με 1 εάν το microservice *k* της *j* εφαρμογής τοποθετείται τελικά στον κόμβο *i* και 0 διαφορετικά, $i = 1,\dots,N \;, j = 1,\dots,M \;, k = 1,\dots,K_j$
- *N* ο συνολικός αριθμός κόμβων
- *M* ο συνολικός αριθμός εφαρμογών
- K_j ο συνολικός αριθμός των microservices της εφαρμογής j, $j = 1,\dots,M$
- $\boldsymbol{L_{i,s}^p}$ η καθυστέρηση του κόμβου *i* από τον κόμβο παραγωγής δεδομένων *s*, $i = 1,\dots.,N$
- $\boldsymbol{CS_i^p}$ το κόστος του κόμβου *i* $i = 1,\dots.,N$
- $\boldsymbol{w}$ το βάρος $\in (0,1)$
- $\boldsymbol{D_j}$ το άνω όριο επιτρεπτής καθυστέρησης για την εφαρμογή *j*, $j = 1,\dots,M$
- $\boldsymbol{RL_j}$ το άνω όριο επιτρεπτής σχετικής καθυστέρησης για την εφαρμογή *j*, $j = 1,\dots,M$
- $\boldsymbol{L_{i_1,i_2}^p}$ η σχετική καθυστέρηση μεταξύ των κόμβων i_1 και i_2 , $i_1 = 1,\dots,N \;, i_2 = 1,\dots,N$
- $\boldsymbol{C_i^p}$ η επεξεργαστική χωρητικότητα του κόμβου i, $i = 1,\dots,N$
- mc_{jk} η ζήτηση σε επεξεργαστική ισχύ από το microservice k της εφαρμογής j, $j = 1,\dots,M \;, k = 1,\dots,K_j$
- $\boldsymbol{R_i^p}$ η χωρητικότητα μνήμης RAM του κόμβου $i, i = 1,\dots,N$
- mr_{jk} η ζήτηση σε μνήμη RAM από το microservice k της εφαρμογής $j, j = 1,\dots,M \;, k = 1,\dots,K_j$

Η αντικειμενική συνάρτηση είναι το βεβαρυμμένο άθροισμα καθυστέρησης και κόστους για όλα τα microservices όλων των εφαρμογών , όπου η τιμή *w*=0 εξετάζει αμιγώς το πρόβλημα ελαχιστοποίησης της καθυστέρησης, ενώ η τιμή *w*=1 το

πρόβλημα ελαχιστοποίησης του κόστους, ενώ οποιαδήποτε ενδιάμεση τιμή λαμβάνει υπόψη και τους 2 αυτούς όρους.

Ο πρώτος περιορισμός επιβάλλει την καθυστέρηση του κόμβου επιλογής κάθε microservice να μην ξεπερνά το άνω όριο καθυστέρησης της εφαρμογής. Ο δεύτερος περιορισμός επιβάλλει την σχετική καθυστέρηση μεταξύ οποιωνδήποτε 2 κόμβων που ανέλαβαν microservices από την ίδια εφαρμογή να μην ξεπερνά το άνω όριο σχετικής καθυστέρησης της εφαρμογής. Ο τρίτος περιορισμός διασφαλίζει την τήρηση του περιορισμού επεξεργαστικής χωρητικότητας κάθε κόμβου, δηλαδή επιβάλλει την αθροιστική επεξεργαστική ζήτηση από όλα τα microservices που τελικά τοποθετούνται σε αυτόν να μην ξεπερνά τη χωρητικότητα του. Ο τέταρτος περιορισμός είναι ίδιας λογικής με τον τρίτο, αλλά για τη μνήμη RAM. Ο τελευταίος περιορισμός εγγυάται πως κάθε microservice θα εξυπηρετηθεί από έναν, ακριβώς, κόμβο.

Έστω πως έχουμε *M* εφαρμογές, με *K* microservices η κάθε μία, δηλαδή ένα φόρτο εργασίας απαρτιζόμενο από $M * K$ microservices. Οι πιθανοί συνδυασμοί για την τοποθέτησή τους στους κόμβους της τοπολογίας ανέρχονται σε $N^{(M*K)}$, όπου *N* ο αριθμός των κόμβων. Εύκολα παρατηρούμε πως οι συνδυασμοί μπορούν εύκολα να φτάσουν τεράστια μεγέθη (για παράδειγμα, για *M*=5, *K*=2 και *N*=10 , προκύπτουν 10000000000 [δέκα δισεκατομμύρια] πιθανοί συνδυασμοί, εκ των οποίων φυσικά πολλοί οδηγούν σε μη εφικτές λύσεις εξαιτίας των περιορισμών), καθιστώντας την ακριβή λύση του παραπάνω προβλήματος χρονικά και υπολογιστικά ασύμφορη.

Συνεπώς, για την επίλυση του προβλήματος αρχικά θα κατασκευάσουμε έναν ισχυρό προσεγγιστικό αλγόριθμο που βρίσκει τη βέλτιστη τοποθέτηση κάθε εφαρμογής ξεχωριστά, δεδομένου των περιορισμών στη χωρητικότητα από τις εφαρμογές που τοποθετήθηκαν πρωτύτερα. Στη συνέχεια, θα παρουσιάσουμε την τεχνική του ξετυλίγματος (Rollout), μια μετά-ευρετική τεχνική που χρησιμοποιείται ευρέως στον τομέα της μηχανικής μάθησης, και παρέχει μια βελτιωμένη λύση μέσω ενός επαναληπτικού μηχανισμού. Για την αποτελεσματική χρήση του Rollout, θα κατασκευάσουμε έναν νέο προσεγγιστικό αλγόριθμο, απλούστερο στη δομή αλλά πολύ ταχύτερο από τον πρώτο. Παρακάτω παρουσιάζεται ο πρώτος προσεγγιστικός αλγόριθμος:

5.2 Περιγραφή του αλγορίθμου GRAA (Greedy Resource Allocation Algorithm)

5.2.1 Είσοδοι στον αλγόριθμο

Θεωρούμε ένα σύνολο κόμβων σε κάθε στρώμα (edge,fog,cloud). Λαμβάνοντας υπόψη τα χαρακτηριστικά τους στα δεδομένα του προβλήματος, μπορούμε να δημιουργήσουμε τα εξής διανύσματα:

- Διάνυσμα επεξεργαστικής χωρητικότητας $c = [c_e\ c_f\ c_c]$, όπου $c_e=[c_{e1},\dots,c_{en1}]$ οι χωρητικότητες, σε cpu units, των *n1* κόμβων του edge στρώματος, και ομοίως $c_f=[c_{f1},\dots,c_{fn2}]$ για τους *n2* κόμβους του fog στρώματος και $c_c=[c_{c1},\dots,c_{cn3}]$ για τους *n3* κόμβους του cloud (*n1+n2+n3=N*).

- Διάνυσμα χωρητικότητας μνήμης RAM $r = [r_e\ r_f\ r_c]$, όπου με αντίστοιχη λογική έχουμε $r_e=[r_{e1},...,r_{en1}]$, $r_f=[r_{f1},...,r_{fn2}]$, $r_c=[r_{c1},...,r_{cn3}]$.
- Διάνυσμα καθυστέρησης $l = [l_e\ l_f\ l_c]$.
- Διάνυσμα κόστους $cost = [cost_e\ cost_f\ cost_c]$.

Ακόμη, για να συμπεριλάβουμε τις σχετικές καθυστερήσεις μεταξύ των κόμβων δημιουργούμε τον NxN πίνακα σχετικών καθυστερήσεων *rl*, όπου το στοιχείο $rl(i_1, i_2)$ εκφράζει την καθυστέρηση του κόμβου i_1 προς τον κόμβο i_2. Θωρούμε επίσης ότι ο *rl* είναι συμμετρικός, ήτοι $rl(i_1, i_2) = rl(i_2, i_1)$, και ότι η καθυστέρηση ενός κόμβου με τον εαυτό του είναι μηδενική, ήτοι $rl(i_1, i_1) = 0$.

Από την άλλη, οι εφαρμογές χαρακτηρίζονται από τα κάτωθι διανύσματα:

- Διάνυσμα αριθμού microservices $K = [K_1, \dots, K_M]$, όπου το στοιχείο K_j είναι ο αριθμός των microservices της εφαρμογής *j*.
- Διάνυσμα άνω ορίων καθυστέρησης $lm = [lm_1, \dots, lm_M]$, όπου το στοιχείο lm_j είναι το άνω όριο καθυστέρησης της εφαρμογής *j*.
- Διάνυσμα άνω ορίων σχετικής καθυστέρησης $rlm = [rlm_1, \dots, rlm_M]$, όπου το στοιχείο rlm_j είναι το άνω όριο σχετικής καθυστέρησης της εφαρμογής *j*.

Τέλος, τα microservices κάθε εφαρμογής δημιουργούν τα διανύσματα:

- Διάνυσμα ζήτησης σε επεξεργαστική ισχύ $mc = [mc_{j1}, \dots, mc_{jKj}]\ \forall j = 1, \dots M$, όπου το στοιχείο mc_{jk} είναι η ζήτηση σε επεξεργαστική ισχύ του *k*-οστού microservice της εφαρμογής εφαρμογής *j*.
- Διάνυσμα ζήτησης σε μνήμη RAM $mr = [mr_{j1}, \dots, mr_{jKj}]\ \forall j = 1, \dots M$, όπου το στοιχείο mr_{jk} είναι η ζήτηση σε μνήμη RAM του *k*-οστού microservice της εφαρμογής εφαρμογής *j*.

5.2.2 Ο αλγόριθμος

Κεντρική ιδέα

Ο αλγόριθμος επιδιώκει την εύρεση μιας ικανοποιητικής υπό-βέλτιστης λύσης ως εξής: Θεωρώντας πως οι εφαρμογές σχηματίζουν μια FIFO ουρά, επιλέγει σε κάθε βήμα το βέλτιστο συνδυασμό κόμβων για την τοποθέτηση των microservices της εφαρμογής του βήματος, με σκοπό την ελαχιστοποίηση του πρόσθετου όρου που θα εισάγει η τοποθέτηση στην αντικειμενική συνάρτηση. Πρόκειται δηλαδή για έναν άπληστο (greedy) αλγόριθμο [45], με μια μικρή τροποποίηση: Όταν το βάρος της αντικειμενικής συνάρτησης είναι μικρό ($w \sim 0$), και άρα το πρόβλημα αφορά στη βελτιστοποίηση της καθυστέρησης, η τυχαία σειρά εξυπηρέτησης των εφαρμογών μπορεί να δημιουργήσει έντονα φαινόμενα μπλοκαρίσματος (blocking). Αυτό συμβαίνει διότι εφαρμογές που δεν έχουν πραγματική ανάγκη για χαμηλή καθυστέρηση (δεν έχουν δηλαδή, χαμηλό άνω όριο καθυστέρησης) θα τοποθετηθούν στους κόμβους των χαμηλότερων στρωμάτων (εξαιτίας της αντικειμενικής συνάρτησης) εξαντλώντας τους πόρους τους, και έτσι μετέπειτα εμφανιζόμενες εφαρμογές δε θα μπορούν να τοποθετηθούν σε κόμβους που να ικανοποιούν το,

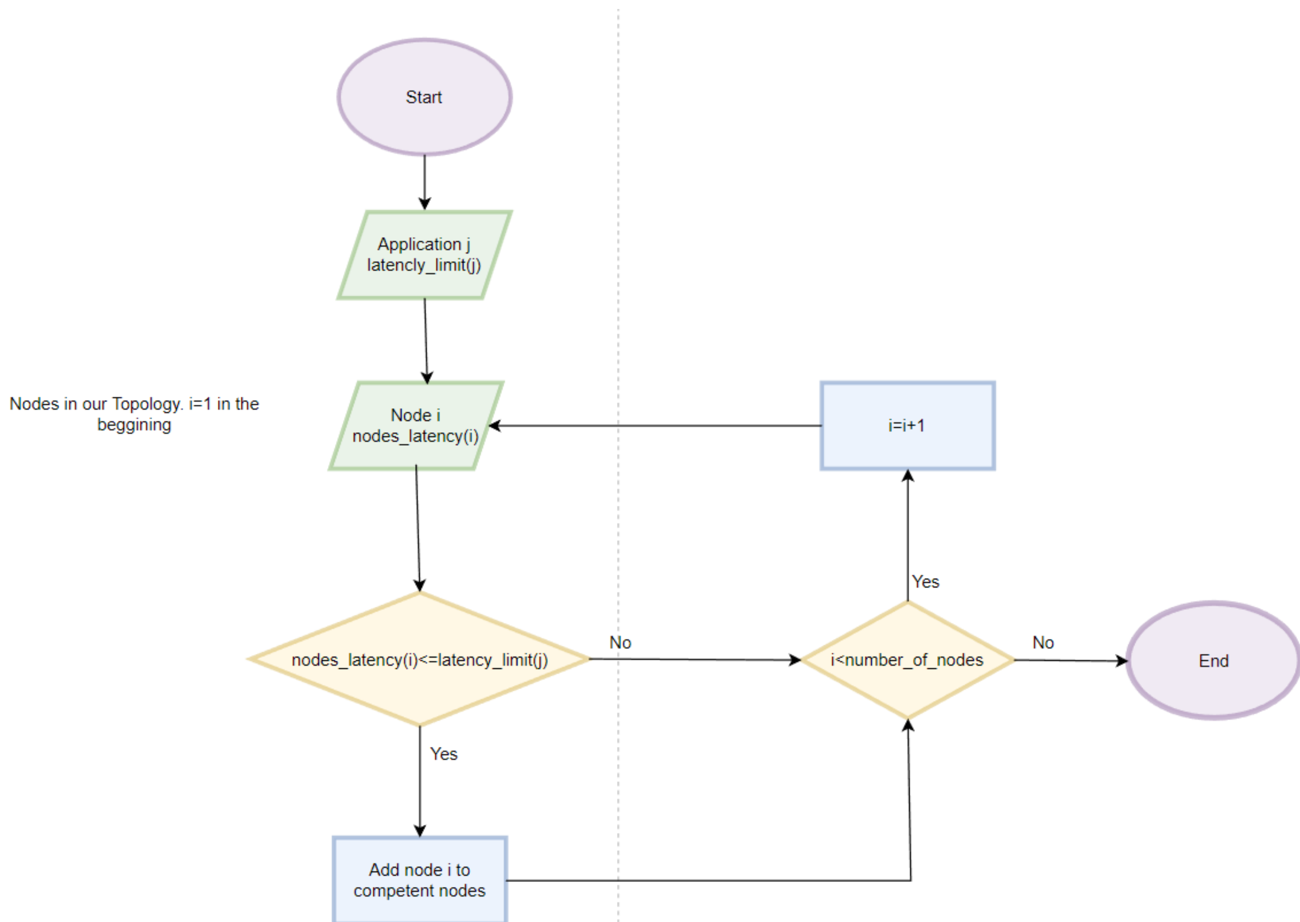
ενδεχομένως χαμηλό, άνω όριο καθυστέρησής τους. Για την αποφυγή του παραπάνω φαινομένου ο αλγόριθμος κατατάσσει αρχικά τις εφαρμογές σύμφωνα με το άνω όριο καθυστέρησης, σε αύξουσα σειρά. Παρακάτω θα εξετάσουμε μια τυπική επανάληψη του αλγορίθμου για την τοποθέτηση των microservices μιας εφαρμογής.

Βήμα 1: Εύρεση των κόμβων που ικανοποιούν το άνω όριο καθυστέρησης

Από όλους τους κόμβους της υποδομής, ο αλγόριθμος ξεχωρίζει τους «ικανούς» , δηλαδή αυτούς που ικανοποιούν το άνω όριο καθυστέρησης της εφαρμογής. Η διαδικασία παρουσιάζεται στον παρακάτω υπό-αλγόριθμο σε ψευδογλώσσα καθώς και με τη μορφή διαγράμματος ροής:

Αλγόριθμος 5.1 Εύρεση ικανών κόμβων

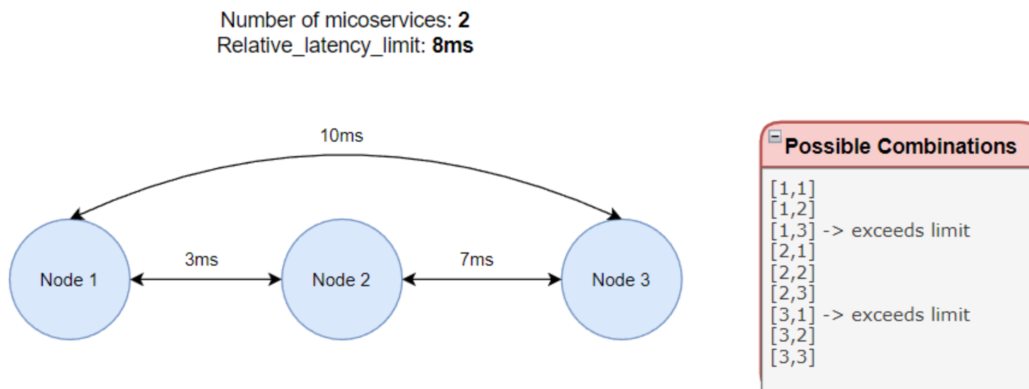
```
1. Input (nodes_latency_vector l, latency limit lm_j)
2. Output(competent nodes)
3. For every node i
4.     If (l_i ≤ lm_j)
5.     Add node i to competent nodes
6.     EndIf
7. EndFor
```



Εικόνα 5-5 Διάγραμμα ροής βήματος 1

Βήμα 2: Εύρεση όλων των πιθανών συνδυασμών και απαλοιφή αυτών που δεν ικανοποιούν το άνω όριο σχετικής καθυστέρησης.

Για όλους του ικανούς κόμβους που βρέθηκαν στο προηγούμενο βήμα, υπολογίζουμε όλους τους πιθανούς συνδυασμούς μήκους K_j για την τοποθέτηση των microservices της εφαρμογής j. Από αυτούς, απαλείφουμε κάθε συνδυασμό για τον οποίο παραβιάζεται το άνω όριο σχετικής καθυστέρησης μεταξύ οποιονδήποτε 2 κόμβων του συνδυασμού.

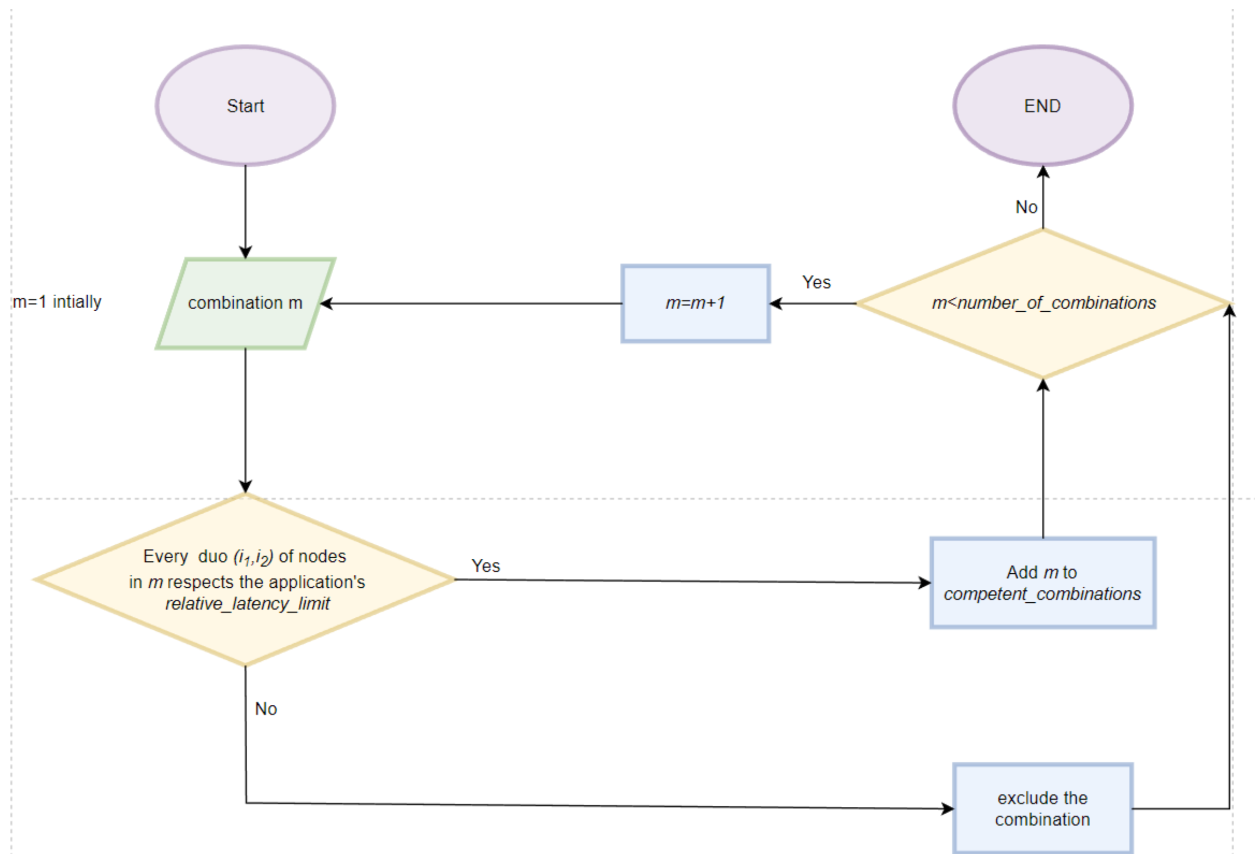


Εικόνα 5-6 Παράδειγμα βήματος 2

Στην παραπάνω εικόνα παρουσιάζεται η διαδικασία για 3 κόμβους, από τους οποίους προκύπτουν 9 συνδυασμοί για την τοποθέτηση μιας εφαρμογής με 2 microservices. O συνδυασμός (i_1, i_2) σημαίνει πως το πρώτο microservice θα τοποθετηθεί στον κόμβο i_1, και το δεύτερο στον i_2. Κάποιοι συνδυασμοί απαρτίζονται από κόμβους που δεν ικανοποιούν το άνω όριο σχετικής καθυστέρησης της εφαρμογής, οπότε απορρίπτονται. Η διαδικασία φαίνεται παρακάτω σε ψευδογλώσσα και με τη μορφή διαγράμματος ροής:

Αλγόριθμος 5.2 Εύρεση συνδυασμών που ικανοποιούν το άνω όριο σχετικής καθυστέρησης

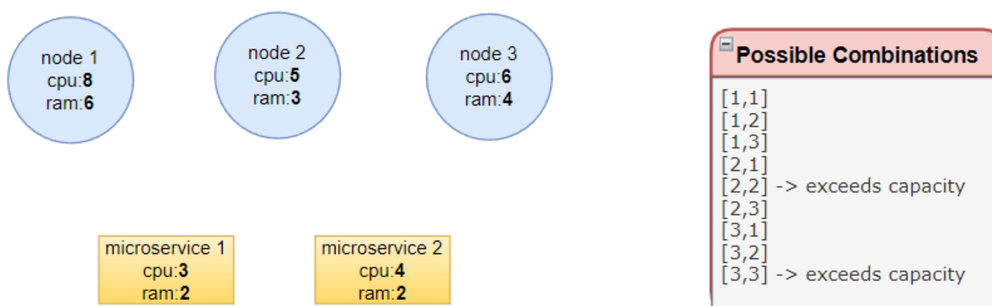
1. Input (relative latency matrix *rl*, relative latency limit rlm_j, *combinations*)
2. Output(*competent_ combinations*)
3. For every combination *m*
4. *violation_ flag*=0;
5. For every node *i1* in *m*
6. For every node *i2*>*i1* in *m*
7. If $rl(i, i_c) > rlm_j$)
8. *violation_ flag*=1;
9. EndFor
10. EndFor
11. If (*violation_ flag* == 1)
12. Delete combination m
13. Else
14. Add *m* to competent_combinations
15. EndIf
16. EndFor



Εικόνα 5-7 Διάγραμμα ροής βήματος 2

Βήμα 3: Απαλοιφή των συνδυασμών που δεν ικανοποιούν τους περιορισμούς χωρητικότητας.

Από τους εναπομείναντες συνδυασμούς, απαλείφουμε αυτούς στους οποίους τουλάχιστον ένας κόμβος δεν ικανοποιεί τους περιορισμούς της χωρητικότητας σε επεξεργαστική ισχύ ή μνήμη RAM.



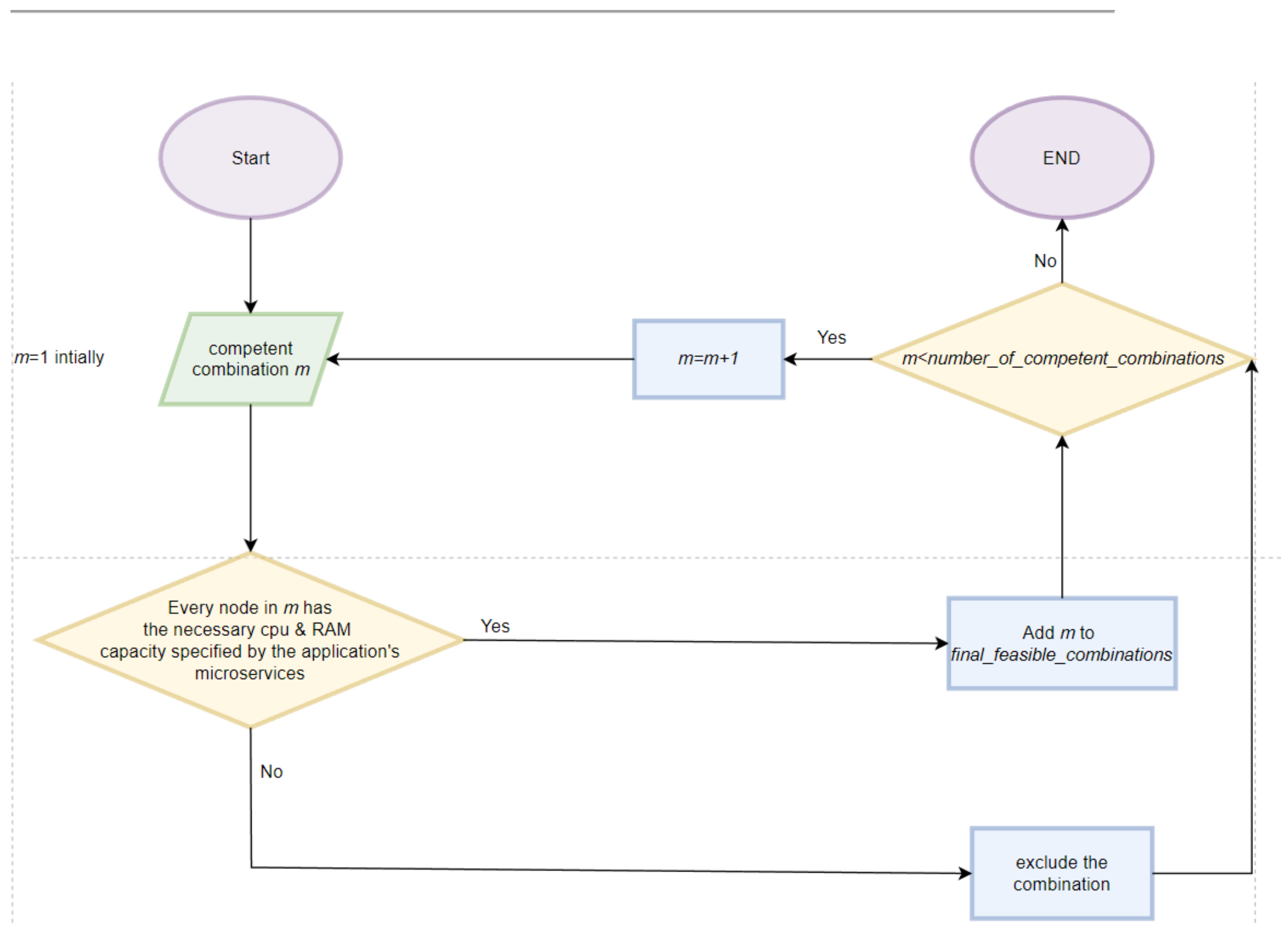
Εικόνα 5-8 Παράδειγμα βήματος 3

Στην εικόνα παρατηρούμε τη διαδικασία για μια εφαρμογή με 2 microservices, και ένα σύνολο 3 κόμβων .Ο συνδυασμός (i_1, i_2) σημαίνει πως το πρώτο microservice θα τοποθετηθεί στον κόμβο i_1, και το δεύτερο στον i_2. Κάποιοι συνδυασμοί

παραβιάζουν την χωρητικότητα (cpu ή/και RAM), οπότε απορρίπτονται. Η διαδικασία φαίνεται παρακάτω σε ψευδογλώσσα και σε διάγραμμα ροής:

Αλγόριθμος 5.3 Εύρεση συνδυασμών που δεν παραβιάζουν τον περιορισμό της χωρητικότητας

1. Inputs (competent_*combinations*, nodes cpu vector *c*, nodes ram vector *r*, number of microservices *Kj*, processing demands mc_j, ram demands mr_j)
2. Outputs (*final_feasible_combinations*)
3. For every competent_combination *m*
4. Add up the needed resources nc_i, nr_i from the combination for each node *i*
5. If ($c_i \geq nc_i$ && $r_i \geq nr_i$) ∀ node *i* in the combination
6. Add *m* to *final_feasible_combinations*
7. EndIf
8. EndFor



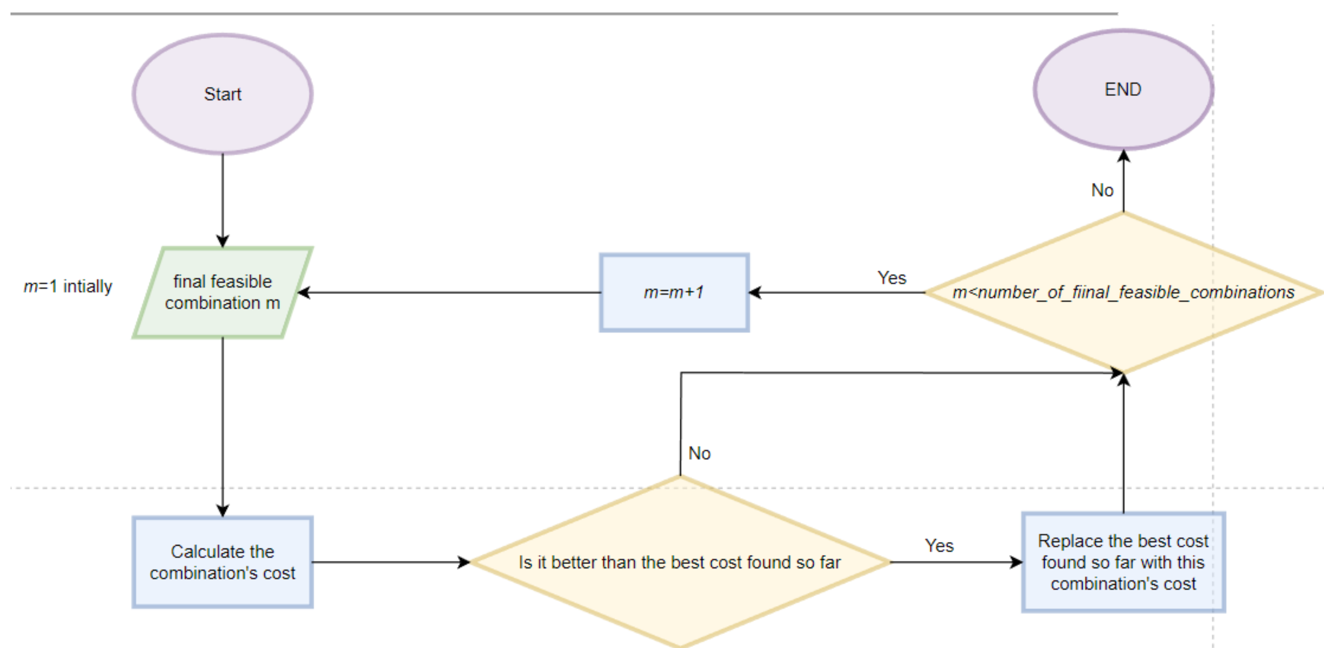
Εικόνα 5-9 Διάγραμμα ροής βήματος 3

Βήμα 4: Εύρεση του βέλτιστου συνδυασμού

Από τους τελικούς εφικτούς συνδυασμούς, επιλέγουμε αυτόν που εγείρει το μικρότερο «κόστος» δηλαδή την μικρότερη πρόσθετη τιμή στην αντικειμενικής συνάρτηση .

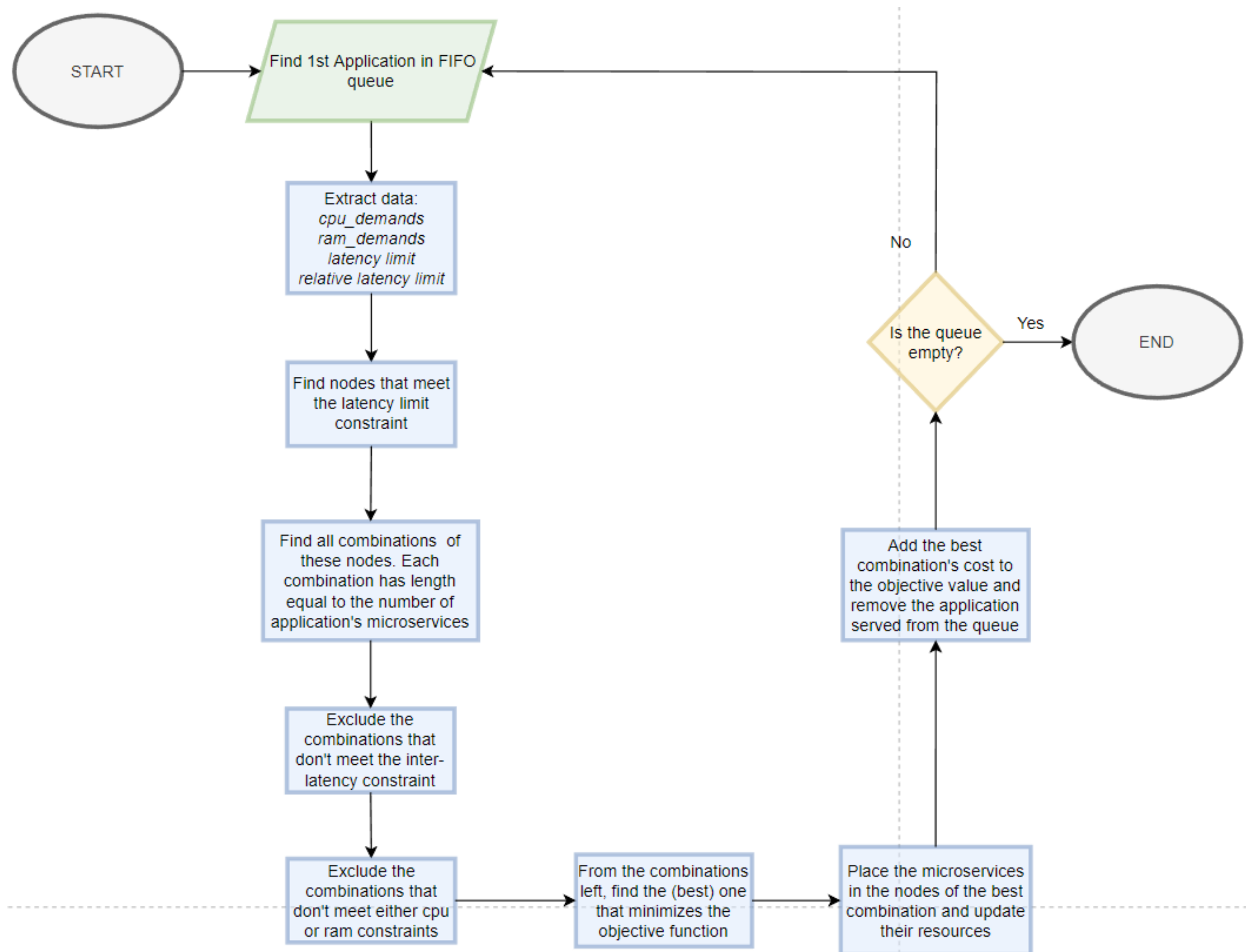
Αλγόριθμος 5.4 Εύρεση του βέλτιστου εφικτού συνδυασμού

1. Inputs (*final_ feasible_ combinations*, nodes cost *cost*, nodes latency *l*)
2. Outputs (*best_ combination*, *best_ added_ value*)
3. *best_ added_ value*=1000;
4. For every *j in final feasible combinations*
5. For every node *i* in *j*
6. *added_ value*=$w*cost_i + (1-w)*l_i$
7. EndFor
8. If (*added_ value* < *best*)
9. *best_ added_ value=added_ value;*
10. *best_ combination=j;*
11. EndIf
12. EndFor



Εικόνα 5-10 Διάγραμμα ροής βήματος 4

Στη συνέχεια ο αλγόριθμος δεσμεύει τους απαραίτητους πόρους από τους κόμβους του καλύτερου συνδυασμού, τοποθετεί τα microservices και συνεχίζει στην επόμενη εφαρμογή. Σημειώνεται επίσης πως αν κανένας συνδυασμός δεν ικανοποιεί κάποιον από τους περιορισμούς, ο αλγόριθμος εμφανίζει μήνυμα για το μπλοκάρισμα της εφαρμογής. Παρακάτω φαίνεται το συνολικό διάγραμμα ροής για τον GRRA:



Εικόνα 5-11 Συνολικό διάγραμμα ροής για τον GRAA

5.3 Αποτελέσματα και αξιολόγηση

Η υλοποίηση και η εκτέλεση των αλγορίθμων θα γίνει με χρήση του Matlab [46]. Το Matlab (Matrix laboratory) είναι ένα περιβάλλον αριθμητικής υπολογιστικής που φέρει τη δική του προγραμματιστική γλώσσα. Βασισμένο στη μητρωική άλγεβρα, αποτελεί ένα ισχυρότατο εργαλείο για την επίλυση μαθηματικών προβλημάτων, συγκεντρώνοντας εκατομμύρια χρήστες προερχόμενους από υπόβαθρα μηχανικής, επιστημών και οικονομικών. Καθώς το πρόβλημά μας αποτελεί ένα μαθηματικό πρόβλημα βελτιστοποίησης, το Matlab προσφέρει την ευκολότερη υλοποίηση χωρίς να εμπλέκει ιδιαίτερες τεχνικές λεπτομέρειες. Φυσικά η επιλογή αυτή δεν είναι δεσμευτική.

Για την προσομοίωση θα θεωρήσουμε τα δεδομένα για την υποδομή που φαίνονται στον παρακάτω πίνακα:

	Αριθμός κόμβων	Αριθμός cpu units/κόμβο	Αριθμός ram units/κόμβο	Καθυστέρηση latency units	Κόστος Cost units
Στρώμα edge	10	15	4-8	0.5-2	6-9
Στρώμα fog	3	100	64	3.5-4.2	3.3-4.3
Στρώμα cloud	2	500	256	7.5-8.5	1-1.5

Πίνακας 5-1 Στοιχεία της πειραματικής υποδομής του GRAA

Η καθυστέρηση και το κόστος λαμβάνουν κανονικοποιημένες τιμές μετρημένες σε μονάδες καθυστέρησης (latency units) και μονάδες κόστους (cost units) αντίστοιχα, με το στρώμα cloud να έχει περίπου 10 φορές μεγαλύτερη καθυστέρηση από το στρώμα του edge και 7-10 φορές λιγότερο κόστος, ενώ το fog λαμβάνει ενδιάμεσες τιμές. Σημειώνεται ότι οι τιμές που δίνονται με τη μορφή διαστήματος αντλούνται τυχαία από ομοιόμορφη κατανομή στο αντίστοιχο διάστημα. Ακόμη, στον παρακάτω πίνακα παρουσιάζονται οι τιμές της σχετικής καθυστέρησης για 2 κόμβους (i_1, i_2) σύμφωνα με τα στρώματα της τοπολογίας που εντοπίζεται ο καθένας.

Κόμβοι στρωμάτων	Σχετική Καθυστέρηση
Edge-Edge	0-1
Edge-Fog	3
Edge-Cloud	7
Fog-Fog	0-2
Fog-Cloud	3
Cloud-Cloud	0-3

Πίνακας 5-2 Σχετικές καθυστερήσεις

Τέλος, οι εφαρμογές προσομοιώνονται σύμφωνα με τις παρακάτω τιμές:

	# microservices	cpu demand/ms	ram demand/ms	latency limit	relative latency limit
Applications (1-50)	1-3	1-5	1-2	1.5-10	0.5-5

Πίνακας 5-3 Στοιχεία πειραματικών εφαρμογών

Οι τιμές του latency-limit για τις εφαρμογές εκκινούν από την τιμή 1.5 latency units, ούτως ώστε να εξασφαλίζεται πάντοτε η ύπαρξη ικανών κόμβων για την εξυπηρέτηση. Ο αριθμός των εφαρμογών προς εξυπηρέτηση είναι 50. Φυσικά η παραμετροποίηση των τιμών μπορεί να προσαρμοστεί στην εκάστοτε τοπολογία και στις ανάγκες του πειράματος.

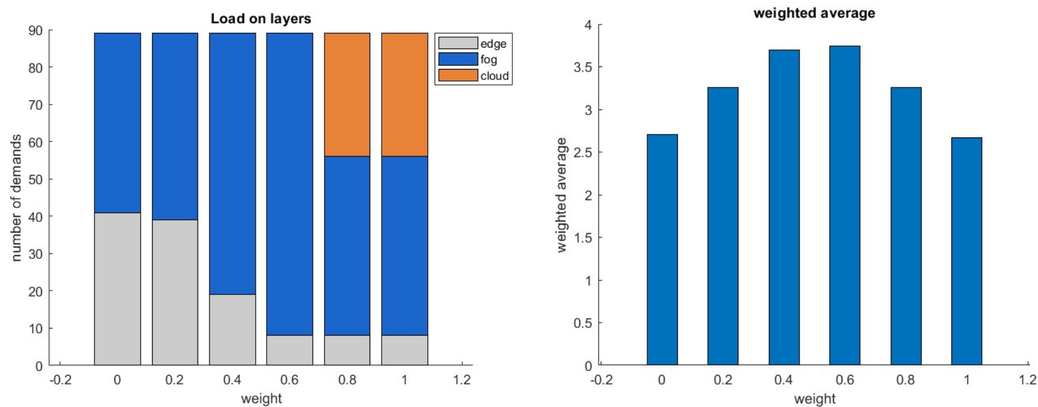
Τα αποτελέσματα της προσομοίωσης δίνονται σε μορφή πίνακα αλλά και διαγράμματος, και αφορούν την κατανομή των microservices των εφαρμογών στα τρία στρώματα edge-fog-cloud καθώς και την τιμή της αντικειμενικής συνάρτησης, διαιρεμένη με τον συνολικό αριθμό microservices των εφαρμογών, για βάρη w από 0

έως 1, με βήμα 0,2.

Αποτελέσματα και αποτίμηση

Βάρος *w*	στρώμα Edge	στρώμα Fog	στρώμα Cloud	Αντικειμενική συνάρτηση
0	41	48	0	2.708
0.2	39	50	0	3.254
0.4	19	70	0	3.692
0.6	8	81	0	3.739
0.8	8	48	33	3.259
1	8	48	33	2.667

Πίνακας 5-4 Αποτελέσματα πειράματος



Εικόνα 5-12 Σχηματικά αποτελέσματα πειράματος. Αριστερά φαίνεται το φορτίο (σε microservices) στα 3 στρώματα της υποδομής (edge-fog-cloud) για διάφορες τιμές του βάρους w, ενώ δεξιά η αντίστοιχη τιμή της αντικειμενικής συνάρτησης.

Με μια πρώτη ματιά στα αποτελέσματα παρατηρούμε ότι τα στρώματα των edge και fog φορτώνονται πάντοτε με κάποιον αριθμό microservices. Αυτό συμβαίνει διότι πάντοτε κάποια microservices, ανεξαρτήτως βάρους, θα πρέπει να τοποθετηθούν στα στρώματα αυτά καθώς οι κόμβοι τους είναι οι μόνοι που ικανοποιούν το άνω όριο καθυστέρησης των αντίστοιχων εφαρμογών. Παρόμοιος είναι και ο λόγος που το cloud στρώμα συναντά ένα μέγιστο στον αριθμό των microservices που μπορεί να φιλοξενήσει , αφού διεκδικεί μόνο αυτά που ανήκουν σε εφαρμογές με αρκετά μεγάλο όριο καθυστέρησης (μεγαλύτερο από την μικρότερη καθυστέρηση που εμφανίζουν οι κόμβοι του στρώματός του). Για την βαθύτερη ανάλυση των αποτελεσμάτων που αφορούν το φόρτο στα διάφορα στρώματα , θα εξετάσουμε τις εξής 2 "διαμάχες":

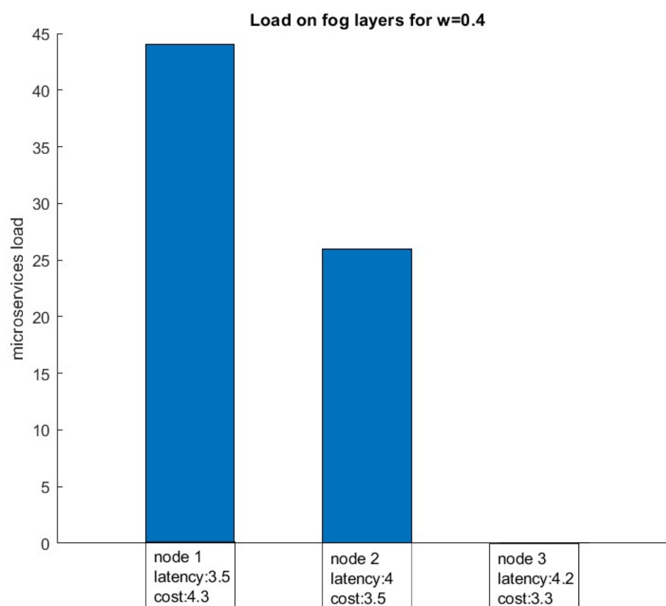
1) Edge-Fog. Όταν το βάρος *w* είναι μικρό, όλα τα microservices που μπορούν να τοποθετηθούν σε αυτά τα 2 στρώματα επιλέγουν το edge , καθώς είναι

σημαντικά ταχύτερο. Αντιθέτως όταν το w μεγαλώνει, τη διαμάχη "νικά" το φθηνότερο στρώμα του Fog.

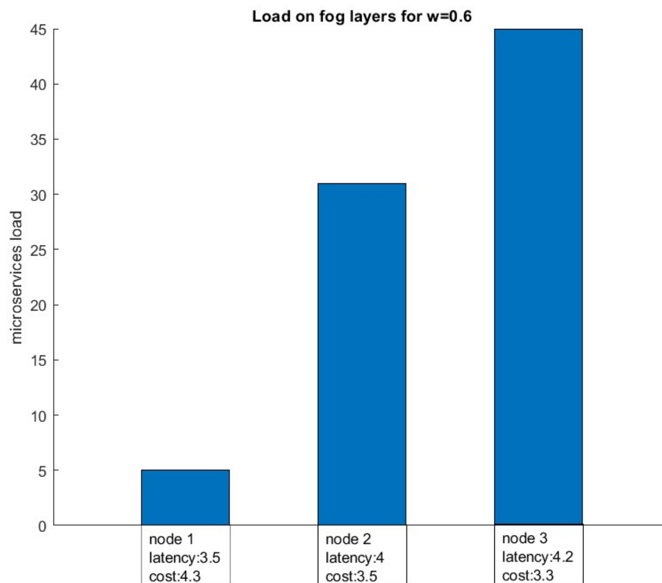
2) Edge-Fog-Cloud. Όταν το βάρος είναι μεγάλο, τα microservices που μπορούν να τοποθετηθούν και στα 3 στρώματα επιλέγουν πάντοτε το πολύ φθηνότερο στρώμα του cloud, ενώ καθώς το w μειώνεται τα αιτήματα αυτά μοιράζονται στα άλλα 2 στρώματα. Ειδικότερα για τις μεσαίες τιμές του βάρους (w=0.4 , w=0.6) , επιλέγεται, κυρίως, το στρώμα του Fog.

Ακόμη, παρατηρούμε ότι οι τιμές της αντικειμενικής συνάρτησης είναι καλύτερες (μικρότερες) για τις ακραίες τιμές του βάρους (w=0, w=1), όπου το πρόβλημα είναι απλοποιημένο σε βελτιστοποίηση της καθυστέρησης ή του κόστους αντίστοιχα, και άρα επιλέγονται όσο το δυνατόν περισσότεροι κόμβοι από τα στρώματα που είναι "ιδανικά" (edge για το w=0, cloud για το w=1). Όταν όμως το w λαμβάνει μεσαίες τιμές (w=0.4, w=0.6) επιλέγονται υπό το πλείστων οι – πιο "μέτριοι" σε χαρακτηριστικά που αφορούν τη βελτιστοποίηση της αντικειμενικής συνάρτησης- κόμβοι του Fog, που προσφέρουν μια δίκαιη ισορροπία μεταξύ καθυστέρησης και κόστους.

Μια ακόμη σημαντική παρατήρηση είναι πως παρόλο που οι τιμές του βάρους αλλάζουν, το φορτίο στα στρώματα δε φαίνεται να επηρεάζεται σημαντικά. Αυτό συμβαίνει επειδή ο αλγόριθμος επιλέγει κάθε φορά διαφορετικούς κόμβους για την τοποθέτηση εντός του ιδίου στρώματος. Για παράδειγμα, ενώ ο αριθμός των microservices που παραλαμβάνει το στρώμα του fog για w=0.4 και w=0.6 δεν αλλάζει σημαντικά , παρακάτω βλέπουμε πως το φορτίο στους κόμβους εντός του fog στρώματος διαφοροποιείται σε πολύ μεγάλο βαθμό:



Εικόνα 5-13 Φορτίο στους κόμβους του Fog στρώματος για βάρος w=0.4



Εικόνα 5-14 Φορτίο στους κόμβους του Fog στρώματος για βάρος w=0.6

Για *w*=0.4, το πρόβλημα εστιάζει ακόμη κατά κύριο λόγο στην ελαχιστοποίηση της καθυστέρησης, συνεπώς ο GRAA επιλέγει τον πρώτο κόμβο (που φέρει τη χαμηλότερη καθυστέρηση) ως βέλτιστο εντός του fog στρώματος. Το ακριβώς ανάποδο συμβαίνει για *w*=0.6, όπου το πρόβλημα αρχίζει να εστιάζει στην ελαχιστοποίηση του κόστους εξυπηρέτησης, και άρα ο πλέον κατάλληλος κόμβος εντός του fog στρώματος είναι ο κόμβος 3 (που φέρει το χαμηλότερο κόστος).

Χρόνος εκτέλεσης και πολυπλοκότητα

Η αλγοριθμική πολυπλοκότητα συνιστά από μόνη της ένα ξεχωριστό πεδίο της επιστήμης των υπολογιστών. Ο ακριβής ορισμός και το θεωρητικό πλαίσιο διαφεύγει από το αντικείμενο της εργασίας, οπότε παραπέμπουμε στα [47], [48] για μια εκτεταμένη ανάλυση του όρου. Σε μια πολύ αφηρημένη προσέγγιση, θα μπορούσαμε να πούμε πως η πολυπλοκότητα ενός αλγορίθμου εκφράζει τον αριθμό των βημάτων που χρειάζεται για την επίλυση του προβλήματος, και άρα τη χρονική του απόδοση, σε σχέση με το μέγεθος των εισόδων του.

Σε ένα σύνολο 100 εκτελέσεων του παραπάνω αλγορίθμου με τις εισόδους που περιγράψαμε παραπάνω (που μεταβάλλονται σε κάθε εκτέλεση εξαιτίας των τυχαία παραγόμενων τιμών), ο μέσος χρόνος εκτέλεσης για κάθε βάρος *w* από 0 έως 1 με βήμα 0.2 βρέθηκε $t_{avg} = [1.77\ 1.42\ 1.28\ 1.22\ 1.18\ 1.23]$ μετρημένος σε δευτερόλεπτα (sec). Παρατηρούμε ότι οι χρόνοι για τα πρώτα δύο βάρη είναι αυξημένοι, καθώς τα microservices των εφαρμογών ανταγωνίζονται για τις θέσεις στα κατώτερα στρώματα, και άρα παρουσιάζονται περισσότερες "συγκρούσεις" και περιορισμοί λόγω χωρητικότητας. Ακόμη, ο χρόνος εκτέλεσης είναι γενικά αυξημένος εξαιτίας της φύσης του αλγορίθμου, που εξετάζει κάθε πιθανό συνδυασμό για την τοποθέτηση, με δυνατότητα επανάληψης, όπου μόνο η διαδικασία αυτή παρουσιάζει πολυπλοκότητα της τάξεως $O(N^K)$, όπου K ο

μέγιστος δυνατός αριθμός microservices ανά εφαρμογή και *N* ο αριθμός των κόμβων. Μπορούμε να το σκεφτούμε ως εξής: Κάθε microservice *k* , μπορεί να επιλέξει μεταξύ *n* διαθέσιμων κόμβων, οπότε μια ακολουθία για παράδειγμα, τεσσάρων microservices, εγείρει $n * n * n * n = n^4$ συνδυασμούς. Συνεπώς η συνολική πολυπλοκότητα του αλγορίθμου ανέρχεται σε $O(M * (N\char`^K))$, όπου *M* ο αριθμός των εφαρμογών. Εύκολα συμπεραίνουμε ότι ο αριθμός των συνδυασμών αυξάνεται εκθετικά σε σχέση με τον αριθμό των microservices της εφαρμογής, καθιστώντας τον GRAA χρονικά ασύμφορο σε τέτοιες περιπτώσεις.

5.4 Rollout

5.4.1 Ευρετικοί και μετά-ευρετικοί αλγόριθμοι

Οι ευρετικοί αλγόριθμοι [49] είναι αλγόριθμοι σχεδιασμένοι ώστε να δώσουν μια λύση σε ένα πρόβλημα με γρηγορότερο και αποτελεσματικότερο τρόπο, ανταλλάσσοντας βελτιστοποίηση, ακρίβεια ή και πληρότητα για ταχύτητα. Συνήθως χρησιμοποιούνται όταν οι κατά-προσέγγιση λύσεις είναι επαρκείς για τους σκοπούς του ενδιαφερόμενου ενώ οι ακριβείς λύσεις κρίνονται υπολογιστικά ασύμφορες. Ενώ υπάρχουν ευρετικοί αλγόριθμοι “γενικού σκοπού” που μπορούν να εφαρμοσθούν σε πληθώρα προβλημάτων, κατά γενικό κανόνα οι ευρετικοί αλγόριθμοι λαμβάνουν υπόψη τα ιδιαίτερα χαρακτηριστικά του εκάστοτε προβλήματος ώστε να μεγιστοποιήσουν την αποτελεσματικότητά τους, και συνεπώς φθίνουν σε αποτελεσματικότητα σε διαφορετικής φύσεως προβλήματα.

Οι μετά-ευρετικοί αλγόριθμοι [50] είναι υψηλότερου επιπέδου τεχνικές για τη λύση δύσκολων προβλημάτων βελτιστοποίησης. Είναι περισσότερο μια στρατηγική για την ανεύρεση της λύσης, που ποικίλει από απλή τοπική αναζήτηση μέχρι πολύπλοκες μεθόδους μάθησης, ενώ συχνά οι μετά-ευρετικοί χρησιμοποιούν άλλους απλούστερους αλγορίθμους. Συνήθως οι μετά-ευρετικοί αλγόριθμοι έχουν γενικό χαρακτήρα και μπορούν να εφαρμοσθούν σε πληθώρα προβλημάτων.

5.4.2 Ο αλγόριθμος του Rollout

Ο αλγόριθμος του Rollout με διάφορες παραλλαγές σχεδιάστηκε από τους Bertsekas et.al. [51], [52] το 1997. Είναι μια μετά-ευρετική μέθοδος που στοχεύει στη βελτίωση των λύσεων γνωστών ευρετικών αλγορίθμων μέσω της διαδοχικής εφαρμογής τους. Ο Rollout αποκτά ιδιαίτερη χρησιμότητα όταν οι ακριβείς μέθοδοι είναι πολύ αργές ή/και όταν οι λύσεις που παρέχονται από ευρετικούς αλγορίθμους κρίνονται ανεπαρκείς. Αν και χαίρει ιδιαίτερης απλότητας στην κατανόηση και την κατασκευή του, είναι συχνά εκπληκτικά αποδοτικός.

Η δομή (framework) που χρησιμοποιείται για τον αλγόριθμο χαρακτηρίζεται από ένα πεπερασμένο σύνολο $\boldsymbol{O}$ εφικτών λύσεων και μια συνάρτηση κόστους g(o). Κάθε λύση απαρτίζεται από *K* “συστατικά” της μορφής $o=(o_1,o_2,\ldots,o_K)$, που καλείται το “μονοπάτι της λύσης”. Σκοπός είναι η εύρεση της λύσης $\boldsymbol{o} \in \boldsymbol{O}$ η οποία ελαχιστοποιεί τη συνάρτηση κόστους *g(o)*.

Μπορούμε να θεωρήσουμε το παραπάνω πρόβλημα σαν πρόβλημα δυναμικού προγραμματισμού (DP- Dynamic Programing) , όπου κάνουμε διαδοχικές επιλογές

για τα συστατικά της λύσης, δηλαδή επιλέγουμε κάθε φορά ένα από τα (o_1,…,o_K). Μια λύση που εμπεριέχει τα πρώτα *k* συστατικά (o_1,…,o_k) είναι μια μερική λύση που καλείται k-λύση, ενώ μια λύση που εμπεριέχει όλα τα *K* συστατικά είναι μια ολική λύση που καλείται K-λύση.

Έστω $J^*(o_1, \ldots, o_k)$ το βέλτιστο κόστος εκκινώντας από μια *k*-λύση. Η ανεύρεση αυτού του βέλτιστου είναι ασύμφορη, αφού και πάλι οι ιδιότητες του προβλήματος διατηρούνται και άρα απαιτείται μεγάλη υπολογιστική προσπάθεια. Αυτό αντιμετωπίζεται αντικαθιστώντας το $J^*(o_1, \ldots, o_k)$ με μια προσέγγισή του $\tilde{J}(o_1, \ldots, o_k)$ και ανακτώντας μια υπό-βέλτιστη λύση $(\tilde{o}_1, \ldots, \tilde{o}_K)$ σύμφωνα με την εξίσωση:

$$\tilde{o}_i = arg \min_{o_i \in O} \tilde{J}(\tilde{o}_1, \ldots, \tilde{o}_{i-1}, o_i), \qquad i = 1, .., K$$

Η συνάρτηση που υπολογίζει την προσέγγιση $\tilde{J}$ καλείται "συνάρτηση βαθμολόγησης". (scoring function) Μια τέτοια συνάρτηση συνήθως είναι ένας ευρετικός αλγόριθμος *H* ο οποίος δοθέντος μιας *k*-λύσης κατασκευάζει μια *K*-λύση. Ο rollout στην τυπική μορφή του για την εισαγωγή ενός συστατικού στην τελική λύση συγκρίνει τις προσεγγίσεις o_i όλων των πιθανών συστατικών που επιστρέφει ο ευρετικός αλγόριθμος και επιλέγει αυτό με την καλύτερη προσέγγιση κάθε φορά. Παρακάτω παρουσιάζεται η εκτέλεση του αλγορίθμου σε ψευδοκώδικα:

Αλγόριθμος 5.6 Rollout

1. Input: A *k*-solution $\boldsymbol{o}_k = (o_1, \ldots, o_k)$ and a Heuristic H
2. Output: A complete *K*-solution
3. **Repeat**
4. For all i: $o_i \in O \; and \; o_i \notin \boldsymbol{o}_k \; do$
5. $\tilde{J}_i = H(\boldsymbol{o}_k, o_i)$
6. EndFor
7. $i^* \leftarrow arg \min_i \tilde{J}_i$
8. $Add \; o_{i^*} \; ath \; the \; end \; of \; k - solution \; \boldsymbol{o}_k$

5.4.3 Επεξήγηση του αλγορίθμου και ιδιότητες

Για μια επίσημη διατύπωση του αλγορίθμου Rollout θα παρουσιάσουμε ένα πρόβλημα αναζήτησης σε γράφο που είναι μια γενική μορφή ενός διακριτού ντετερμινιστικού προβλήματος βελτιστοποίησης. Για την επεξήγηση των ειδικών ορολογιών και ορισμών της θεωρίας των γράφων – που διαφεύγει από τα πλαίσια της παρούσας εργασίας- παραπέμπουμε στα [53], [54]

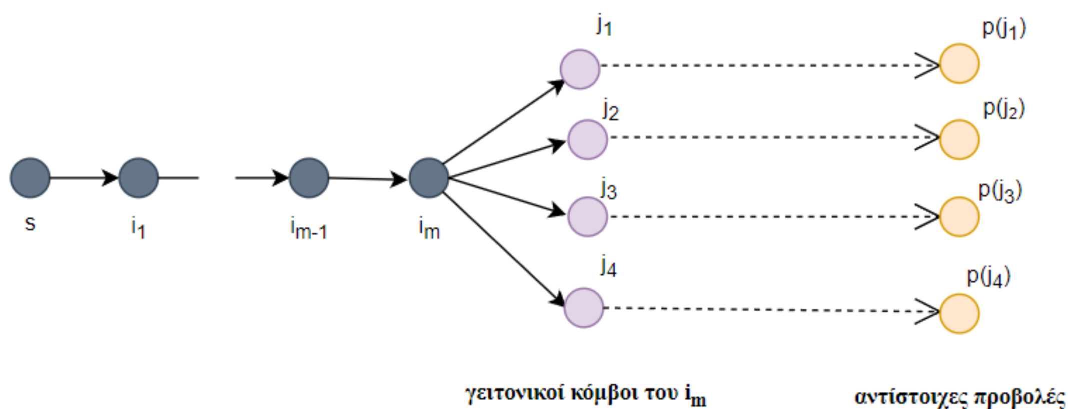
Δίνεται ένας γράφος με ένα πεπερασμένο σύνολο κόμβων *N*, ένα πεπερασμένο σύνολο προσανατολισμένων ακμών *A* και έναν ειδικό "κόμβο αρχής" *s*. Δίνεται επίσης ένα $\overline{N}$ υποσύνολο των κόμβων *N* που καλείται "κόμβοι προορισμού" και είναι "τελικοί" κόμβοι με την έννοια ότι δεν έχουν εξερχόμενες ακμές. Κάθε κόμβος προορισμού *i* φέρει επιπλέον ένα κόστος *g(i).* Το πρόβλημα έγκειται στην εύρεση ενός μονοπατιού $(s, i_1, \ldots, i)$ στο γράφο που εκκινεί από τον κόμβο αρχής *s* και

καταλήγει σε έναν κόμβο προορισμού $i \in \bar{N}$, έτσι ώστε να ελαχιστοποιείται το κόστος *g(i)*.

Υποθέτουμε ακόμη τη διαθεσιμότητα ενός ευρετικού αλγορίθμου *H*, ο οποίος δοθέντος ενός κόμβου i εκτός των κόμβων προορισμού $i \notin \bar{N}$, κατασκευάζει το μονοπάτι από τον *i* μέχρι έναν κόμβο προορισμού $\bar{\imath}$ $(i, i_1, \dots, i_m, \bar{\imath})$. Ο *H* καλείται "βασικός ευρετικός" αλγόριθμος, και αποτελεί τη δομική μονάδα για την κατασκευή του Rollout αλγορίθμου που θα δούμε παρακάτω.

Ο κόμβος προορισμού $\bar{\imath}$ του μονοπατιού που επιστρέφει ο *H* είναι αποκλειστικά προσδιορισμένος από τον εναρκτήριο κόμβο *i*. Ο κόμβος $\bar{\imath}$ καλείται η "προβολή του i " από τον *H* και συμβολίζεται με p(i), ενώ το αντίστοιχο κόστος που προκύπτει από τον *H* είναι $H(i) = g(p(i))$.

Ο Rollout επιχειρεί να κατασκευάσει σταδιακά μια λύση με διαδοχική χρήση του *H*. Σε ένα τυπικό βήμα της διαδικασίας, υπάρχει ένα μονοπάτι από τον κόμβο αρχής *s* μέχρι έναν κόμβο *i*, και ο αλγόριθμος *H* καλείται για κάθε κατάντη γειτονικό κόμβο *j* του *i*, επιστρέφοντας τις αντίστοιχες προβολές H(j). Ο γειτονικός κόμβος που δίνει την καλύτερη προβολή προστίθεται στο μονοπάτι, μέχρις ότου φτάσουμε σε έναν κόμβο προορισμού. Η παραπάνω διαδικασία συνιστά τον Rollout αλγόριθμο βασισμένο στον *H*, και συμβολίζεται με *RH*.



Εικόνα 5-15 Τυπικό βήμα του Rollout. Μετά από m βήματα, ο αλγόριθμος έχει κατασκευάσει το μονοπάτι $(s, i_1, \dots, i_m)$. Για την επέκταση του μονοπατιού, θεωρείται το σύνολο των γειτόνων του τελικού κόμβου i_m, $N(i_m)$, και επιλέγεται ο κόμβος που δίνει την καλύτερη προβολή.

Για μια πιο τυπική παρουσίαση, έστω *N(i)* το σύνολο των κατάντων γειτονικών κόμβων ενός μη τερματικού κόμβου *i*:

$N(i) = \{ j \mid (i, j)$ είναι εξερχόμενη ακμή από τον *i* $\}$

Ο αλγόριθμος RH εκκινεί με τον κόμβο αρχής *s*. Στο τυπικό του βήμα, δοθέντος ενός μονοπατιού $(s, i_1, .., i_m)$ όπου ο i_m δεν είναι κόμβος προορισμός, ο RH προσθέτει στο μονοπάτι έναν κόμβο i_{m+1} τέτοιο ώστε:

$$i_{m+1} \in arg \min_{j \in N(i_m)} H(j)$$

Εάν ο i_{m+1} είναι κόμβος προορισμού ο αλγόριθμος τερματίζει επιστρέφοντας το συνολικό μονοπάτι της λύσης, ειδάλλως συνεχίζει με την ίδια διαδικασία με δεδομένο το μονοπάτι $(s, i_1, \ldots, i_m, i_{m+1})$.

Όταν ο *RH* τερματίσει επιστρέφοντας ένα μονοπάτι $(s, i_1, .., i_m)$, οι προβολές $p(i_k)$ από κάθε έναν εκ των κόμβων $i_k, k = 1, \ldots, m$ θα έχουν υπολογιστεί. Η καλύτερη εξ αυτών των προβολών δημιουργεί κόστος:

$$\min_{k=1,..,m} H(i_k) = \min_{k=1,\ldots,m} g(p(i_k))$$

Η προβολή που αντιστοιχεί στο ανωτέρω ελάχιστο λαμβάνεται ως η τελική λύση που παράγεται από τον *RH*. Τέλος, η λύση αυτή μπορεί να συγκριθεί και με το κόστος *g(p(s))* από την προβολή του κόμβου αρχής *s p(s)*, ώστε να εξασφαλιστεί ότι ο RH θα επιστρέψει λύση που είναι μη-χειρότερη από αυτή που επιστρέφει ο βασικός ευρετικός H. Μια λογική απορία είναι η εξής: Γιατί ο *RH* συγκρίνει τις προβολές όλων των κόμβων για το τελικό αποτέλεσμα, αντί να δώσει κατευθείαν το κόστος από τον τελικό κόμβο προορισμού σύμφωνα με το μονοπάτι που δημιούργησε; Η απάντηση δίνεται από το γεγονός πως μπορεί ο βέλτιστος γειτονικός κόμβος ενός κόμβου i_m να είναι ο i_{m+1}, αλλά αυτό δε σημαίνει πως ο γείτονας αυτός έχει καλύτερη προβολή από τον ίδιο, δηλαδή:

$$H(i_{m+1}) \leq H(i_m)$$

Ο λόγος που η παραπάνω συνθήκη δεν εξασφαλίζεται τουλάχιστον σαν ισότητα, είναι επειδή εάν το μονοπάτι που παράγει ο *H* εκκινώντας από τον i_m είναι το $(i_m, j_1, \ldots, j_k, \tilde{\iota})$, δεν υπάρχει καμία εγγύηση ότι το μονοπάτι που θα παράξει ο H εκκινώντας από τον j_1 είναι το $(j_1, \ldots, j_k, \tilde{\iota})$. Δηλαδή, αν ο H εκκινήσει από επόμενο γειτονικό κόμβο, μπορεί να επιστρέψει μια εντελώς διαφορετική προβολή με διαφορετικό κόστος. Παρακάτω θα εξετάσουμε ενδλεχέστερα αυτό το ζήτημα και θα δούμε πότε ο H μπορεί να εξασφαλίζει την παραπάνω συνθήκη και άρα η διαδικασία να απλουστευτεί, ενώ ταυτόχρονα να εξασφαλιστεί και η περατότητα του RH.

Ορισμός 1: Ο βασικός ευρετικός αλγόριθμος *H* ονομάζεται διαδοχικά συνεπής αν εκκινώντας από τον κόμβο *i* κατασκευάζει το μονοπάτι $(i, i_1, \ldots, i_m, \bar{\iota})$ ενώ εκκινώντας από τον κόμβο i_1, κατασκευάζει το μονοπάτι $(i_1, \ldots, i_m, \bar{\iota})$.

Συνεπώς, ένας διαδοχικά συνεπής αλγόριθμος επιστρέφει την ίδια προβολή για όλους τους κόμβους ενός μονοπατιού που κατασκευάζει.

Πρόταση 1: Έστω πως ο βασικός ευρετικός *H* είναι διαδοχικά συνεπής. Τότε, ο *RH* φέρει την ιδιότητα της περατότητας, δηλαδή εγγυημένα τερματίζει εκκινώντας από οποιονδήποτε κόμβο. Επιπλέον, αν $(i_1, \ldots, i_{\bar{m}})$ είναι το μονοπάτι που κατασκευάζεται από τον RH εκκινώντας από έναν μη τερματικό κόμβο i_1 και καταλήγωντας σε έναν τερματικό κόμβο i_m, θα ισχύει:

$$H(i_1) \geq H(i_2) \geq \cdots \geq H(i_{\bar{m}-1}) \geq H(i_{\bar{m}})$$

Επιπλέον, για κάθε $m = 1, \ldots, \bar{m}$

$$H(i_m) = \min\{ H(i_1), \min_{j \in N(i_1)} H(j), \ldots, \min_{j \in N(i_{m-1})} H(j)\}$$

Η πρόταση 1 αναδεικνύει την ιδιότητα της "αυτόματης ταξινόμησης κόστους" του *RH* στην περίπτωση όπου ο *H* είναι διαδοχικά συνεπής, αφού ο *RH* εμμέσως ταξινομεί τα διάφορα κόστη και επιλέγει το καλύτερο μονοπάτι που κατασκευάζεται από τον *H*. Συγκεκριμένα, όταν ο *RH* κατασκευάζει ένα μονοπάτι $(i_1, \ldots, i_m)$, χρησιμοποιεί τον *H* για τη δημιουργία μιας συλλογής μονοπατιών και αντίστοιχων προβολών εκκινώντας από όλους τους επόμενους κόμβους των ενδιάμεσων κόμβων $i_1, \ldots, i_{m-1}$ και επιλέγει τα βέλτιστα κάθε φορά. Έτσι, εγγυάται ότι το μονοπάτι $(i_1, \ldots, i_m)$ είναι το καλύτερο της συλλογής και ο i_m έχει το ελάχιστο κόστος από όλες τις δημιουργηθείσες προβολές. Το γεγονός αυτό ωστόσο δεν είναι αρκετό ώστε να εγγυηθεί ότι το μονοπάτι του *RH* είναι σχεδόν-βέλτιστο, αφού η συλλογή μονοπατιών που παράγεται από τον *H* μπορεί να είναι φτωχή από άποψη βελτιστοποίησης.

Ορισμός 2: Ο βασικός ευρετικός H θεωρείται ότι βελτιώνεται διαδοχικά εάν για κάθε μη τελικό κόμβο i ισχύει:

$$H(i) \geq \min_{j \in N(i)} H(j)$$

Εύκολα εξάγουμε το συμπέρασμα πως ένας διαδοχικά συνεπής H επίσης βελτιώνεται διαδοχικά, λόγω της ισότητας στην παραπάνω ανισότητα. Έτσι η πρόταση 1 μπορεί να επαναδιατυπωθεί σε μια γενικότερη μορφή:

Πρόταση 2: Έστω πως ο βασικός ευρετικός H βελτιώνεται διαδοχικά και ο RH τερματίζει. Έστω επίσης το μονοπάτι $(i_1, \ldots, i_{\bar{m}})$ που κατασκευάζει ο RH από έναν μη τερματικό κόμβο i_1 μέχρι έναν κόμβο προορισμού $i_{\bar{m}}$. Τότε το κόστος του RH εκκινώντας από τον i_1 είναι μικρότερο ή ίσο από το κόστος του H εκκινώντας από τον i_1. Συγκεκριμένα, για κάθε $m = 1, \ldots, \bar{m}$ ισχύει:

$$H(i_m) = \min\{ H(i_1), \min_{j \in N(i_1)} H(j), \ldots, \min_{j \in N(i_{m-1})} H(j)\}$$

Συμπεραίνουμε λοιπόν πως ο *RH* εκμεταλλεύεται κατάλληλα τον *H* δημιουργώντας διαδοχικά ένα μονοπάτι με κόστος στη χειρότερη περίπτωση ίσο με αυτό του *H*. Στην πράξη η βελτίωση του κόστους μπορεί να είναι ιδιαίτερα δραστική.

Ο GRAA κρίνεται ανεπαρκής στην (ρεαλιστική) περίπτωση όπου οι εφαρμογές φέρουν μεγαλύτερο αριθμό microservices, καθώς γιγαντώνονται οι χρόνοι εκτέλεσής του, ενώ ακόμη η greedy προσέγγιση που χρησιμοποιεί δεν εγγυάται απαραίτητα μια συνολικά ικανοποιητική (πλησία της βέλτιστης) λύση. Είναι λοιπόν σκόπιμο να κατασκευάσουμε έναν απλούστερο βασικό ευρετικό αλγόριθμο *H*, που θα προσφέρει ταχύτερους χρόνους εκτέλεσης χωρίς ενδεχομένως να στερείται αποτελεσματικότητας. Η ταχύτητα εκτέλεσης είναι , όπως είδαμε, κρίσιμης σημασίας για την υιοθέτηση ενός αλγορίθμου από το Rollout, καθώς στη διαδικασία ανεύρεσης του "μονοπατιού" της λύσης ο βασικός ευρετικός καλείται εκατοντάδες ή και δεκάδες χιλιάδες φορές ανάλογα με το μέγεθος του προβλήματος.

5.5 Ο βασικός ευρετικός Η

Κεντρική Ιδέα

Ο αλγόριθμος αποσκοπεί στη λύση του προβλήματος που παρουσιάστηκε στο παρόν κεφάλαιο, έτσι η δομή του προβλήματος διατηρείται αυτούσια (η τοπολογία και τα στοιχεία της, η λογική δομή των εφαρμογών κλπ.) και άρα ο αλγόριθμος δέχεται τις ίδιες εισόδους με τον GRAA. Η διαδικασία που ακολουθεί για την εύρεση της υπό-βέλτιστης λύσης είναι η εξής: Οι εφαρμογές σχηματίζουν και πάλι ουρά FIFO, ενώ είναι τοποθετημένες σε αύξουσα σειρά σύμφωνα με το άνω όριο καθυστέρησης (για την αποφυγή φαινομένων blocking, όπως εξηγήθηκε στον GRAA). Αφού επιλέξει την εφαρμογή που βρίσκεται πρώτη στην ουρά, βρίσκει τους ικανούς κόμβους (που ικανοποιούν το άνω όριο καθυστέρησης της εφαρμογής). Στη συνέχεια επιλέγει το βέλτιστο κόμβο εξ αυτών (που έχει το μικρότερο κόστος σύμφωνα με το δεδομένο βάρος και έχει επαρκή χωρητικότητα) και τοποθετεί το πρώτο microservice. Έπειτα υπολογίζει τους ικανούς κόμβους που ικανοποιούν το άνω όριο σχετικής καθυστέρησης σε σχέση με το βέλτιστο κόμβο που βρέθηκε προηγουμένως, και τοποθετεί το 2ο microservice στο βέλτιστο εξ αυτών. Με την ίδια λογική, για την τοποθέτηση του 3ου microservice υπολογίζει τους ικανούς κόμβους που ικανοποιούν το άνω όριο σχετικής καθυστέρησης με καθέναν εκ των κόμβων όπου τοποθέτησε τα προηγούμενα 2 microservices, και επιλέγει το βέλτιστο. Η διαδικασία επαναλαμβάνεται για όλα τα microservices της εφαρμογής. Πρόκειται δηλαδή πάλι για έναν greedy αλγόριθμο, αλλά αυτή τη φορά δε βρίσκει τον καλύτερο συνδυασμό κόμβων για την τοποθέτηση ολόκληρης της εφαρμογής, αλλά τον καλύτερο κόμβο για την τοποθέτηση κάθε microservice διαδοχικά. Στην απλή αυτή λογική ελλοχεύει μια παγίδα: Όταν κάθε microservice τοποθετείται με τη σειρά στο βέλτιστο κόμβο, ενδέχεται για κάποιο microservice να μην υπάρχουν άλλοι κόμβοι που να ικανοποιούν τα όρια καθυστέρησης ή τους περιορισμούς χωρητικότητας, οδηγώντας στο μπλοκάρισμα της εφαρμογής! Για την αποφυγή του παραπάνω φαινομένου, εάν για κάποιο microservice δεν βρεθεί κόμβος τοποθέτησης, ο αλγόριθμος επαναλαμβάνει τη διαδικασία, αυτή τη φορά τοποθετώντας το 1ο microservice στον 2ο κατά σειρά προτίμηση κόμβο, στη συνέχεια στον 3ο και ούτω καθεξής. Σημειώνουμε πως για την πλήρη αποφυγή του μπλοκαρίσματος θα έπρεπε να εξερευνηθούν και άλλοι συνδυασμοί, (π.χ. άλλα microservices εκτός του πρώτου να επέλεγαν κόμβο χαμηλότερης προτίμησης), ωστόσο πειραματικά απεδείχθη ότι η τροποποίηση αυτή επαρκεί για τα δεδομένα του προβλήματος.

Βήμα 1 Εύρεση βέλτιστου κόμβου για το πρώτο microservice

Αρχικά ο αλγόριθμος βρίσκει όλους τους κόμβους που ικανοποιούν το άνω όριο καθυστέρησης της εφαρμογής. Στη συνέχεια, απορρίπτει αυτούς που δεν έχουν τους απαραίτητους εναπομείναντες πόρους για την εξυπηρέτηση. Από το τελικό σύνολο "ικανών" κόμβων, βρίσκει το βεβαρυμμένο άθροισμα καθυστέρησης – κόστους για κάθε έναν από αυτούς, και τους τοποθετεί σε αύξουσα σειρά. Ο πρώτος εξ αυτών είναι ο βέλτιστος για την τοποθέτηση του πρώτου microservice. Αν η εφαρμογή αποτελείται από μόνο ένα microservice, η διαδικασία ολοκληρώνεται. Αν κανένας κόμβος δεν έχει την απαραίτητη χωρητικότητα, εμφανίζεται μήνυμα για το μπλοκάρισμα της εφαρμογής. Η διαδικασία παρουσιάζεται παρακάτω σε ψευδογλώσσα:

Αλγόριθμος 5.7 Εύρεση του βέλτιστου κόμβου για το πρώτο microservice

1. Inputs (nodes latency vector *l* , nodes cost vector *cost*, latency limit lm_j, weight *w*, cpu demand mc_{j1}, ram demand mr_{j1}, nodes remaining cpu *rcpu*, nodes remaining RAM *rram*)
2. Outputs (*best node* , *order_of_preference*)
3. For every node *i*
4. If $l_i \leq lm_j$
5. If ($rcpu_i \geq mc_{j1}$ && $rram_i \geq mr_{j1}$)
6. add i to competent_nodes *cn*
7. calculate $x_i = w * cost_i + (1-w) * l_i$
8. EndIf
9. EndIf
10. EndFor
11. If competent_nodes is empty
12. display "Application *j* was blocked"
13. else
14. *order_of_preference*=sort(x)
15. *best_node*= id of *(order_of_preference(1))*
16. EndIf

Στη συνέχεια ο αλγόριθμος ανανεώνει τους πόρους του βέλτιστου κόμβου μετά την τοποθέτηση του πρώτου microservice, προσαυξάνει την τιμή της αντικειμενικής συνάρτησης κατά το βεβαρυμμένο άθροισμα του κόμβου αυτού και συνεχίζει στο παρακάτω βήμα.

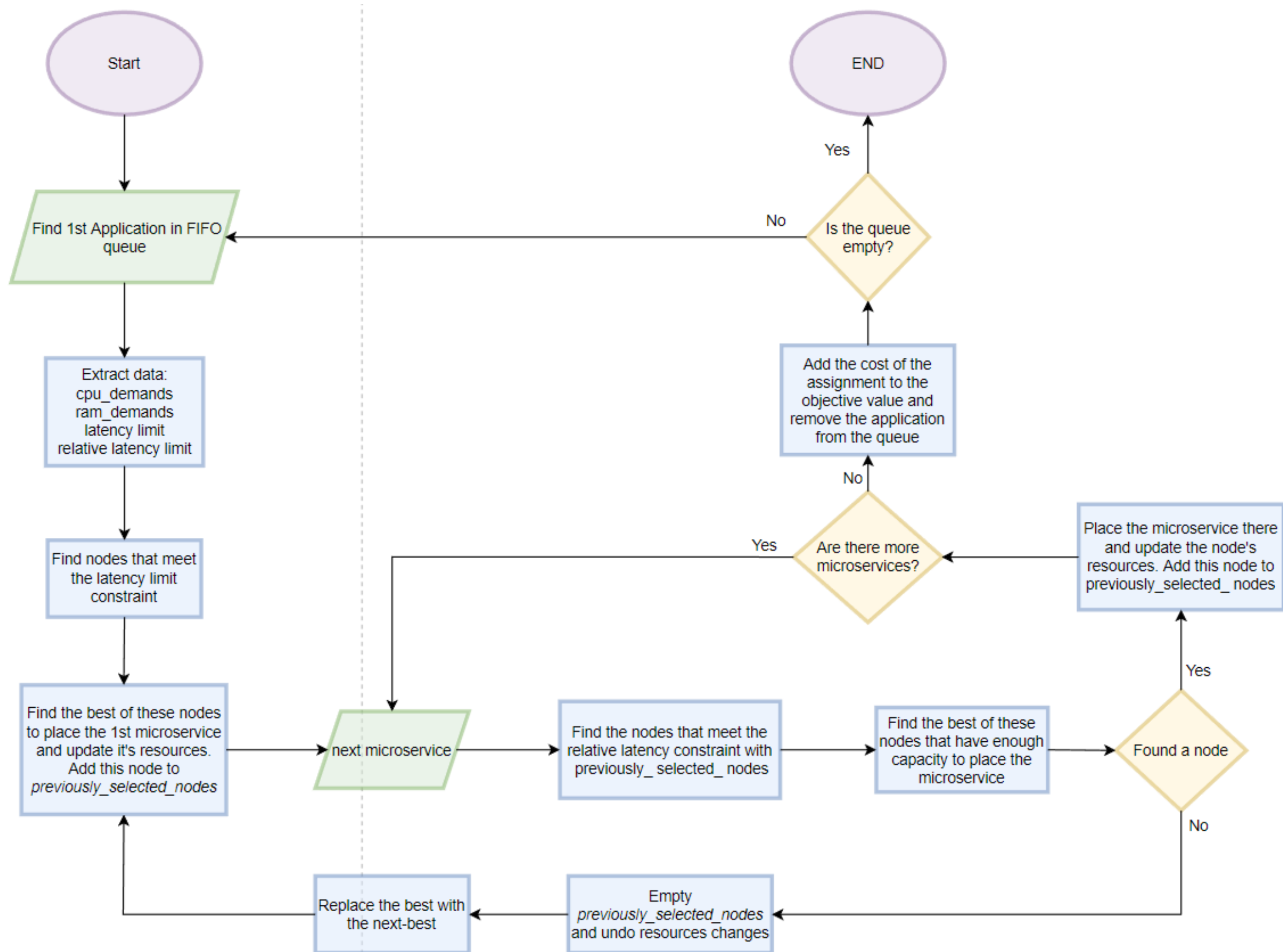
Βήμα 2 Τοποθέτηση των υπόλοιπων microservices

Για κάθε ένα από τα υπόλοιπα microservices, υπολογίζεται ο βέλτιστος ικανός κόμβος που ικανοποιεί το κριτήριο σχετικής καθυστέρησης με κάθε έναν από τους κόμβους που έχουν ήδη επιλεγεί και έχει την απαραίτητη χωρητικότητα. Αν δε βρεθεί κανένας τέτοιος κόμβος, ο αλγόριθμος επιστρέφει στο προηγούμενο βήμα και επιλέγει τον 2ο κατά προτίμηση κόμβο για την τοποθέτηση του 1ου microservice. Αν τελικά με καμία τοποθέτηση του πρώτου κόμβου δεν ευρίσκονται ικανοί κόμβοι για την τοποθέτηση έστω ενός εκ των υπόλοιπων microservices, εμφανίζεται μήνυμα για μπλοκάρισμα της εφαρμογής. Η διαδικασία φαίνεται παρακάτω:

Αλγόριθμος 5.8 Τοποθέτηση των υπόλοιπων microservices

1. Inputs (*best_node*, nodes latency vector *l* , nodes cost vector *cost*, relative latency limit rlm_j weight *w*, cpu demands mc_{jk}, ram demands mr_{jk}, nodes remaining cpu *rcpu*, nodes remaining RAM *rram*, relative latency table *rl*, competent nodes *cn*, number of microservices K_j)
2. *best*_nodes=best_*node*
3. For *k*=2 to K_j
4. For every *i* in *cn*
5.   *violation_flag=0;*
6.   For every *b* in best_nodes
7.   If $rl\ (i, b) > rlm_j$
8.   *violation_flag*=1;
9.   EndIf
10.   EndFor
11.   If (*violation_flag* == 0)
12.   add i to *new_competent*
13.   EndIf
14. EndFor
15.   For every *i* in *new_competent*
16.   If ($rcpu_i \geq mc_{jk}$ && $rram_i \geq mr_{jk}$)
17.   calculate $y_i = w * cost_i + (1 - w) * l_i$
18.   EndIf
19.   EndFor
20. EndFor
21. If y is not empty
22.   Find minimum (*y*)
23.   %Update resources of the node attending the above minimum
24. $rcpu_{id} = rcpu_{id} - mc_{jk}$
25. $rram_{id} = rram_{id} - mr_{jk}$
26.   Add *id* to *best_nodes*
27.   Add minimum(*y*) to the objective value
28. else
29.   try the next node in *order_of_preference* as *best_node* and re-do this step
30. EndIf

Η παραπάνω διαδικασία επαναλαμβάνεται για όλες τις εφαρμογές, υπολογίζοντας την τελική τοποθέτηση στους κόμβους και την τελική τιμή της αντικειμενικής συνάρτησης. Ο συνολικός αλγόριθμος παρουσιάζεται στο παρακάτω διάγραμμα ροής:



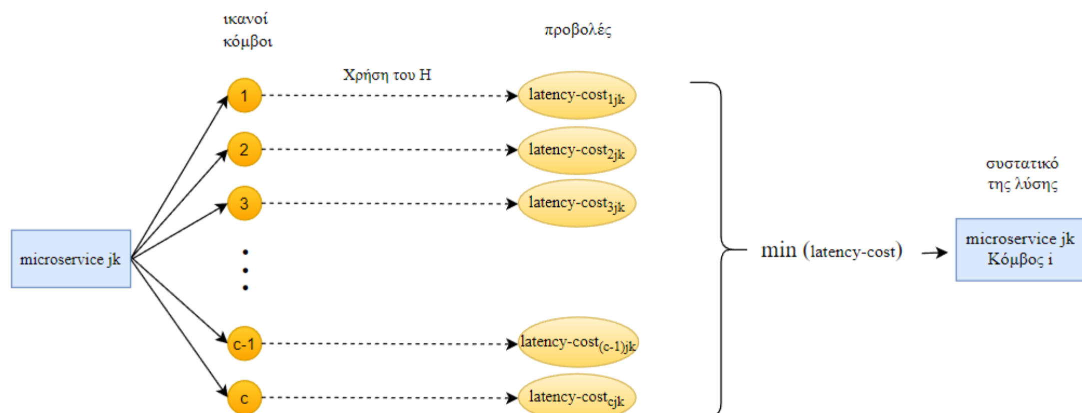
Εικόνα 5-16 Διάγραμμα ροής για τον ευρετικό αλγόριθμο H

5.6 Rollout με χρήση του *H*

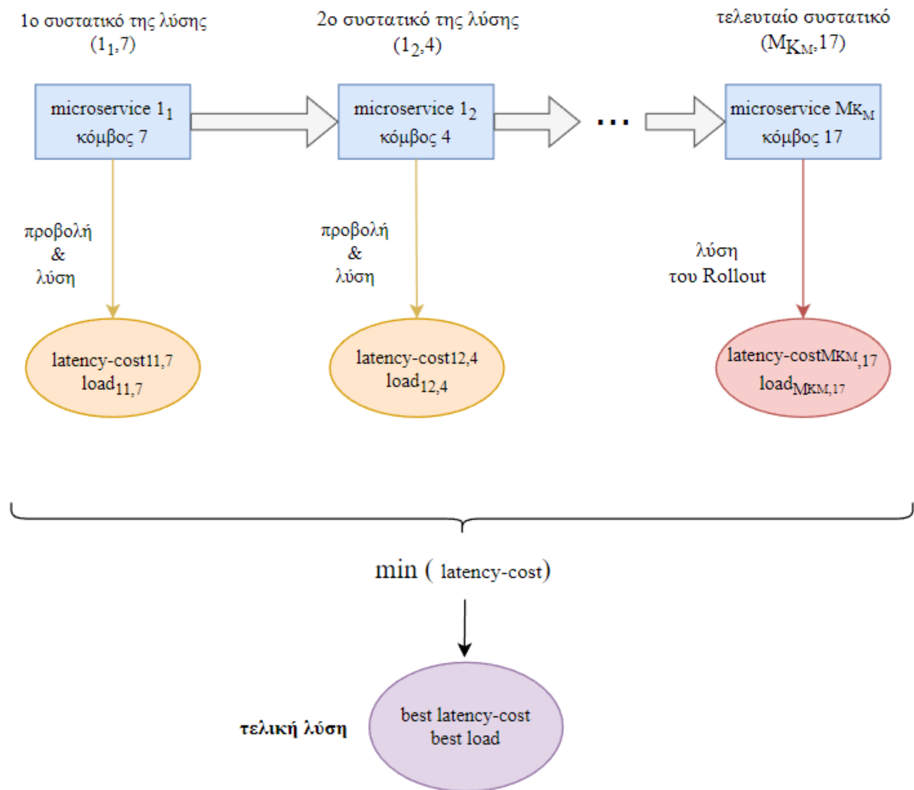
Θα επιστρατεύσουμε την τεχνική του Rollout χρησιμοποιώντας ως βασικό ευρετικό αλγόριθμο τον αλγόριθμο *H* που παρουσιάστηκε παραπάνω. Εκκινώντας από τον κόμβο αρχής *s* (εδώ το *s* συμβολίζει τον κόμβο αρχής στη διαδικασία εύρεσης μονοπατιού του Rollout, και όχι τον κόμβο παραγωγής δεδομένων της τοπολογίας), επιχειρούμε να κατασκευάσουμε το μονοπάτι της λύσης $(s, m_{11,} \dots m_{1K_1}, \dots, m_{M1}, \dots, m_{MK_M})$. Κάθε νέο συστατικό m_{jk} είναι η τοποθέτηση του *k*-οστού microservice της εφαρμογής *j* σε κάποιο κόμβο. Συγκεκριμένα , στην αρχή της διαδικασίας το πρώτο microservice της πρώτης εφαρμογής θα τοποθετηθεί στον πρώτο, κατά σειρά εμφάνισης, κόμβο. Τα υπόλοιπα microservices της εφαρμογής καθώς και όλες οι υπόλοιπες εφαρμογές θα τοποθετηθούν με χρήση του *H*. Η αντιστοίχιση αυτή θα επιφέρει ένα αποτέλεσμα (τιμή της αντικειμενικής συνάρτησης) *latency-cost*$_{111}$, που είναι το κόστος της προβολής του πρώτου υποψήφιου συστατικού. Στη συνέχεια θα επαναλάβουμε τη διαδικασία, αυτή τη φορά τοποθετώντας το πρώτο microservice της εφαρμογής στον δεύτερο κατά σειρά κόμβο, και λαμβάνοντας ένα αποτέλεσμα *latency-cost*$_{211}$, και ούτω καθεξής. Η τελική τοποθέτηση του πρώτου microservice της πρώτης εφαρμογής θα είναι στον

κόμβο που επέφερε το μικρότερο αποτέλεσμα (φυσικά δοκιμάζονται μόνο οι επιτρεπόμενοι κόμβοι, ώστε να έχουμε εφικτές λύσεις), δηλαδή στον κόμβο $i = arg \min_{i \in comp} latency - cost_{i11}$. Ακολουθώντας την ίδια διαδικασία για όλα τα αιτήματα , λαμβάνουμε την τελική τοποθέτηση στους κόμβους , δηλαδή το διάνυσμα *rload* από το Rollout, καθώς και την τελική τιμή της αντικειμενικής συνάρτησης *rlatency-cost*.

Φαίνεται ωστόσο να αγνοούμε μια σημαντική λεπτομέρεια: Δε γνωρίζουμε αν ο ευρετικός αλγόριθμος *H* είναι διαδοχικά συνεπής, ούτε αν βελτιώνεται διαδοχικά. Επομένως, για την εξαγωγή του βέλτιστου αποτελέσματος, δεν θα πάρουμε απευθείας τα τελικά στοιχεία *rload*, *rlatency-cost*, αλλά θα συγκρίνουμε σε κάθε βήμα του Rollout (δηλαδή σε κάθε τοποθέτηση microservice στο βέλτιστο κόμβο) τα αποτελέσματα που προκύπτουν (το κόστος των αντίστοιχων προβολών) και θα επιλέξουμε εν τέλη το βέλτιστο από αυτά. Δηλαδή , κάθε φορά που ένα microservice *jk* τοποθετείται στον βέλτιστο κόμβο $i = arg \min_{j \in comp} latency - cost_{ijk}$, θα συγκρατούμε τόσο το κόστος της προβολής $\mathrm{latency - cost}_{ijk}$, δηλαδή την αντίστοιχη τιμή της αντικειμενικής συνάρτησης, όσο και το παρελκόμενα load_{ijk} , και εν τέλει θα επιλέξουμε το βέλτιστο εξ αυτών. Η συνολική διαδικασία παρουσιάζεται στα παρακάτω διαγράμματα:



Εικόνα 5-17 Ανεύρεση συστατικού της λύσης. Το k-οστό microservice της εφαρμογής j τοποθετείται με τη σειρά σε καθέναν από τους ικανούς κόμβους και υπολογίζονται οι αντίστοιχες προβολές. Το συστατικό της λύσης θα αποτελέσει η τοποθέτηση που επιστρέφει την καλύτερη προβολή.



Εικόνα 5-18 Παράδειγμα εξαγωγής της τελικής λύσης. Υπολογίζονται οι προβολές και οι λύσεις όλων των συστατικών, καθώς και η λύση του συνολικού μονοπατιού, και ως τελική λύση επιλέγεται η βέλτιστη εξ αυτών.

5.7 Αποτελέσματα και αξιολόγηση

Για την διεξαγωγή των πειραμάτων θα χρησιμοποιήσουμε τα δεδομένα που χρησιμοποιήθηκαν για τον GRAA, ωστόσο θα αυξήσουμε τον αριθμό των κόμβων του edge , προκειμένου να αυξήσουμε τον μέγιστο αριθμό microservices/ εφαρμογή και να αποφύγουμε φαινόμενα blocking που δεν εξυπηρετούν τη διαδικασία. Τα δεδομένα φαίνονται παρακάτω:

	Αριθμός κόμβων	Αριθμός cpu units/κόμβο	Αριθμός ram units/κόμβο	Καθυστέρηση latency units	Κόστος Cost units
Στρώμα edge	15	15	4-8	0.5-2	6-9
Στρώμα fog	3	100	64	3.5-4.2	3.3-4.3
Στρώμα cloud	2	500	256	7.5-8.5	1-1.5

Πίνακας 5-5 Στοιχεία της πειραματικής υποδομής

Κόμβοι στρωμάτων	Σχετική Καθυστέρηση
Edge-Edge	0-1
Edge-Fog	3
Edge-Cloud	7
Fog-Fog	0-2
Fog-Cloud	3
Cloud-Cloud	0-3

Πίνακας 5-6 Σχετικές καθυστερήσεις

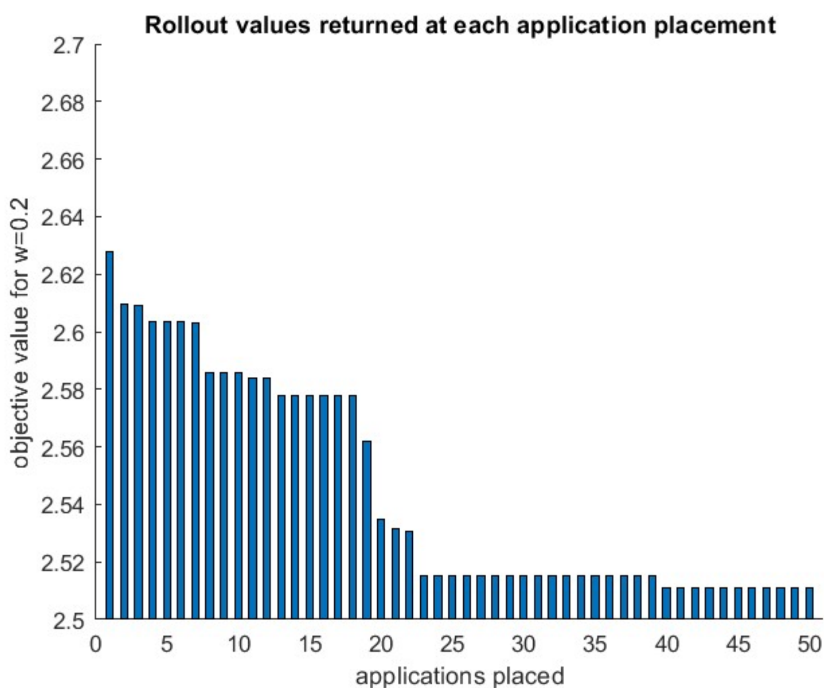
	# microservices	cpu demand/ms	ram demand/ms	latency limit	relative latency limit
Applications (1-50)	1-5	1-5	1-2	1.5-10	0.5-6

Πίνακας 5-7 Στοιχεία πειραματικών εφαρμογών

Αποτελέσματα και αποτίμηση

Έλεγχος διαδοχικής βελτίωσης του *H*

Αρχικά, μπορούμε να εξετάσουμε τη συμπεριφορά των τιμών που επιστρέφονται από το Rollout κάθε φορά που τοποθετείται μια εφαρμογή. Μια ενδεικτική εκτέλεση φαίνεται παρακάτω, για βάρος *w*=0.2:



Εικόνα 5-19 Τιμές από το Rollout για κάθε τοποθέτηση εφαρμογής, για βάρος w=0.2

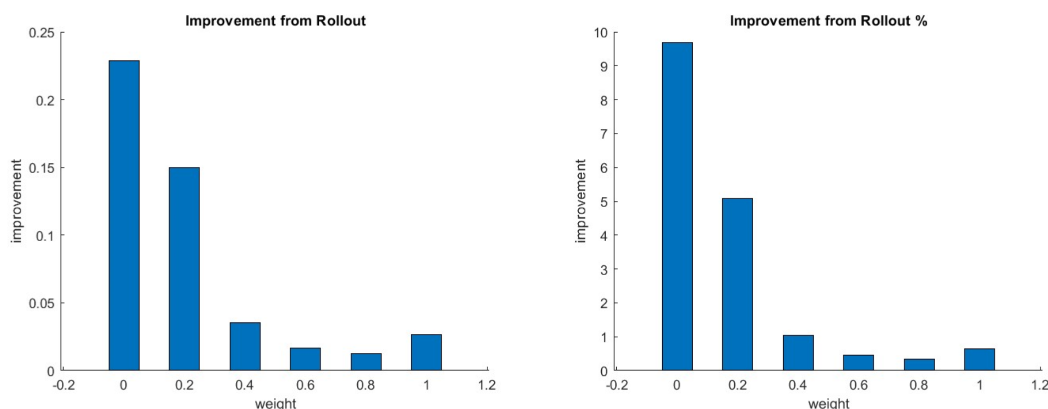
Παρατηρούμε ότι οι επιστρεφόμενες τιμές (δηλαδή οι προβολές των συστατικών της λύσης που παράγει το Rollout με χρήση του H) βελτιώνονται διαδοχικά. Μεγαλύτερος αριθμός πειραμάτων επιβεβαίωσε τη συμπεριφορά αυτή, οπότε μπορούμε να αποφανθούμε ότι ο ευρετικός αλγόριθμος H βελτιώνεται διαδοχικά. Επομένως, θα επιλέξουμε σαν τελική τιμή της αντικειμενικής συνάρτησης από το Rollout αυτή που παράγεται τελευταία, μετά τη δημιουργία του συνολικού μονοπατιού (διότι θα είναι πάντοτε ίση ή καλύτερη από τις προηγούμενες). Έχοντας το παραπάνω ως δεδομένο, θα συνεχίσουμε την πειραματική διαδικασία ώστε να εξετάσουμε καλύτερα τον τρόπο λειτουργίας του Rollout και τη βελτίωση που παρέχει.

Βελτίωση από το Rollout για διάφορες τιμές του βάρους *w*

Θα αντιπαραβάλλουμε τις τιμές της αντικειμενικής συνάρτησης, για βάρη *w* από 0 έως 1 με βήμα 0.2, που προέκυψαν αφενός από την απλή χρήση του ευρετικού *H* , και αφετέρου από τη χρήση του Rollout βάση του *H*.

Βάρος w	Τιμή H	Τιμή Rollout	Βελτίωση από Rollout	Βελτίωση από Rollout %
0	2.3469	2.1169	0.2312	9.8132
0.2	2.8958	2.7451	0.1507	5.2001
0.4	2.9166	2.8816	0.0351	1.1541
0.6	3.7491	3.7318	0.01478	0.4008
0.8	3.2258	3.2154	0.0109	0.2914
1	3.9655	3.9424	0.024	0.5862

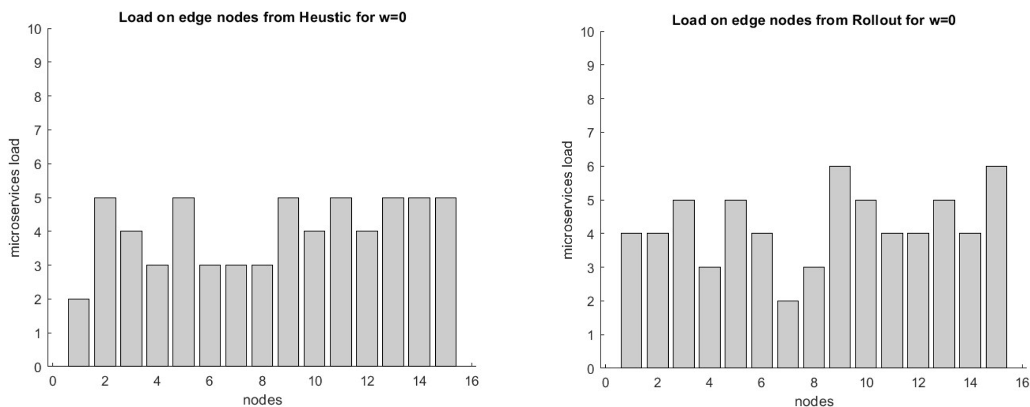
Πίνακας 5-8 Αποτελέσματα πειράματος



Εικόνα 5-20 Βελτίωση από τη χρήση του Rollout

Αρχικά παρατηρούμε ότι το Rollout επιτυγχάνει βελτίωση του αλγορίθμου για κάθε βάρος. Ακόμη, η βελτίωση φαίνεται να είναι αυξημένη για μικρές του *w*, γεγονός που επιβεβαιώθηκε από μεγαλύτερο αριθμό προσομοιώσεων. Αυτό συμβαίνει διότι για μικρότερα *w* το πρόβλημα εστιάζει στην ελαχιστοποίηση της καθυστέρησης των κόμβων τοποθέτησης, και άρα αποκτά μεγαλύτερη πολυπλοκότητα, καθώς τα microservices ανταγωνίζονται για την τοποθέτηση στους, περιορισμένων πόρων, κόμβους του edge στρώματος.

Ακόμη, ενδιαφέρον παρουσιάζει να παρατηρήσουμε το φορτίο των κόμβων που επιστρέφει ο απλός ευρετικός σε σύγκριση με το φορτίο από το Rollout, ως ένα μέτρο για τη διαφοροποίηση μεταξύ των δύο λύσεων. Ένα παράδειγμα φαίνεται παρακάτω, για βάρος *w*=0 και για τους κόμβους του edge στρώματος.



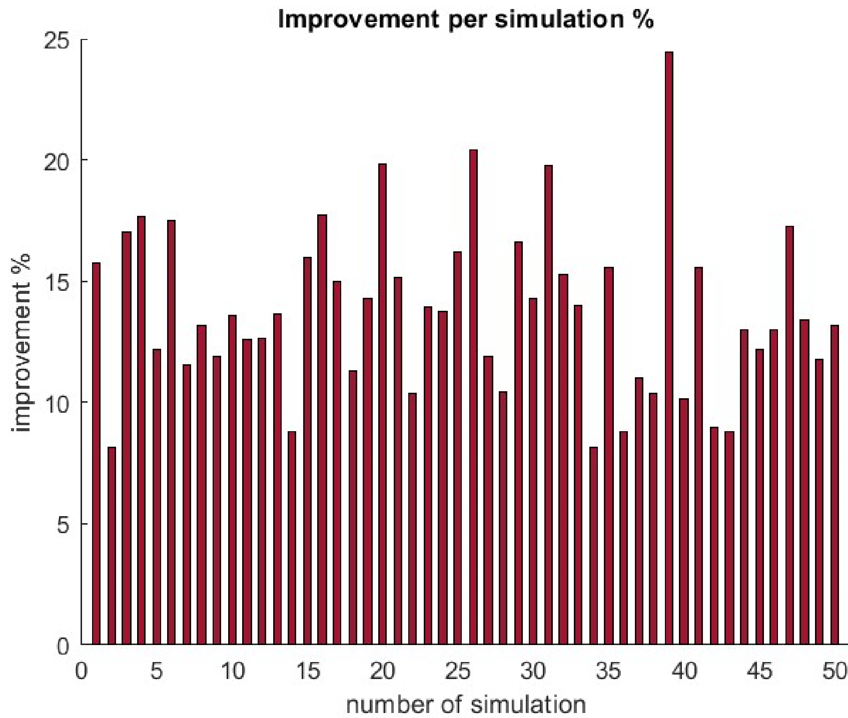
Εικόνα 5-21 Σύγκριση φορτίου κόμβων

Βλέπουμε ότι παρόλο που το Rollout βασίζεται στον ευρετικό *H* για την κατασκευή της λύσης, η λύση που τελικά παράγει διαφοροποιείται σημαντικά. Ακόμη, παρατηρούμε πως το Rollout επιτυγχάνει καλύτερη διαχείριση, τοποθετώντας συνολικά 64 microservices στο edge στρώμα προτού εξαντλήσει τους πόρους των κόμβων του, έναντι των 61 που τοποθέτησε ο ευρετικός. (Υπενθυμίζουμε ότι για *w*=0 το πρόβλημα εστιάζει στην ελαχιστοποίηση της καθυστέρησης, και άρα το στρώμα του edge είναι το πιο "ποθητό" για την τοποθέτηση των microservices). Στην πραγματικότητα το Rollout είναι ακόμη ευφυέστερο, καθώς τοποθετεί και περισσότερα microservices στους ταχύτερους κόμβους του edge στρώματος. Για παράδειγμα, στο προκείμενο πείραμα, ο ταχύτερος κόμβος ήταν ο κόμβος 1, με $l_1 = 1.09$ latency units, στον οποίο το Rollout κατάφερε να τοποθετήσει 4 microservices έναντι 2 από τον ευρετικό.

Ωστόσο, με τη μεμονωμένη αυτή εκτέλεση δεν μπορούμε να αποφανθούμε για το εύρος των τιμών και τη διακύμανση της βελτίωσης που προσφέρει το Rollout.

Μέση βελτίωση και διακύμανση

Σε μια προσπάθεια αποτίμησης των δυνατοτήτων του Rollout, θα διεξάγουμε μια σειρά 50 πειραμάτων με τα παραπάνω δεδομένα, και για βάρος *w*=0. Κάθε προσομοίωση θα παραμετροποιείται διαφορετικά εξαιτίας των τυχαία παραγόμενων τιμών.



Εικόνα 5-22 Βελτίωση ανά εκτέλεση

Μέση τιμή	Διακύμανση
13.7536	11.6115

Πίνακας 5-9 Μέση τιμή και διακύμανση βελτίωσης

Η μέση τιμή της ποσοστιαίας βελτίωσης είναι περίπου 13.75%, τιμή σίγουρα ικανοποιητική όσον αφορά τα προβλήματα βελτιστοποίησης. Από την άλλη, η μεγάλη τιμή της διακύμανσης φανερώνει μια εξάρτηση από τα δεδομένα εισόδου. Αυτό ίσως συμβαίνει επειδή σε κάποιες περιπτώσεις ο ευρετικός αλγόριθμος H δίνει από μόνος του ικανοποιητική λύση , με μικρά περιθώρια για περεταίρω βελτιστοποίηση. Σε κάθε περίπτωση, το Rollout εξαρτάται σε μεγάλο βαθμό από την ποιότητα και τη φύση του βασικού ευρετικού, και μια συστηματικότερη μελέτη των ιδιοτήτων του μπορεί ενδεχομένως να παράγει μια στρατηγική για την εκτίμηση των περιπτώσεων όπου κρίνεται σκόπιμη η χρήση του.

Χρόνος εκτέλεσης και πολυπλοκότητα

1. Βασικός Ευρετικός Αλγόριθμος H

Ο μέσος χρόνος εκτέλεσης του H για 100 εκτελέσεις με τα παραπάνω δεδομένα είναι t_{avgH}~0.0016 δευτερόλεπτα, δηλαδή περίπου 1000 φορές μικρότερος από αυτόν του GRAA. Αυτό συμβαίνει διότι ο αλγόριθμος αντί να υπολογίζει κάθε πιθανό συνδυασμό, συγκρίνει μεμονωμένα για κάθε microservice τον καλύτερο κόμβο, εγείροντας πολυωνυμική μέση πολυπλοκότητα $O(K * N)$ για την τοποθέτηση της κάθε εφαρμογής, και άρα συνολική πολυπλοκότητα

$O(M*K*N)$, όπου M ο αριθμός των εφαρμογών, K ο μέγιστος αριθμός microservices ανά εφαρμογή, και N ο αριθμός των κόμβων. Η πολυπλοκότητα ενδέχεται ωστόσο να αυξηθεί σε $O(K*N^2)$ στην περίπτωση πολλαπλών αποτυχιών για την τοποθέτηση μιας εφαρμογής.

2. Rollout

Ο αντίστοιχος μέσος χρόνος εκτέλεσης του Rollout ανέρχεται σε t_{avgR}~0.34 δευτερόλεπτα, δηλαδή περίπου 200 φορές μεγαλύτερος από αυτόν του *H*. Αυτό γίνεται κατανοητό αν υπολογίσουμε τον μέγιστο αριθμό κλήσεων του *H*, που είναι $m = M*K*N$, στην περίπτωση όπου δοκιμάζονται όλοι οι κόμβοι για την τοποθέτηση κάθε microservice, και όλες οι εφαρμογές φέρουν το μέγιστο αριθμό microservices που θεωρούμε στα δεδομένα του προβλήματος. Συνεπώς, η πολυπλοκότητα του Rollout είναι $m*O(H) = O(m*M*K*N)$=$O(M^2*K^2*N^2)$.

5.8 Σύγκριση των αποτελεσμάτων των GRAA, H, και Rollout με την ακριβή λύση του προβλήματος ΑΓΠ

Στο τελευταίο πείραμα θα συγκρίνουμε τα αποτελέσματα των αλγορίθμων που κατασκευάσαμε μέχρι στιγμής. Για τη σωστή αξιολόγηση των αποτελεσμάτων δεν αρκεί να συγκρίνουμε απλά τις τιμές της αντικειμενικής συνάρτησης που επιστρέφει ο καθένας, καθώς είναι πιθανό όλοι να δίνουν φτωχά αποτελέσματα βελτιστοποίησης. Για το λόγο αυτό θα χρησιμοποιήσουμε τον ενσωματωμένο "λύτη" (solver) του Matlab για προβλήματα γραμμικού προγραμματισμού. Αρχικά επιχειρείται η εύρεση κάποιων στοιχείων της βέλτιστης λύσης όπως το άνω και το κάτω φράγμα της, και στη συνέχεια το πρόβλημα επιλύεται με ακρίβεια μέσω της μεθόδου Branch and Bound [55].

Η ακριβής επίλυση σύνθετων προβλημάτων βελτιστοποίησης όπως το προκείμενο, είναι συχνά μια ιδιαίτερα χρονοβόρα διαδικασία (για ένα αρκετά μεγάλο πρόβλημα, ο χρόνος εκτέλεσης του branch and bound μπορεί να διαρκέσει δεκάδες ώρες!). Κάνοντας ένα συμβιβασμό για πρακτικούς λόγους, θα σμικρύνουμε το πρόβλημα διατηρώντας τις τιμές σε παρόμοια επίπεδα, ώστε να μπορούμε εύκολα να διεξάγουμε μια σειρά προσομοιώσεων. Τα δεδομένα που θα χρησιμοποιηθούν φαίνονται στους παρακάτω πίνακες:

	Αριθμός κόμβων	Αριθμός cpu units/κόμβο	Αριθμός ram units/κόμβο	Καθυστέρηση latency units	Κόστος Cost units
Στρώμα edge	5	12-15	4-8	0.5-2	6-9
Στρώμα fog	1	100	64	3.5-4.2	3.3-4.3
Στρώμα cloud	1	500	256	7.5-8.5	1-1.5

Πίνακας 5-10 Στοιχεία της πειραματικής υποδομής

Κόμβοι στρωμάτων	Σχετική Καθυστέρηση
Edge-Edge	0-1
Edge-Fog	3
Edge-Cloud	7
Fog-Fog	0-2
Fog-Cloud	3
Cloud-Cloud	0-3

Πίνακας 5-11 Σχετικές καθυστερήσεις

	# microservices	cpu demand/ms	ram demand/ms	latency limit	relative latency limit
Applications (1-20)	1-3	1-5	1-2	1.5-10	0.5-6

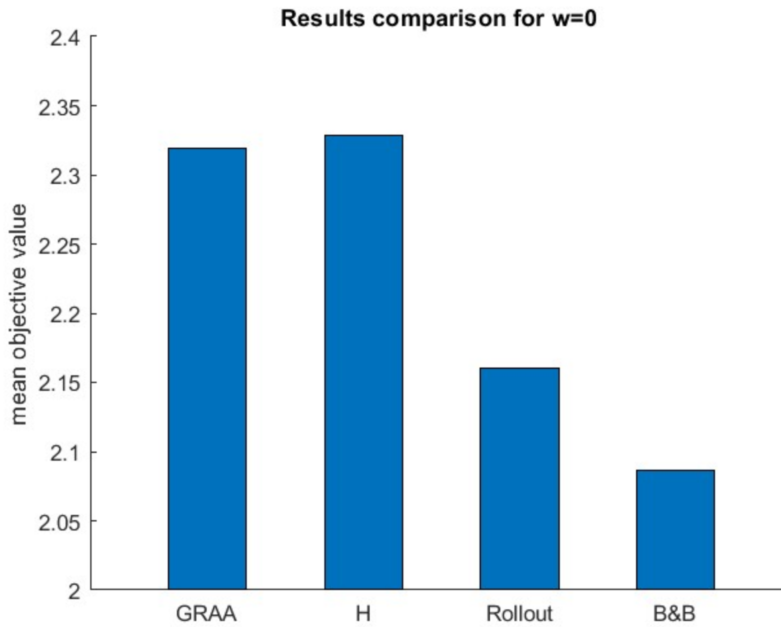
Πίνακας 5-12 Στοιχεία πειραματικών εφαρμογών

Αποτελέσματα

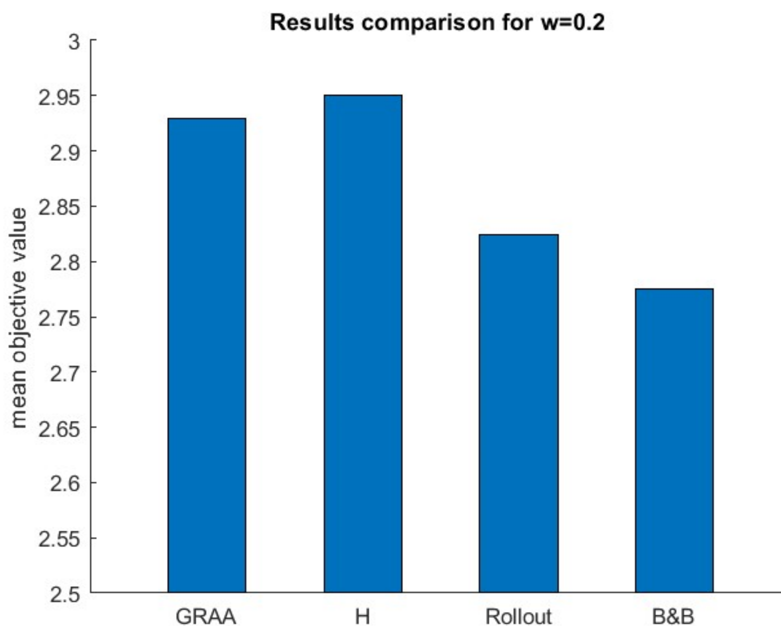
Για την εξαγωγή αξιόπιστων αποτελεσμάτων θα διεξάγουμε μια σειρά από 20 προσομοιώσεις για διάφορες τιμές του βάρους *w* από 0 έως 1. Για κάθε τιμή του βάρους θα βρίσκουμε τη μέση τιμή της αντικειμενικής συνάρτησης που επέστρεψε ο κάθε αλγόριθμος και θα αντιπαραβάλλουμε τα αποτελέσματα. Καθώς το πλήθος των προσομοιώσεων είναι μεγάλο, για πρακτικούς λόγους θα θέσουμε τις εξής δύο συνθήκες τερματισμού για τον ακριβή αλγόριθμο του Branch and Bound:

- Τερμάτισε όταν η σχετική διαφορά μεταξύ του υπολογισμένου άνω και κάτω ορίου της βέλτιστης λύσης φτάσει στο 0.05%.
- Τερμάτισε όταν εξερευνηθούν 5.000.000 "κόμβοι" χωρίς να βρεθεί λύση που να ικανοποιεί την παραπάνω συνθήκη. Η ευρεθείσα λύση θεωρείται άκυρη αν η σχετική διαφορά άνω και κάτω ορίου ξεπερνά το 0.25%.

Οι συνθήκες αυτές εξασφαλίζουν μια λύση που σε κάθε περίπτωση είναι ικανοποιητικά κοντά στη βέλτιστη για τις ανάγκες του πειράματος. Ο χρόνος εκτέλεσης του Branch and Bound έχει άνω όριο λόγω της 2ης συνθήκης τερματισμού, που είναι ~ 15 λεπτά.

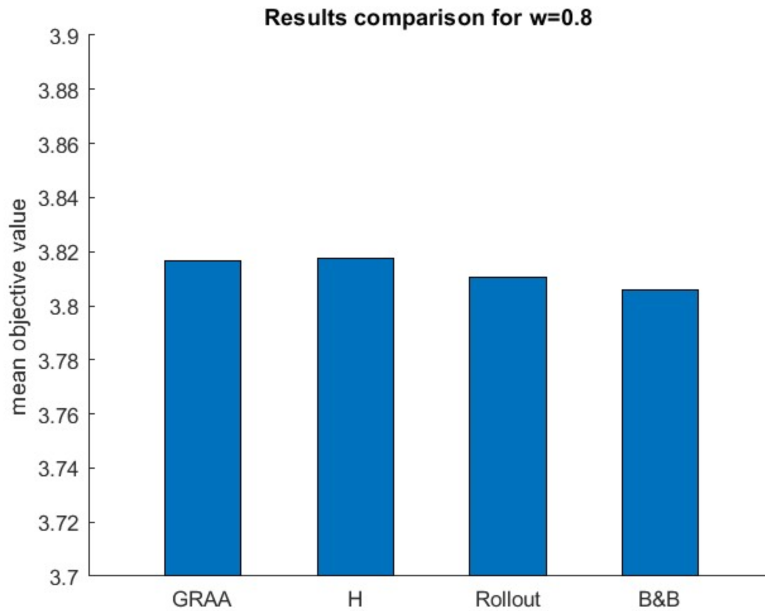


Εικόνα 5-23 Σύγκριση αποτελεσμάτων για w=0

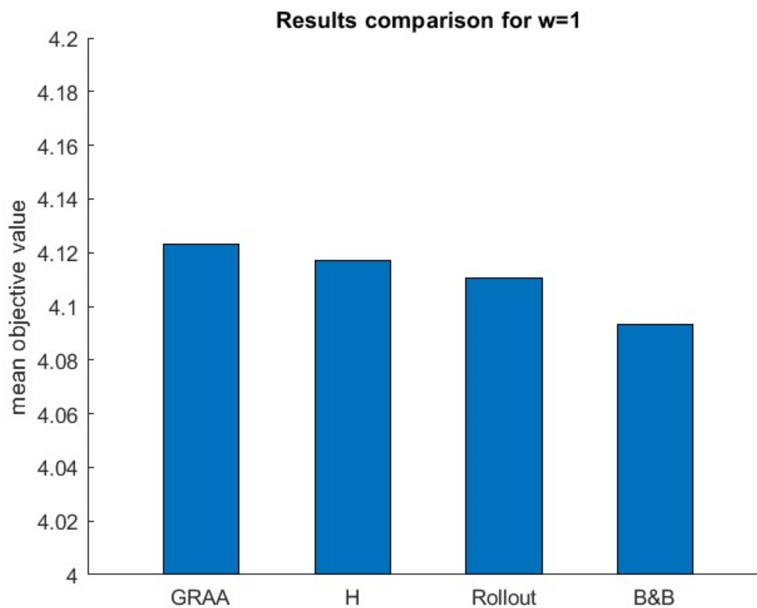


Εικόνα 5-24 Σύγκριση αποτελεσμάτων για w=0.2

Για χαμηλές τιμές του βάρους παρατηρούμε σημαντικές διαφορές στην επίδοση των αλγορίθμων. Ο *GRAA* δίνει ελαφρώς καλύτερα αποτελέσματα από τον *H*, ενώ το *Rollout* επιστρέφει με διαφορά τα καλύτερα αποτελέσματα και η λύση του είναι πολύ κοντά στη βέλτιστη λύση του ακριβούς αλγορίθμου. Πιο συγκεκριμένα, για *w*=0 το αποτέλεσμα του Rollout απέχει μόλις 3.5% από τη βέλτιστη λύση, έναντι του 11.1% από το αποτέλεσμα του GRAA και 11.6% από του βασικού ευρετικού H.



Εικόνα 5-25 Σύγκριση αποτελεσμάτων για w=0.8



Εικόνα 5-26 Σύγκριση αποτελεσμάτων για w=1

Για μεγαλύτερες τιμές του βάρους, το πρόβλημα απλοποιείται σε μεγάλο βαθμό (καθώς εστιάζει στην ελαχιστοποίηση του κόστους, και άρα δεν υπάρχουν πολλές συγκρούσεις στη διαδικασία). Για το λόγο αυτό, και οι 4 αλγόριθμοι επιστρέφουν πολύ κοντινές τιμές, διατηρώντας ωστόσο την προηγούμενη ιεραρχία $result(H) > result(GRAA) > result(Rollout) > result(B\&B)$, με τη διαφορά ότι για *w*=1 ο ευρετικός αλγόριθμος *H* έδωσε ελαφρώς καλύτερο (χαμηλότερο) αποτέλεσμα από τον GRAA.

5.9 Ανακεφαλαίωση, συμπεράσματα και συνολική αξιολόγηση

Η παρούσα εργασία εστίασε στην ανεύρεση αλγοριθμικών μηχανισμών για την ενσωμάτωσή τους σε συστήματα ενορχηστρωτών, με σκοπό τη βελτιστοποίηση της εξυπηρέτησης των εφαρμογών, και συγκεκριμένα την ελαχιστοποίησης ενός βεβαρυμμένου συνδυασμού του μέσου κόστους και της μέσης καθυστέρησης για τα microservices. Αρχικά αναπτύχθηκε ο άπληστος αλγόριθμος *GRAA*, ο οποίος αν και επέστρεψε ικανοποιητικά αποτελέσματα, ήταν λιγότερο αποδοτικός σε μεγαλύτερα προβλήματα εξαιτίας της πολυπλοκότητας του. Στη συνέχεια αναλύθηκε η τεχνική του *Rollout*, και εφαρμόσθηκε στην πράξη για την επίλυση του ίδιου προβλήματος με τη βοήθεια του ευρετικού αλγορίθμου *H*. Τα αποτελέσματα του Rollout ανέδειξαν τη χρησιμότητα του για την επίλυση σύνθετων προβλημάτων βελτιστοποίησης, καθώς οι λύσεις του αφενός ήταν σημαντικά καλύτερες από αυτές του ευρετικού και αφετέρου προσέγγιζαν σε μεγάλο βαθμό τις βέλτιστες (περίπου στο 3.5% επί της βέλτιστης λύσης). Ωστόσο, η πραγματική εμβέλεια των ικανοτήτων του Rollout παραμένει ένα ανοιχτό ζήτημα για μελλοντική ενασχόληση. Στο επόμενο και τελευταίο κεφάλαιο, παρουσιάζουμε μερικά ακόμη ενδιαφέροντα πεδία μελλοντικής εργασίας.

6 Επεκτάσεις – μελλοντική εργασία

Στο κεφάλαιο αυτό θα παρουσιάσουμε ιδέες για πιθανές προεκτάσεις, εξερευνώντας διαφορετικές πτυχές του προβλήματος αλλά και τροποποιήσεις για μια πληρέστερη και πιο εύρωστη προσέγγιση. Επισημαίνουμε ότι σε κάθε νέο αλγόριθμο που δημιουργούμε, μπορούμε και πάλι να κατασκευάσουμε τον αντίστοιχο αλγόριθμο Rollout για την περαιτέρω βελτιστοποίηση όπου κρίνεται απαραίτητο.

6.1 Εξέταση του "σε σύνδεση" (online) προβλήματος

Η παρούσα εργασία εξέτασε την "εκτός σύνδεσης" (offline) έκδοση του προβλήματος. Αυτό σημαίνει πως το σύνολο των εισόδων (των εφαρμογών και των χαρακτηριστικών τους) ΄θεωρείτο γνωστό πριν την εκκίνηση των αλγορίθμων. Αντιθέτως, στο online σενάριο οι εφαρμογές προς εξυπηρέτηση καταφθάνουν δυναμικά, και άρα ένας online αλγόριθμος πρέπει να λάβει αποφάσεις σε συνθήκες αβεβαιότητας. Ο καταλληλότερος αλγόριθμος για την αναγωγή του σε online έκδοση είναι ο GRAA, καθώς η λογική της μεμονωμένης βέλτιστης τοποθέτησης ανά εφαρμογή συνάδει με το δυναμικό χαρακτήρα του νέου προβλήματος. Ο μόνος περιοριστικός παράγοντας που εντοπίζεται είναι πως ο GRAA αρχικά κατέτασσε το σύνολο των εφαρμογών στη FIFO ουρά σύμφωνα με το άνω όριο καθυστέρησης, προς αποφυγή φαινομένων μπλοκαρίσματος όταν η αντικειμενική συνάρτηση εστίαζε στην ελαχιστοποίηση της καθυστέρησης εξυπηρέτησης. Παρακάτω θα παρουσιάσουμε εν συντομία την online έκδοση του προβλήματος, και μια πιθανή τροποποίηση του GRAA για την επίλυσή του:

Περιγραφή του προβλήματος

- Η υποδομή και τα χαρακτηριστικά της διατηρούνται ακριβώς όπως στο offline πρόβλημα.
- Η περιγραφή των εφαρμογών διατηρείται ακριβώς όπως στο offline πρόβλημα.
- Οι εφαρμογές πλέον εμφανίζονται δυναμικά σύμφωνα με κάποιον ρυθμό αφίξεων (arrival rate) *ar* μετρημένο σε *εφαρμογές/μονάδα χρόνου*. Επιπλέον, κάθε microservice *k της εφαρμογής j* φέρει ένα χρόνο εξυπηρέτησης mt_{jk} μετρημένο σε μονάδες χρόνου. Αυτό σημαίνει πως μετά τη διαδικασία τοποθέτησης του microservice σε κόμβο της τοπολογίας, απαιτούνται mt_{jk} μονάδες χρόνου για την επεξεργασία του, και ύστερα οι πόροι που κατέλαβε αποδεσμεύονται από τον κόμβο τοποθέτησης. Σημειώνουμε πως ο χρόνος επεξεργασίας μπορεί να μεταβάλλεται και μεταξύ των κόμβων (καθώς αλλάζει η δυναμική των υπολογιστικών συστημάτων, μπορεί να υπάρχουν επιταχυντές υλικού κλπ.), στην οποία περίπτωση η μεταβλητή λαμβάνει τη μορφή mt_{ijk}.

Online GRAA

Η λειτουργία του GRAA παραμένει υπό το πλείστων η ίδια. Αναλαμβάνει τις εφαρμογές κατά σειρά αφίξεως, και τοποθετεί τα microservices αυτών στο βέλτιστο συνδυασμό κόμβων της τοπολογίας. Ακόμη, φροντίζει για την αποδέσμευση των κατεληφθέντων πόρων μετά το πέρας του χρόνου εξυπηρέτησης κάθε microservice. Ωστόσο, καθώς οι εφαρμογές δεν καταφθάνουν με αύξουσα σειρά άνω ορίων καθυστέρησης, για την αποφυγή μπλοκαρίσματος θα κάνουμε την εξής τροποποίηση: Όταν η χρησιμοποίηση των edge κόμβων φτάσει πάνω από ένα (πειραματικά/στατιστικά υπολογισμένο) όριο u_{thr}, ο αλγόριθμος θα τοποθετεί στο στρώμα του edge μόνο εφαρμογές που δεν δύναται να τοποθετηθούν αλλού, ενώ θα προωθεί τις υπόλοιπες στα ανώτερα στρώματα.

Για περισσότερες πληροφορίες σχετικά με τους εντός και εκτός σύνδεσης αλγορίθμους παραπέμπουμε στο [56]

6.2 Χρήση γράφων για την αναπαράσταση των εφαρμογών

Στην αρχή του κεφαλαίου 5 κάναμε μια σύντομη νύξη στη χρήση γράφων για την αναπαράσταση των εφαρμογών, αλλά μετέπειτα δεν υιοθετήσαμε τη λογική αυτή εξαιτίας της πολυπλοκότητας που εισάγει τόσο στη δόμηση των αλγορίθμων, όσο και στην εύρεση κατάλληλων πειραματικών τιμών για την αποφυγή σφαλμάτων. Κρίνουμε σκόπιμη την επανεξέταση αυτής της θεώρησης, καθώς το μοντέλο του γράφου είναι το πλέον αντιπροσωπευτικό της πραγματικής δομής των cloud-native εφαρμογών με χρήση microservices. Ακόμη, σκοπεύουμε στη χρήση τιμών αντλημένων από πραγματικές εφαρμογές για μια ποιοτικότερη και πιο ρεαλιστική λύση.

6.3 Επίγνωση τοποθεσίας – Εξατομικευμένη καθυστέρηση

Σε αμφότερα τα προβλήματα που παρουσιάσαμε, θεωρήσαμε αμελητέες τις διαφορές στην καθυστέρηση των αιτημάτων από και προς συγκεκριμένο κόμβο εξαιτίας της χωρικής τους διασποράς. Σε ένα πραγματικό σενάριο, δύο αιτήματα που παράγονται σε διαφορετικά σημεία του τοπικού δικτύου, θα φέρουν σημαντικά διαφορετικές καθυστερήσεις για τον ίδιο edge κόμβο, ενώ οι διαφορές θα συρρικνώνονται ποσοστιαία καθώς απομακρυνόμαστε στους κόμβους της τοπολογίας, και πράγματι θα είναι σχεδόν αμελητέες για το ανώτερο στρώμα του cloud. Για παράδειγμα, ένα αυτόνομο όχημα μπορεί να έχει καθυστέρηση 2ms προς έναν κοντινό MEC εξυπηρετητή, ενώ ένα άλλο που βρίσκεται σε απόσταση κάποιων χιλιομέτρων μακρινότερα να φέρει καθυστέρηση της τάξεως 5-10 ms για τον ίδιο εξυπηρετητή. Ωστόσο, και τα 2 οχήματα θα φέρουν, σχεδόν, την ίδια καθυστέρηση προς τους απομακρυσμένους cloud κόμβους.

Για να ενσωματώσουμε την παραπάνω διαπίστωση στα δεδομένα μας, αρκεί να θεωρήσουμε περισσότερους του ενός κόμβους παραγωγής δεδομένων στην τοπολογία μας, ομαδοποιώντας άρα, χωρικά, τις εφαρμογές.

6.4 Επίγνωση δικτύωσης

Ένα προφανές μειονέκτημα της προσέγγισής μας είναι η απαλοιφή των δικτυακών μεταβλητών. Ένα πρώτο βήμα προς αυτή την κατεύθυνση είναι η θεώρηση του εύρους ζώνης στους συνδέσμους του γράφου της τοπολογίας μας, εισάγοντας έτσι έναν επιπλέον περιορισμό στο πρόβλημα.

6.5 Διερεύνηση άλλων συναφών προβλημάτων

Το πρόβλημα της δέσμευσης πόρων μπορεί να βελτιστοποιηθεί σε πολλά ακόμη επίπεδα πέραν της καθυστέρησης και του κόστους των microservices. Ενδεικτικά αναφέρουμε την ενεργειακή κατανάλωση, το βαθμό χρησιμοποίησης των πόρων κόμβων, το βαθμό χρησιμοποίησης των δικτυακών πόρων των συνδέσμων, την ελαχιστοποίηση του συνολικού χρόνου εκτέλεσης (makespan) κλπ. Μια ενδιαφέρουσα μελλοντική επέκταση θα ήταν η δημιουργία ενός πιο εξελιγμένου αλγορίθμου πολυδιάστατης βελτιστοποίησης με περισσότερα του ενός βάρη για τη βελτιστοποίηση της εξυπηρέτησης σύμφωνα με τις εκάστοτε ανάγκες.

6.6 Χρήση αλγορίθμων πλειστηριασμού

Οι αλγόριθμοι πλειστηριασμού παρουσιάζουν μεγάλη συνάφεια με το πρόβλημα της δέσμευσης πόρων. Κάθε αίτημα θα μπορούσε να εκφράζει ένα όφελος για κάθε κόμβο της υποδομής, ανάλογα με την καθυστέρηση προς αυτόν, την ύπαρξη συγκεκριμένων συστημάτων (π.χ. ASICs) κλπ. Στη συνέχεια, η διαδικασία του πλειστηριασμού θα αναλάμβανε την αντιστοίχιση, με σκοπό τη μεγιστοποίηση του συνολικού οφέλους των αιτημάτων. Στην ανάποδη περίπτωση, κάθε κόμβος μπορεί να εκφράζει ένα όφελος για την εξυπηρέτηση κάθε αιτήματος, σύμφωνα με το βαθμό χρησιμοποίησής του, το κόστος για την επεξεργασία, την ενεργειακή κατανάλωση κλπ. Στη συνέχεια η αντιστοίχιση θα συνέβαινε και πάλι με τη μέθοδο του πλειστηριασμού, με σκοπό τη βελτιστοποίηση κάποιου από τους ανωτέρω παράγοντες.

Βιβλιογραφία

[1] J. Singh, J. Powles, T. Pasquier, and J. Bacon, "Data flow management and compliance in cloud computing," *IEEE Cloud Computing*, vol. 2, no. 4, pp. 24–32, Jul. 2015, doi: 10.1109/MCC.2015.69.

[2] Eric Griffith, "What Is Cloud Computing?", Accessed: Mar. 04, 2022. [Online]. Available: https://www.pcmag.com/how-to/what-is-cloud-computing

[3] "What is Cloud Computing_ _ IBM", Accessed: Mar. 04, 2022. [Online]. Available: https://www.ibm.com/cloud/learn/cloud-computing

[4] L. Qian, Z. Luo, Y. Du, and L. Guo, "Cloud computing: An overview," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009, vol. 5931 LNCS, pp. 626–631. doi: 10.1007/978-3-642-10665-1_63.

[5] "26 Cloud Computing Statistics, Facts & Trends for 2022." https://www.cloudwards.net/cloud-computing-statistics/ (accessed Aug. 07, 2022).

[6] "Cloud computing - statistics on the use by enterprises - Statistics Explained." https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Cloud_computing_-_statistics_on_the_use_by_enterprises (accessed Aug. 07, 2022).

[7] "13 Benefits of Cloud Computing for Your Business _ GlobalDots." https://www.globaldots.com/resources/blog/cloud-computing-benefits-7-key-advantages-for-your-business/ (accessed Aug. 07, 2022).

[8] "Virtualization - Wikipedia", Accessed: Mar. 04, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Virtualization

[9] "What is virtualization_ _ Opensource.com", Accessed: Mar. 04, 2022. [Online]. Available: https://opensource.com/resources/virtualization

[10] "What is a hypervisor_", Accessed: Mar. 04, 2022. [Online]. Available: https://www.redhat.com/en/topics/virtualization/what-is-a-hypervisor

[11] "What Is Cloud Computing_ A Beginner's Guide _ Microsoft Azure", Accessed: Mar. 04, 2022. [Online]. Available: https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/

[12] "What is a Container_ _ App Containerization _ Docker." https://www.docker.com/resources/what-container (accessed Mar. 04, 2022).

[13] "What is IaaS (Infrastructure-as-a-Service) _ IBM." https://www.ibm.com/cloud/learn/iaas (accessed Mar. 11, 2022).

[14] "What is PaaS_ Platform as a Service _ Microsoft Azure." https://azure.microsoft.com/en-us/overview/what-is-paas/ (accessed Mar. 11, 2022).

[15] "IaaS vs PaaS vs SaaS." https://www.redhat.com/en/topics/cloud-computing/iaas-vs-paas-vs-saas (accessed Mar. 11, 2022).

[16] P. P. Ray, "An Introduction to Dew Computing: Definition, Concept and Implications," *IEEE Access*, vol. 6. Institute of Electrical and Electronics Engineers Inc., pp. 723–737, Nov. 16, 2017. doi: 10.1109/ACCESS.2017.2775042.

[17] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct. 2016, doi: 10.1109/JIOT.2016.2579198.

[18] T. Alam, "A Reliable Communication Framework and Its Use in Internet of Things (IoT) IoT-Security View project Smart Home Automation System View project A Reliable Communication Framework and Its Use in Internet of Things (IoT)," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology © 2018 IJSRCSEIT*, vol. 5, no. 10, pp. 450–456, 2018, [Online]. Available: https://www.researchgate.net/publication/325645304

[19] "Edge computing - Wikipedia." Accessed: Aug. 05, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Edge_computing

[20] "Edge computing - Wikipedia." https://en.wikipedia.org/wiki/Edge_computing (accessed Mar. 11, 2022).

[21] R. Dangi, P. Lalwani, G. Choudhary, I. You, and G. Pau, "Study and Investigation on 5G Technology: A Systematic Review.," *Sensors (Basel)*, vol. 22, no. 1, Dec. 2021, doi: 10.3390/s22010026.

[22] Y. C. Hu, M. Patel, D. Sabella, and V. Young, "ETSI White Paper #11 Mobile Edge Computing - a key technology towards 5G," 2015. [Online]. Available: www.etsi.org

[23] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proceedings of the International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Jun. 2015, vol. 2015-June, pp. 37–42. doi: 10.1145/2757384.2757397.

[24] Y. Kim, J. Kong, and A. Munir, "CPU-Accelerator Co-Scheduling for CNN Acceleration at the Edge," *IEEE Access*, vol. 8. Institute of Electrical and Electronics Engineers Inc., pp. 211422–211433, 2020. doi: 10.1109/ACCESS.2020.3039278.

[25] "Page 2 of [Discussion post]Need of Cloud,Fog and Edge Computing - Huawei Enterprise Support Community." https://forum.huawei.com/enterprise/en/discussion-post-need-of-cloud-fog-and-edge-computing/thread/748771-893?page=2 (accessed Mar. 11, 2022).

[26] "What Is Edge Computing_ - Cisco." https://www.cisco.com/c/en/us/solutions/computing/what-is-edge-computing.html (accessed Aug. 05, 2022).

[27] M. Gerla, D. Huang, Association for Computing Machinery. Special Interest Group on Data Communications, ACM Digital Library., and F. ACM SIGCOMM Conference (2012 : Helsinki, *MCC'12 : proceedings of the 1st ACM Mobile Cloud Computing Workshop : August 17, 2012, Helsinki, Finland*.

[28] "Cloudlet - Wikipedia." https://en.wikipedia.org/wiki/Cloudlet (accessed Mar. 11, 2022).

[29] "Introduction to cloud-native applications _ Microsoft Docs." https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/introduction (accessed Mar. 11, 2022).

[30] "What is Cloud Native_ _ Microsoft Docs." https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/definition (accessed Mar. 11, 2022).

[31] D. Gannon, R. Barga, W. Services, and N. Sundaresan, "Cloud-Native Applications."

[32] N. Kratzke and P. C. Quint, "Understanding cloud-native applications after 10 years of cloud computing - A systematic mapping study," *Journal of Systems and Software*, vol. 126, pp. 1–16, Apr. 2017, doi: 10.1016/j.jss.2017.01.001.

[33] 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," *IEEE Access*, vol. 9. Institute of Electrical and Electronics Engineers Inc., pp. 10903–10924, 2021. doi: 10.1109/ACCESS.2021.3050155.

[34] "What is Container Orchestration_ _ VMware Glossary." https://www.vmware.com/topics/glossary/content/container-orchestration.html (accessed Jul. 16, 2022).

[35] "Kubernetes - Wikipedia." Accessed: Aug. 12, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Kubernetes

[36] L. Kubernetes Basics, "Home Blog Training Partners Community Case Studies Production-Grade Container Orchestration," 2022. Accessed: Aug. 12, 2022. [Online]. Available: https://kubernetes.io

[37] X. Li, Z. Lian, X. Qin, and W. Jie, "Topology-aware resource allocation for IoT services in clouds," *IEEE Access*, vol. 6, pp. 77880–77889, 2018, doi: 10.1109/ACCESS.2018.2884251.

[38] M. Leconte, G. S. Paschos, P. Mertikopoulos, and U. Kozat, "A Resource Allocation Framework for Network Slicing."

[39] N. Kiran, X. Liu, S. Wang, and Y. Changchuan, "VNF Placement and Resource Allocation in SDN/NFV-enabled MEC Networks," 2020.

[40] D. Santoro, D. Zozin, D. Pizzolli, F. de Pellegrini, and S. Cretti, "Foggy: A Platform for Workload Orchestration in a Fog Computing Environment," Sep. 2018, doi: 10.1109/CloudCom.2017.62.

[41] S. Pallewatta, V. Kostakos, and R. Buyya, "Microservices-based IoT application placement within heterogeneous and resource constrained fog computing

environments," in *UCC 2019 - Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, Dec. 2019, pp. 71–81. doi: 10.1145/3344341.3368800.

[42] C. T. Joseph and K. Chandrasekaran, "IntMA: Dynamic Interaction-aware resource allocation for containerized microservices in cloud environments," *Journal of Systems Architecture*, vol. 111, Dec. 2020, doi: 10.1016/j.sysarc.2020.101785.

[43] Institute of Electrical and Electronics Engineers., *2018 IEEE International Conference on Communications (ICC) : proceedings : Kansas City, MO, USA, 20 -24 May 2018.*

[44] X. Wu, W. Jiang, Y. Zhang, and W. Yu, "Online combinatorial based mechanism for MEC network resource allocation," *International Journal of Communication Systems*, vol. 32, no. 7, May 2019, doi: 10.1002/dac.3928.

[45] "Greedy Algorithm," *Wikipedia*. Accessed: Aug. 05, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Greedy_algorithm

[46] "MATLAB - Wikipedia." Accessed: Aug. 05, 2022. [Online]. Available: https://en.wikipedia.org/wiki/MATLAB

[47] A. N. Kolmogorov, "On tables of random numbers," *Theoretical Computer Science*, vol. 207, no. 2, pp. 387–395, Nov. 1998, doi: 10.1016/S0304-3975(98)00075-9.

[48] A. Vardy, "Algorithmic Complexity in Coding Theory and the Minimum Distance Problem."

[49] "Heuristic - Wikipedia." Accessed: Jul. 09, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Heuristic

[50] "Metaheuristic - Wikipedia." https://en.wikipedia.org/wiki/Metaheuristic (accessed Jul. 09, 2022).

[51] D. P. Bertsekas, "Rollout Algorithms for Discrete Optimization: A Survey," 2010.

[52] D. P. Bertsekas, J. N. Tsitsiklis, C. Wu, D. P. Bertsekas, J. N. Tsitsiklis, and C. Wu ', "Rollout Algorithm For Combinatorial Optimization ROLLOUT ALGORITHMS FOR COMBINATORIAL OPTIMIZATION," 1997.

[53] "Graph Theory _ Brilliant Math & Science Wiki", Accessed: Jul. 09, 2022. [Online]. Available: https://brilliant.org/wiki/graph-theory/

[54] "Graph theory - Wikipedia." Accessed: Jul. 09, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Graph_theory

[55] "Branch and bound - Wikipedia." Accessed: Aug. 10, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Branch_and_bound

[56] "Online algorithm - Wikipedia." Accessed: Aug. 19, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Online_algorithm