ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΤΟΜΙΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μάιος 2023

Ατομική Διπλωματική Εργασία

ΑΞΙΟΛΟΓΗΣΗ ΕΠΙΔΟΣΕΩΝ ΤΗΣ ΕΦΑΡΜΟΓΗΣ ML-ENABLED DRONE ΣΕ EMULATED EDGE ΠΕΡΙΒΑΛΛΟΝΤΑ

Δημήτρης Γρηγοράς

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2023

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Αξιολόγηση επιδόσεων της εφαρμογής ML-enabled drone σε emulated edge περιβάλλοντα

Δημήτρης Γρηγοράς

Επιβλέπων Καθηγητής
Γιώργος Πάλλης

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2023

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον κ. Μωυσή Συμεωνίδη για την ανεκτίμητη συμβολή του καθ' όλη τη διάρκεια του έργου, από την ανάπτυξη και την υλοποίησή του μέχρι την πραγματοποίηση και την ανάλυση των πειραμάτων. Η βοήθεια και η καθοδήγησή του ήταν καθοριστική για τη διαμόρφωση της ερευνητικής μου εργασίας και την επιτυχή ολοκλήρωσή της.

Περίληψη

Στην τεχνολογικά προηγμένη κοινωνία του 2023, το Διαδίκτυο των Πραγμάτων (IoT) έχει ενσωματωθεί όλο και περισσότερο στην καθημερινή μας ζωή. Αυτή η ταχεία πρόοδος έχει μετατρέψει τις κάποτε απλές τεχνολογίες σε έξυπνα και διασυνδεδεμένα συστήματα. Ένας τομέας που έχει επωφεληθεί ιδιαίτερα από αυτή την πρόοδο είναι η διαχείριση της οδικής κυκλοφορίας, η οποία εξελίσσεται πλέον σε ένα "έξυπνο" μοντέλο για την ενίσχυση της αποτελεσματικότητας και της ευκολίας.

Καθώς η οδική κυκλοφορία συνεχίζει να επεκτείνεται και να γίνεται πιο πολύπλοκη, η ανάγκη για δεδομένα σε πραγματικό χρόνο και ακρίβεια, έχει καταστεί υψίστης σημασίας. Η ενσωμάτωση συσκευών και αισθητήρων IoT στις οδικές υποδομές έχει επιτρέψει τη συλλογή πολύτιμων, αλλά και τεράστιου όγκου, δεδομένων σχετικά με τις συνθήκες κυκλοφορίας.

Τα δεδομένα αυτά τροφοδοτούνται σε μοντέλα ML ειδικά σχεδιασμένα για την ανίχνευση και καταμέτρηση οχημάτων. Μόλις τα μοντέλα ML επεξεργαστούν τα δεδομένα, παράγουν πληροφορίες σε πραγματικό χρόνο.

Διεξήχθη ερευνητικό έργο για την ανάπτυξη μιας παραμετροποιήσιμης εφαρμογής για την καταμέτρηση οχημάτων με τη χρήση ενός drone, ενός σταθμού βάσης και ενός cloud. Το drone καταγράφει εικόνες, οι οποίες επεξεργάζονταν από τον σταθμό βάσης με τη χρήση μοντέλων μηχανικής μάθησης για την καταμέτρηση του αριθμού των οχημάτων σε κάθε εικόνα. Επίσης, η εφαρμογή χρησιμοποίει υποδομή 5G για αποτελεσματική επικοινωνία μεταξύ των services. Το cloud, αν και θεωρητικό, προοριζόταν για να ειδοποιά τους οδηγούς με βάση τις διαδρομές τους.

Η απόδοση της εφαρμογής αξιολογήθηκε με τη διεξαγωγή πειραμάτων σε ένα προσομοιωμένο περιβάλλον. Μετρήθηκαν διάφορες μετρικές για την αξιολόγηση της αποδοτικότητας και της αποτελεσματικότητας του προγράμματος. Η χρήση time variables παρείχε πληροφορίες σχετικά με τη χρονική διάρκεια και την καθυστέρηση κάθε microservice, επιτρέποντας την ανάλυση της απόδοσής τους. Δοκιμάστηκαν επίσης, διάφοροι αλγόριθμοι τεχνητής νοημοσύνης για την αξιολόγηση της ακρίβειας και της καθυστέρησης. Επιπρόσθετα, γίνετε έλεγχος της χρήσης και της απόδοσης του υλικού, όπως η μνήμη RAM και η CPU για το κάθε microservice. Οι μετρήσεις απόστασης δικτύου βοήθησαν στη βελτιστοποίηση της μετάδοσης δεδομένων. Πολλαπλά drones προστέθηκαν για τη δοκιμή της επεκτασιμότητας και τη διαχείριση μεγαλύτερων όγκων δεδομένων. Οι υπολογισμοί της χρήσης του δικτύου και του κόστους παρείχαν πολύτιμες πληροφορίες σχετικά με την οικονομική σκοπιμότητα.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Έξυπνες λύσεις για τη διαχείριση της οδικής κυκλοφορίας

Εν έτη 2023, σε μια τεχνολογικά ανεπτυγμένη κοινωνία, και σε μια κοινωνία όπου το IoT ενσωματώνεται όλο και περισσότερο στην καθημερινή μας ζωή, τεχνολογίες που θεωρούσαμε εμείς απλές μέχρι σήμερα, αναπτύσσονται ραγδαία και γίνονται «έξυπνες». Αυτό, έχει ως αποτέλεσμα η οδική κυκλοφορία που είναι μια από αυτές, να βρίσκετε σε συνεχή ανάπτυξη και να ακολουθεί «έξυπνο» μοντέλο τεχνολογιών. Έτσι, τα δεδομένα για τους δρόμους και συγκεκριμένα για την κυκλοφορία θα πρέπει να παρέχονται στους οδηγούς όσο πιο γρήγορα και πιο ακριβές γίνετε. Δεν μπορούμε να βασιστούμε σε δορυφόρους ή σε κάποιες άλλες εφαρμογές για να παρακολουθούμε την κυκλοφορία στους δρόμους αφού, τις πλείστες φορές τα δεδομένα δεν είναι τόσο up to date για μικρές είτε μεγάλες χρονικές στιγμές. Αυτό όμως μπορεί να ξεπεραστεί με την χρήση διάφορων μέσων, ίσως και πιο σύγχρονων τεχνολογιών, τα οποία θα συλλαμβάνουν τις αναγκαίες πληροφορίες σε πολύ μικρά χρονικά διαστήματα για την παρόν οδική κυκλοφορία σε συγκεκριμένους δρόμους. Η γενική ιδέα προϋποθέτει ότι, μετά την συλλογή των δεδομένων, την αποστολή τους σε κοντινά base stations εντός εμβέλειας και μέσω

διαδικτύου, ώστε να γίνει επεξεργασία και καταμέτρηση των οχημάτων. Αφού τελειώσει η επεξεργασία, τα αποτελέσματα θα προωθούνται στο cloud service το οποίο με την σειρά του θα ενημερώνει με διάφορα μέσα, τους επερχόμενους ή παρόν οδηγούς με την κυκλοφορία για όλους τους σχετικούς δρόμους.

Παρόλα αυτά, φαίνεται να παρουσιάζονται κάποιες δυσκολίες. Αρχικά, μία από τις κύριες προκλήσεις είναι η διασφάλιση της συλλογής και επεξεργασίας δεδομένων σε πραγματικό χρόνο, για την παροχή ακριβών και ενημερωμένων πληροφοριών για την κυκλοφορία. Η παράμετρος απόδοσης σε αυτό το σημείο είναι κρίσιμη, καθώς οποιαδήποτε καθυστέρηση στη συλλογή ή επεξεργασία δεδομένων μπορεί να οδηγήσει στην παροχή εσφαλμένων πληροφοριών στους οδηγούς. Μια άλλη πρόκληση αφορά την αξιοπιστία του δικτύου επικοινωνίας που χρησιμοποιείται για τη μετάδοση των δεδομένων μεταξύ των σταθμών βάσης, συσκευών συλλογής δεδομένων και των υπηρεσιών νέφους. Το σύστημα πρέπει να είναι σχεδιασμένο για να διαχειρίζεται μεγάλο όγκο δεδομένων και να παρέχει αξιόπιστη συνδεσιμότητα, ακόμη και σε περιοχές με κακή κάλυψη δικτύου. Επιπρόσθετα, ενδέχεται να υπάρξουν προκλήσεις που σχετίζονται με την ενσωμάτωση διαφορετικών τεχνολογιών και συστημάτων. Οι τεχνολογίες πρέπει να είναι σε θέση να επικοινωνούν απρόσκοπτα μεταξύ τους και να ενσωματώνουν τα συστήματα επεξεργασίας και αποθήκευσης δεδομένων τους. Η διασφάλιση της συμβατότητας και της διαλειτουργικότητας μεταξύ διαφορετικών τεχνολογιών μπορεί να αποτελέσει σημαντική πρόκληση σε ένα τόσο πολύπλοκο σύστημα. Τέλος, έχουμε και την δυσκολία του κόστους στην χρήση hardware ή infrastructure για να μπορούμε να δημιουργήσουμε αυτή την ιδέα.

1.2 Drone-based παρακολούθηση κυκλοφορίας με Edge Computing

Προχωρώντας, ένα μέσο πολλών χρήσεων, ακόμα και ενημερώσεων διάφορων εργασιών, παρατηρείτε τον τελευταίο καιρό να είναι τα drones. Τα drones είναι «ιπτάμενα ρομπότ» που μπορεί να ελέγχονται εξ αποστάσεως ή να λειτουργούν με software-controlled flight plans. Τα τελευταία χρόνια, τα μη επανδρωμένα αεροσκάφη έχουν αποκτήσει μεγάλη δημοτικότητα και χρησιμοποιούνται σε ένα ευρύ φάσμα εφαρμογών. Το σημαντικότερο χαρακτηριστικό για την επιλογή τους στην έρευνα, είναι η αυξημένη προσβασιμότητα στην εναέρια απεικόνιση και τη συλλογή real-time δεδομένων. Έτσι, η έρευνα μελετά

την χρήση drone, αντενών basestation αλλά και cloud, όλα σε συνεργασία, ώστε τα «έξυπνα» αυτοκίνητα να είναι συνεχώς up to date για την κυκλοφορία στον δρόμο βάση της διαδρομής που θα κινηθούν, μέχρι και τον τελικό προορισμό τους. Για να επιτευχθεί αυτό, θα χρησιμοποιηθεί ο όρος του “edge computing” το οποίο θα διαδραματίσει κρίσιμο ρόλο στην επεξεργασία σε πραγματικό χρόνο των δεδομένων που συλλέγονται. Επιπρόσθετα η μετάδοση δεδομένων μεταξύ των τριών αυτών τεχνολογιών θα επιτυγχάνεται με την σύνδεση τους μέσω του 5G δικτύου. Έτσι, ο οδηγός θα είναι ενήμερος και θα μπορεί να προγραμματίσει την διαδρομή του αποφεύγοντας την κυκλοφοριακή συμφόρηση αλλά και «απελευθερώνοντας» τον δρόμο από επιπλέον κυκλοφορία.

Όπως κάθε τεχνολογία, έτσι και η ανάπτυξη μιας εφαρμογής με κινητούς κόμβους και real time ενημέρωση αντιμετωπίζει επίσης κάποιες δυσκολίες. Αρχικά, έχουμε το πρόβλημα του latency που μπορεί να προκληθεί από θέματα καθυστέρησης επεξεργασίας και μεταφοράς δεδομένων από το drone στο σταθμό βάσης και στη συνέχεια στο νέφος. Αυτό επηρεάζει την επεξεργασία και ανάλυση των δεδομένων σε πραγματικό χρόνο. Παράλληλα με το πιο πάνω πρόβλημα, υπάρχει η πρόκληση του συγχρονισμού των δεδομένων, το οποίο είναι η διασφάλιση της συνέπειας και της ακεραιότητας των δεδομένων σε όλα τα services. Πρέπει να συνεργάζονται απρόσκοπτα για να διασφαλίζεται ο σωστός συγχρονισμός των δεδομένων. Η έγκαιρη αποστολή των δεδομένων και η ακριβής καταμέτρηση των οχημάτων πρέπει να γίνονται αμέσως για real-time ενημέρωση. Επίσης, ακόμα μια πρόκληση είναι η επεκτασιμότητα του συστήματος ώστε να μπορεί να υποδεχθεί μεγάλο αριθμό τόσο drones όσο και base stations για την διαχείριση μεγαλύτερου όγκου κίνησης και μεταφοράς δεδομένων. Ακόμη, έχουμε την δυσκολία από την φύση των drones, με την μικρή διάρκεια ζωής της μπαταρίας τους με τον περιορισμό του χρόνου λειτουργίας και τον όγκο των δεδομένων που μπορούν να μεταδώσουν που αυτό φέρνει ως απαραίτητες προϋποθέσεις την αποτελεσματική χρήση των πόρων και την βελτιστοποίηση του κώδικα. Τέλος, έχουμε το κόστος το οποίο για να υλοποιηθεί η πιο πάνω ιδέα θα πρέπει να αποκτηθεί ακριβώς εξοπλισμός, hardware, αλλά επίσης θα ξοδευτεί πάρα πολλής χρόνος για τις δοκιμές.

1.3 Edge Computing, 5G και emualators για βελτιωμένα συστήματα

Το Edge Computing και το δίκτυο 5G είναι δύο ισχυρές τεχνολογίες που μπορούν να αλληλοσυμπληρωθούν με διάφορους τρόπους και να προσφέρουν πολλά οφέλη. Με το edge computing, η επεξεργασία δεδομένων και οι υπολογισμοί μπορούν να γίνουν στην άκρη του δικτύου, πιο κοντά στην πηγή των δεδομένων, μειώνοντας την καθυστέρηση και επιτρέποντας την επεξεργασία δεδομένων σε πραγματικό χρόνο. Το edge computing, μπορεί να επιτρέψει την ταχύτερη επεξεργασία των εικόνων που καταγράφονται από το drone, η οποία είναι χρήσιμη για την παρακολούθηση σε πραγματικό χρόνο. Το δίκτυο 5G μπορεί να παρέχει συνδεσιμότητα υψηλής ταχύτητας και χαμηλής καθυστέρησης μεταξύ των στοιχείων του συστήματος, επιτρέποντας ταχύτερη και αποτελεσματικότερη επικοινωνία. Το base station, μπορεί να μετρήσει τον αριθμό των οχημάτων σε πραγματικό χρόνο και με τη βοήθεια του δικτύου 5G, να στείλει γρήγορα τα αποτελέσματα στο cloud, το οποίο στη συνέχεια χρησιμοποιώντας το edge computing για να ενημερώσει τους οδηγούς, σχετικά με τις συνθήκες κυκλοφορίας σε πραγματικό χρόνο, τους επιτρέπει να προβούν στις απαραίτητες ενέργειες, όπως η αλλαγή διαδρομής ή ταχύτητας. Μαζί, το edge computing και το δίκτυο 5G, μπορούν να βοηθήσουν στη βελτίωση της απόδοσης και της αποδοτικότητας του προγράμματος, επιτρέποντας την επεξεργασία δεδομένων σε πραγματικό χρόνο, την ταχύτερη επικοινωνία και την δημιουργία πιο αποδοτικών συστημάτων.

Ο emulator είναι ένα χρήσιμο εργαλείο για σκοπούς δοκιμής και ανάπτυξης. Με τη χρήση του, δημιουργούμε και δοκιμάζουμε λογισμικό για πολλαπλές πλατφόρμες χωρίς να χρειάζεται η αγορά και η συντήρηση φυσικών συσκευών. Επίσης γι’ αυτό ακριβώς το λόγο είναι ιδιαίτερα χρήσιμο όταν αναπτύσσετε μια πλατφόρμα. Επιπλέον, οι εξομοιωτές μπορούν να παρέχουν ένα ελεγχόμενο περιβάλλον δοκιμών για λογισμικό και υλικό, επιτρέποντας την αναπαραγωγή σφαλμάτων και δοκιμή διορθώσεων σε ελεγχόμενο περιβάλλον. Αυτό, μπορεί να βοηθήσει να διασφαλιστεί ότι το λογισμικό είναι σταθερό και λειτουργεί όπως αναμένεται με τις φυσικές συνθήκες βελτιώνοντας την ποιότητα του τελικού προϊόντος. Τέλος, μας προσφέρει την εξοικονόμηση χρόνου και την αποφυγή επιπλέον ταλαιπωρίας ή άσκοπης μεταφοράς, όχι μόνο εμάς αλλά και του αναγκαίου εξοπλισμού, αφού δεν χρειάζεται η αυτοπροσώπως παρουσία, ούτε και οι ιδανικές

συνθήκες για την δοκιμή των απαιτούμενων πειραμάτων από την στιγμή που μπορούν να προσομοιωθούν όλα στο σύστημα.

Ως κομμάτι της έρευνας, υλοποιήθηκε μια παραμετροποιήσιμη μελέτη περίπτωσης εφαρμογής (use-case application). Αναπτύχθηκε κώδικας για microservice του drone αλλά και για microservice του basestation το οποίο θα λαμβάνει εικόνες από το drone και θα τις επεξεργάζεται στο infrastructure του με την χρήση ML models για να καταμετρά τον αριθμό των οχημάτων σε κάθε εικόνα ξεχωριστά και μετά θα τα αποστέλλει στο cloud ώστε να μπορεί να ενημερωθεί ο κάθε οδηγός ανάλογα με την διαδρομή που ακολουθεί. Και τα τρία πιο πάνω microservices, είναι εγκατεστημένα σε 5G υποδομές ώστε να μπορούν να επικοινωνούν αλλά και για πειραματικούς σκοπούς. Το κομμάτι του cloud δεν υλοποιήθηκε κι είναι απλά μια θεωρία.

Η αξιολόγηση των επιδόσεων της εφαρμογής περιλαμβάνει τη διεξαγωγή διαφόρων πειραμάτων σε ένα προσομοιωμένο περιβάλλον. Τα πειράματα αυτά, αποσκοπούν στη μέτρηση διαφόρων πτυχών του προγράμματος και στην αξιολόγηση της αποτελεσματικότητάς του. Πρώτον, η χρήση time variables επιτρέπει τον προσδιορισμό της ακριβούς χρονικής διάρκειας που χρειάζεται κάθε microservice για να ολοκληρώσει την αντίστοιχη εργασία της στο πλαίσιο ολόκληρου του προγράμματος αλλά και τον συνολικό χρόνο του προγράμματος να ολοκληρώσει μία εκτέλεση. Επιπλέον, με την ανάλυση των time variables, είναι δυνατή η μέτρηση της καθυστέρησης του προγράμματος, η οποία βοηθά στην αξιολόγηση της απόκρισης και της ταχύτητας επεξεργασίας δεδομένων.
Επιπλέον, στο πρόγραμμα ενσωματώθηκε η ευελιξία επιλογής μεταξύ δύο διαφορετικών αλγορίθμων ML models κατά τη διάρκεια της εκτέλεσης του. Αυτό το χαρακτηριστικό διευκολύνει την αξιολόγηση τόσο της ακρίβειας όσο και της καθυστέρησης του προγράμματος για κάθε αλγόριθμο. Προκειμένου να βελτιστοποιηθεί η μετάδοση των δεδομένων, το πείραμα περιλαμβάνει επίσης τη μέτρηση της απόστασης δικτύου μεταξύ του drone και του σταθμού βάσης. Με την εύρεση της βέλτιστης απόστασης για τη μετάδοση δεδομένων, μπορεί να ενισχυθεί η αποτελεσματικότητα της διαδικασίας επικοινωνίας.
Για να αυξηθεί η πολυπλοκότητα και ο όγκος των δεδομένων, στο πείραμα προστέθηκαν περισσότερα από ένα αντίγραφα drone. Αυτό επιτρέπει την αξιολόγηση της

επεκτασιμότητας του προγράμματος και της ικανότητάς του να διαχειρίζεται μεγαλύτερους όγκους δεδομένων σε ένα πραγματικό σενάριο.

Καθ' όλη τη διάρκεια των πειραμάτων, η χρήση του δικτύου παρακολουθείτε συνεχώς σε κάθε microservice. Αυτό, παρέχει πολύτιμες πληροφορίες σχετικά με τη χρήση των πόρων δικτύου σε διάφορα στάδια της εκτέλεσης του προγράμματος. Επιπρόσθετα, γίνετε έλεγχος της χρήσης και της απόδοσης του υλικού, όπως η μνήμη RAM και η CPU για το κάθε microservice ξεχωριστά αλλά και συνολικά του προγράμματος. Ο έλεγχος αυτός, είναι απαραίτητος για τη διασφάλιση της βέλτιστης λειτουργίας του συστήματος και τον εντοπισμό πιθανών σημείων συμφόρησης ή προβλημάτων απόδοσης. Επιπλέον, στο τέλος της αξιολόγησης, υπολογίστηκε το πραγματικό κόστος της αποστολής των δεδομένων μέσω του δικτύου, λαμβάνοντας υπόψη παράγοντες όπως ο όγκος των δεδομένων και η χρήση του δικτύου. Η αξιολόγηση αυτή βοηθά στην κατανόηση των οικονομικών επιπτώσεων και της σκοπιμότητας ανάπτυξης του προγράμματος σε ένα πραγματικό περιβάλλον.

Κεφάλαιο 2

Related Work

Προκειμένου να αξιοποιηθεί η υπάρχουσα γνώση και να συνεισφερθεί στον τομέα, είναι απαραίτητο να επανεξεταστεί και να αναλυθεί η σχετική βιβλιογραφία. Η παρούσα ενότητα, παρουσιάζει μια επισκόπηση των βασικών μελετών και ερευνών που έχουν διεξαχθεί στον τομέα του benchmarking και evaluation. Με την εξέταση των προηγούμενων ερευνών στον τομέα αυτό, μπορούμε να εντοπίσουμε κενά, προκλήσεις και ευκαιρίες που θα ενημερώσουν και θα καθοδηγήσουν την παρών έρευνα.

Ο υπολογισμός άκρων έχει αναδειχθεί ως μια πολυ υποσχόμενη τεχνολογία, επιτρέποντας την αποτελεσματική επεξεργασία και ανάλυση δεδομένων πιο κοντά στην άκρη του δικτύου. Οι A.Edenbrandt, Hong, C. H., & Varghese, B. [1] παρουσίασαν ένα πλαίσιο που ενσωματώνει τον υπολογισμό στην άκρη, αποδεικνύοντας τα οφέλη της μειωμένης καθυστέρησης και του βελτιωμένου χρόνου απόκρισης στη διαχείριση της ανάλυσης δεδομένων σε πραγματικό χρόνο.

Η χρήση των δοχείων Docker σε περιβάλλον συγκριτικής αξιολόγησης για την αξιολόγηση της απόδοσης, έχει κερδίσει σημαντική προσοχή σε πρόσφατη έρευνα. Η προσέγγιση όπως περιγράφεται και από το έγγραφο “DocLite: A Docker-Based Lightweight Cloud Benchmarking Tool”, προσφέρει πλεονεκτήματα όπως η συνέπεια, η απλούστευση και μια ταχύτερη και αυτοματοποιημένη διαδικασία συγκριτικής αξιολόγησης [10].

Το Fogify [8], είναι ένα άλλο ισχυρό εργαλείο που επιτρέπει τη ρεαλιστική δοκιμή και αξιολόγηση εφαρμογών υπολογισμού ομίχλης. Αντιμετωπίζει την ανάγκη προσομοίωσης περιβαλλόντων ομίχλης για την αξιολόγηση των επιδόσεων και της συμπεριφοράς αυτών των εφαρμογών υπό διάφορες συνθήκες. Το Fogify, όπως και το MockFog [3], εκμεταλλεύονται τους κατανεμημένους πόρους του νέφους για την εκτέλεση πειραμάτων

μεγάλης κλίμακας με πολλούς κεντρικούς υπολογιστές. Αυτό επιτυγχάνεται αφού και τα δυο εστιάζουν στην επεκτασιμότητα και τις ad-hoc αλλαγές στο χρόνο εκτέλεσης.

Με την έλευση των δικτύων 5G, αναδύθηκε η έννοια του network slicing, που επιτρέπει τη δημιουργία εικονικών τμημάτων δικτύου προσαρμοσμένων σε συγκεκριμένες απαιτήσεις εφαρμογών. Ένα εργαλείο, γνωστό ως 5G-Slicer έχει αναπτυχθεί για την ακριβή προσομοίωση αυτών των συνθηκών και συμπεριφορών του δικτύου, επιτρέποντας την αξιολόγηση των επιδόσεων, της επεκτασιμότητας και της ποιότητας υπηρεσιών σε ένα προσομοιωμένο περιβάλλον δικτύου 5G. Το εργαλείο αυτό, όπως αναφέρεται και στο άρθρο “5G-Slicer: An emulator for mobile IoT applications deployed over 5G network slices” [9], παρέχει μια πολύτιμη πλατφόρμα για τη δοκιμή και τη βελτιστοποίηση εφαρμογών πριν από την ανάπτυξη σε δίκτυα 5G σε πραγματικό κόσμο. Επιπλέον, αρκετές μελέτες συμβάλλουν στην κατανόηση του fog computing, της διαχείρισης πόρων, της συγκριτικής αξιολόγησης συσκευών και της αξιολόγησης διαδραστικών εφαρμογών σε υπολογιστές ακμής. Μεταξύ αυτών, περιλαμβάνονται το "Mock-Fog: Emulating Fog Computing Infrastructure in the Cloud", που παρουσιάζει το Mock-Fog [3], το οποίο αναφέρθηκε και πιο πάνω, ένα πλαίσιο για την εξομοίωση υποδομών υπολογιστικής ομίχλης σε περιβάλλον υπολογιστικού νέφους, και το "Resource Management in Fog/Edge Computing" [4] που παρέχει μια επισκόπηση της διαχείρισης πόρων στην ομίχλη και την υπολογιστική ακμής, τονίζοντας τα αρχιτεκτονικά μοντέλα και την ενσωμάτωση με τις αναδυόμενες τεχνολογίες. Επιπλέον, το "Benchmarking Leading-edge Mobile Devices for Data-intensive Distributed Mobile Cloud Applications" επικεντρώνεται στην αξιολόγηση των επιδόσεων των κινητών συσκευών, σε εργασίες έντασης δεδομένων, λαμβάνοντας υπόψη παράγοντες όπως η επεξεργαστική ισχύς, η κατανάλωση ενέργειας και η απόδοση του δικτύου [7].

Επιπρόσθετα, για να καταστεί δυνατή η πιο ρεαλιστική δοκιμή των διαμορφώσεων του δικτύου, έχουν προταθεί από τον ακαδημαϊκό χώρο αρκετοί εξομοιωτές ομίχλης. Οι FogBed [2], EmuFog [6] και EmuEdge [11] επεκτείνουν αξιοσημείωτους εξομοιωτές δικτύων, όπως το MiniNet [5], ώστε να υποστηρίζουν ετερογενείς πόρους και δίκτυα ομίχλης. Αυτοί οι εξομοιωτές παρέχουν πολύτιμες πλατφόρμες για την προσομοίωση και την αξιολόγηση περιβαλλόντων ομίχλης, επιτρέποντας στους ερευνητές και τους προγραμματιστές να αξιολογούν την απόδοση και τη συμπεριφορά των εφαρμογών υπολογισμού ομίχλης σε ελεγχόμενες και αναπαραγώγιμες ρυθμίσεις.

Κεφάλαιο 3

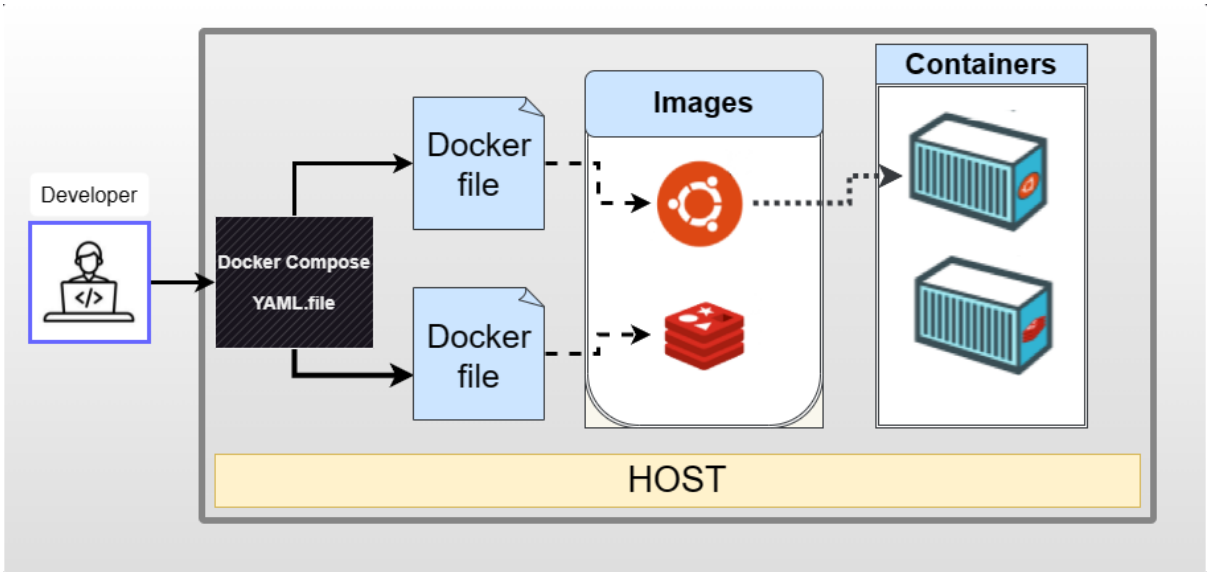
Background Technologies

3.1 Docker – Docker Compose

Το Docker, είναι μια πλατφόρμα ανοικτού κώδικα που προσφέρει μια σειρά από λειτουργίες για την ανάπτυξη, την αποστολή και την εκτέλεση εφαρμογών, επιτρέποντας έτσι την δημιουργία απομονωμένων περιβαλλόντων με συνέπεια και αξιοπιστία, γνωστά ως containers. Οι εφαρμογές αναλύονται σε μικρότερα, πιο διαχειρίσιμα στοιχεία που μπορούν εύκολα να ενημερώνονται και να κλιμακώνονται ανεξάρτητα. Αυτό, βοηθά στη μείωση της πολυπλοκότητας των εφαρμογών και τις κάνει πιο αρθρωτές, διευκολύνοντας τη συντήρηση και την κλιμάκωση της εφαρμογής με την πάροδο του χρόνου. Το Docker απλοποιεί τη διαδικασία διαμόρφωσης και διατήρησης περιβαλλόντων ανάπτυξης, δοκιμής και παραγωγής. Επιπλέον, τα Docker containers είναι ανεξάρτητα από πλατφόρμες, καθιστώντας εύκολη τη μετακίνηση εφαρμογών μεταξύ διαφορετικών

λειτουργικών συστημάτων ή παρόχων cloud. Αυτή η φορητότητα επιτρέπει επίσης βελτιωμένη επεκτασιμότητα, καθώς η προσθήκη ή αφαίρεση των containers γίνετε γρήγορα για να προσαρμόζονται στην αντιμετώπιση αλλαγών στην κίνηση ή τη ζήτηση της εφαρμογής. Παράλληλα, το επίπεδο απομόνωσης του Docker μπορεί να ενισχύσει την ασφάλεια μειώνοντας τον κίνδυνο επιθέσεων στο υποκείμενο λειτουργικό σύστημα. Έπειτα, η ελαφριά αρχιτεκτονική του Docker παρέχει μια οικονομικά αποδοτική εναλλακτική λύση επιτρέποντάς περισσότερη χρήση από τη χωρητικότητα του διακομιστή. Τέλος, το Docker χρησιμοποιεί μια αρχιτεκτονική client-server, με τον Docker client να επικοινωνεί με το Docker daemon μέσω ενός REST API μέσω υποδοχών UNIX ή μιας διασύνδεσης δικτύου, παρέχοντας μια επεκτάσιμη και ευέλικτη λύση για τη διαχείριση εφαρμογών.

Για να γίνει πλήρως κατανοητό το πώς λειτουργούν τα Docker Containers, είναι βοηθητικό να γίνει μια σύγκριση με τις παραδοσιακές εικονικές μηχανές (VM). Τα Docker Containers και οι εικονικές μηχανές (VM) λειτουργούν σε διαφορετικά επίπεδα εικονικοποίησης, οδηγώντας σε διαφορετικές προσεγγίσεις χρήσης πόρων και ανάπτυξης. Ενώ οι εικονικές μηχανές (VM) εξομοιώνουν ένα ολόκληρο λειτουργικό σύστημα (OS) σε έναν φυσικό διακομιστή απαιτώντας αποκλειστικούς πόρους συστήματος, όπως CPU, μνήμη και αποθήκευση. Τα δοχεία Docker, λειτουργούν σε επίπεδο λειτουργικού συστήματος, χρησιμοποιώντας και απευθείας τον πυρήνα του κεντρικού λειτουργικού συστήματος μοιράζοντας τους πόρους του. Επίσης, η διαδικασία ανάπτυξης διαφέρει μεταξύ των δύο τεχνολογιών. Τα VM συνήθως περιλαμβάνουν τη δημιουργία και τη διαμόρφωση μιας πλήρους εικόνας λειτουργικού συστήματος, η οποία μπορεί να είναι χρονοβόρα και να απαιτεί πολλούς πόρους. Αντιθέτως, τα Docker Containers χρησιμοποιούν ελαφριές, φορητές εικόνες που περιλαμβάνουν μόνο την εφαρμογή και τις συγκεκριμένες εξαρτήσεις της, απομονώνοντάς τα από την υποκείμενη υποδομή. Ωστόσο, σε αντίθεση με τα VMs, τα οποία απαιτούν έναν hypervisor για τη διαχείριση πολλαπλών περιπτώσεων, τα Docker containers αξιοποιούν τη μηχανή Docker για να παρέχουν ένα απομονωμένο περιβάλλον για κάθε εφαρμογή.



Σχήμα 3.1

Το Docker Compose είναι ένα ισχυρό εργαλείο για τον ορισμό και την εκτέλεση εφαρμογών Docker με πολλαπλά containers. Με το Docker Compose, χρησιμοποιείτε ένα μόνο αρχείο YAML για την ρύθμιση και την εκκίνηση όλων των υπηρεσιών που χρειάζονται, απλοποιώντας την εγκατάσταση και τη διαχείριση της εφαρμογής. Αυτό είναι ιδιαίτερα χρήσιμο για μεγαλύτερες εφαρμογές, όπου πολλαπλές υπηρεσίες πρέπει να οριστούν και να συντονιστούν για να λειτουργήσουν ορθά. Το Docker Compose έχει την ικανότητά να λειτουργεί σε όλα τα περιβάλλοντα, συμπεριλαμβανομένων της παραγωγής, του staging, της ανάπτυξης και των δοκιμών. Αυτό δίνει την δυνατότητα για δημιουργία μιας εφαρμογής που μπορεί εύκολα να μετακινηθεί μεταξύ διαφορετικών περιβαλλόντων, χωρίς σημαντικές αλλαγές. Επιπρόσθετα, διευκολύνει την αύξηση ή μείωση της κλίμακας των εφαρμογών ανάλογα με τις ανάγκες. Είναι εξαιρετικά πρακτικό, καθώς παρέχει τη δυνατότητα γρήγορης προσθήκης ή αφαίρεσης των containers, εξασφαλίζοντας έτσι την απρόσκοπτη λειτουργία της εφαρμογής. Συν των προηγουμένων, επειδή το Docker Compose αυτοματοποιεί τη διαδικασία εκκίνησης και παύσης των containers, μειώνει την πολυπλοκότητα της διαχείρισης εφαρμογών με πολλαπλά containers. Εν τέλει, το Docker Compose μπορεί να βελτιώσει τη φορητότητα και την αξιοπιστία των εφαρμογών Docker. Ορίζοντας την εφαρμογή ως ένα σύνολο υπηρεσιών και εξαρτήσεων σε ένα ενιαίο αρχείο YAML, το Docker Compose διευκολύνει τη μετακίνηση της εφαρμογής μεταξύ διαφορετικών περιβαλλόντων. Αυτό ελαχιστοποιεί τον κίνδυνο πιθανών σφαλμάτων διαμόρφωσης ή άλλων προβλημάτων που μπορεί να προκύψουν κατά τη μετακίνηση.

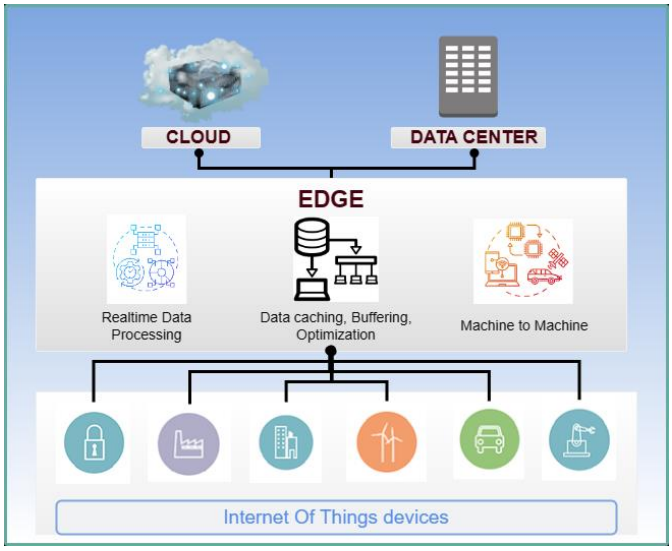
3.2 Edge Computing & Fog Computing

Το Edge Computing και το Fog Computing είναι δύο distributed computing μοντέλα που φέρνουν επανάσταση στον τρόπο με τον οποίο επεξεργαζόμαστε και αποθηκεύουμε δεδομένα. Αν και έχουν διακριτά χαρακτηριστικά, και τα δύο στοχεύουν να φέρουν τους υπολογιστικούς πόρους πιο κοντά στην άκρη του δικτύου, προσφέροντας βελτιωμένη απόδοση, επεξεργασία σε πραγματικό χρόνο και ενισχυμένη ασφάλεια.

Το edge computing εστιάζει στην επεξεργασία δεδομένων στην πηγή, ή κοντά στην πηγή, μειώνοντας την καθυστέρηση και επιτρέποντας ταχύτερους χρόνους απόκρισης. Αξιοποιεί ένα δίκτυο συσκευών, όπως έξυπνα αυτοκίνητα, τηλέφωνα, μη επανδρωμένα αεροσκάφη και σταθμούς βάσης, για την εκτέλεση τοπικών υπολογισμών. Αυτό το μοντέλο είναι ιδιαίτερα επωφελές σε σενάρια όπου απαιτούνται αντιδράσεις υψηλής ταχύτητας και η στήριξη σε μια κεντρική υποδομή cloud, θα εισήγαγε σημαντικές καθυστερήσεις. Με την έλευση της τεχνολογίας 5G, ο υπολογισμός ακμών αποκτά ακόμη μεγαλύτερη σημασία, επιτρέποντας συναρπαστικές εφαρμογές όπως οι έξυπνες πόλεις και τα αυτόνομα μη επανδρωμένα αεροσκάφη. Μειώνοντας την καθυστέρηση, ενισχύοντας την αποδοτικότητα του εύρους ζώνης και αυξάνοντας την ασφάλεια, το edge computing ανοίγει το δρόμο για διαδραστικές εμπειρίες, ανάλυση δεδομένων σε πραγματικό χρόνο και οικονομικά αποδοτικές λύσεις για εργασίες έντασης δεδομένων.

Από την άλλη πλευρά, το fog computing επεκτείνει τις δυνατότητες του cloud computing στην άκρη του δικτύου. Αντιμετωπίζει τους περιορισμούς του παραδοσιακού μοντέλου cloud, φέρνοντας τους υπολογισμούς, την αποθήκευση και τη δικτύωση πιο κοντά στις συσκευές και τους χρήστες. Με την κατανομή των υπολογιστικών πόρων στα στρώματα cloud, fog και edge, το fog computing μειώνει την ανάγκη για εκτεταμένη μετάδοση δεδομένων και ξεπερνά τις προκλήσεις καθυστέρησης που σχετίζονται με τις cloud-centric αρχιτεκτονικές. Με την fog computing, η επεξεργασία και η αποθήκευση δεδομένων γίνεται πιο κοντά στην πηγή, εξασφαλίζοντας χαμηλότερη καθυστέρηση, αυξημένη ασφάλεια δεδομένων και βελτιωμένη ιδιωτικότητα. Επιπλέον, το fog computing προάγει την ανθεκτικότητα μειώνοντας την εξάρτηση από κεντρικές υπηρεσίες cloud, καθιστώντας το λιγότερο ευαίσθητο σε διακοπές δικτύου και μετριάζοντας το κόστος μετάδοσης δεδομένων. Αυτή η κατανεμημένη υπολογιστική προσέγγιση, ανοίγει νέες ευκαιρίες για επεξεργασία σε πραγματικό χρόνο, εφαρμογές χαμηλής καθυστέρησης και αποτελεσματική χρήση των πόρων.

Εν κατακλείδι, η υπολογιστική ακμών και το fog computing, αντιπροσωπεύουν σημαντικές προόδους στην κατανεμημένη υπολογιστική. Ενώ η υπολογιστική ακμής δίνει έμφαση στην τοπική επεξεργασία και την ανταπόκριση σε πραγματικό χρόνο, η υπολογιστική ομίχλης επεκτείνει τις δυνατότητες της υπολογιστικής νέφους στην άκρη του δικτύου, μειώνοντας την καθυστέρηση και βελτιώνοντας την ασφάλεια των δεδομένων. Αυτά τα μοντέλα επιτρέπουν ένα ευρύ φάσμα εφαρμογών και περιπτώσεων χρήσης, από εργασίες ευαίσθητες στο χρόνο έως διαδικασίες έντασης δεδομένων, και παρέχουν τη βάση για μετασχηματιστικές τεχνολογίες όπως οι έξυπνες πόλεις και τα αυτόνομα συστήματα.



Σχήμα 3.2

3.3 Rest API / FastAPI / Client - Server Architecture

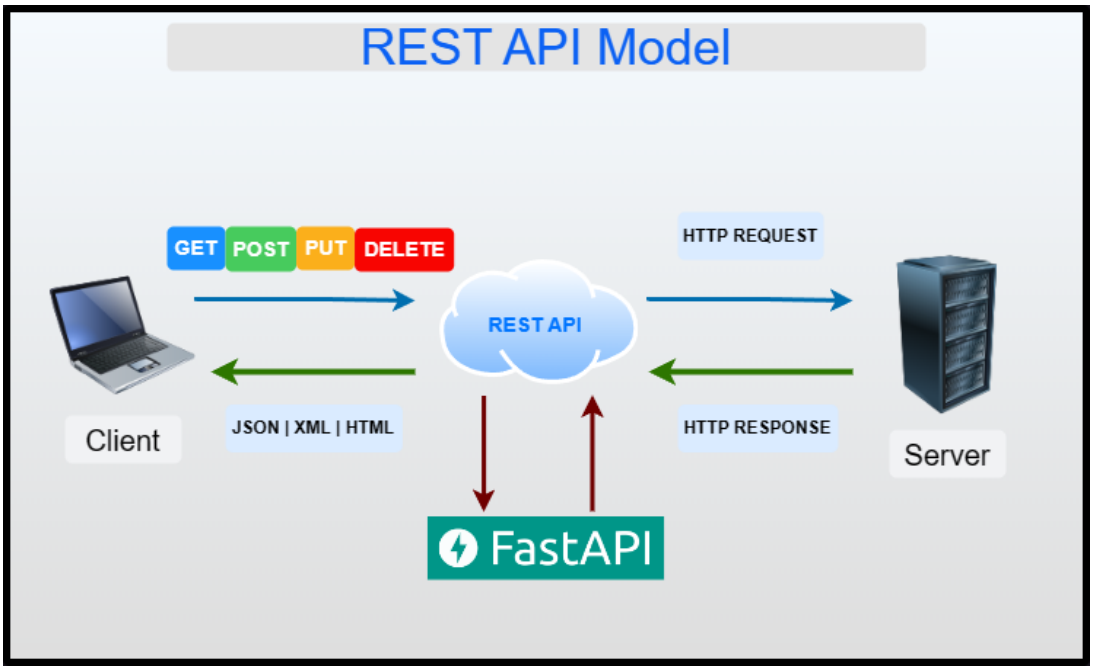
Τα RESTful APIs (Representational State Transfer) είναι μια δημοφιλής επιλογή για τη δυνατότητα επικοινωνίας μεταξύ συστημάτων λογισμικού με τη χρήση αιτημάτων HTTP, όπως GET, PUT, POST και DELETE. Διευκολύνουν την απομακρυσμένη ανταλλαγή δεδομένων σε συγκεκριμένη μορφή, καθιστώντας τα, ελαφριά, αποδοτικά και χωρίς κατάσταση. Ένα από τα βασικά πλεονεκτήματα των RESTful APIs είναι η εύκολη δυνατότητα προσωρινής αποθήκευσης, μειώνοντας το φορτίο του διακομιστή και βελτιώνοντας την απόδοση του πελάτη. Με τυποποιημένες μεθόδους HTTP και κωδικούς απόκρισης, τα RESTful APIs είναι φιλικά προς το χρήστη και ευρέως εφαρμόσιμα, ακόμη και σε σενάρια περιορισμένων πόρων, όπως οι εφαρμογές για κινητά και drone.
Το FastAPI, ένα διαδικτυακό πλαίσιο βασισμένο στην Python, είναι γνωστό για την ανάπτυξη μικρουπηρεσιών υψηλής απόδοσης και συστημάτων πραγματικού χρόνου. Ο

ελαφρύς και αποδοτικός σχεδιασμός του εξασφαλίζει γρήγορους χρόνους απόκρισης και χαμηλό overhead, καθιστώντας το ιδανικό για τη διαχείριση μεγάλου όγκου αιτημάτων και επιτρέποντας την εύκολη ανάπτυξη και επεκτασιμότητα. Το FastAPI υποστηρίζει ασύγχρονο προγραμματισμό, καθιστώντας το κατάλληλο για επεξεργασία δεδομένων σε πραγματικό χρόνο. Παράλληλα, απλοποιεί τη δημιουργία API με ενσωματωμένη υποστήριξη για OpenAPI και JSON Schema, επιτρέποντας καλά τεκμηριωμένες και δομημένα microservices. Το FastAPI, προσφέρει εργαλεία δοκιμών και αποσφαλμάτωσης, συμπεριλαμβανομένης της αυτόματης δημιουργίας τεκμηρίωσης API και της υποστήριξης διαδραστικής αποσφαλμάτωσης. Η συμβατότητά του με δημοφιλείς βιβλιοθήκες Python, όπως οι NumPy και TensorFlow, το καθιστά εξαιρετική επιλογή για εφαρμογές επιστήμης δεδομένων, εξασφαλίζοντας αποτελεσματική επεξεργασία δεδομένων. Η φιλική προς τον χρήστη διεπαφή του FastAPI, σε συνδυασμό με τις δυνατότητες γρήγορης ανάπτυξης, επιτρέπουν εύκολα την κατασκευή πρωτοτύπων και τον πειραματισμό.

Η αρχιτεκτονική πελάτη-εξυπηρετητή, χρησιμεύει ως θεμελιώδες μοντέλο για κατανεμημένες εφαρμογές. Περιλαμβάνει συστήματα-πελάτες που ζητούν υπηρεσίες ή πόρους από συστήματα διακομιστών. Οι πελάτες ξεκινούν τα αιτήματα και οι διακομιστές παρέχουν τις ζητούμενες υπηρεσίες μέσω ενός δικτύου, είτε πρόκειται για τοπικό δίκτυο (LAN), είτε για δίκτυο ευρείας περιοχής (WAN), είτε για το Διαδίκτυο. Αυτή η αρχιτεκτονική ,προωθεί το διαχωρισμό των προβλημάτων, απλοποιώντας το σχεδιασμό, την ανάπτυξη και τη συντήρηση πολύπλοκων συστημάτων. Ενισχύει την επεκτασιμότητα, καθώς οι διακομιστές μπορούν να διεκπεραιώνουν ταυτόχρονα πολλαπλά αιτήματα πελατών. Έπειτα, ενισχύει την ασφάλεια, επιτρέποντας πολιτικές ελέγχου πρόσβασης και ασφαλή κανάλια επικοινωνίας.

Οι αρχιτεκτονικές δύο επιπέδων περιλαμβάνουν ένα ενιαίο σύστημα πελάτη και διακομιστή, ενώ άλλες παραλλαγές εξυπηρετούν την αυξανόμενη πολυπλοκότητα, τις απαιτήσεις κλιμάκωσης και τα επιθυμητά επίπεδα επιδόσεων.

Συνδυάζοντας τα RESTful APIs, το FastAPI και την αρχιτεκτονική πελάτη-εξυπηρετητή, οι προγραμματιστές μπορούν να δημιουργήσουν αποδοτικά, κλιμακούμενα και υψηλής απόδοσης κατανεμημένα συστήματα που καλύπτουν ποικίλες ανάγκες εφαρμογών.

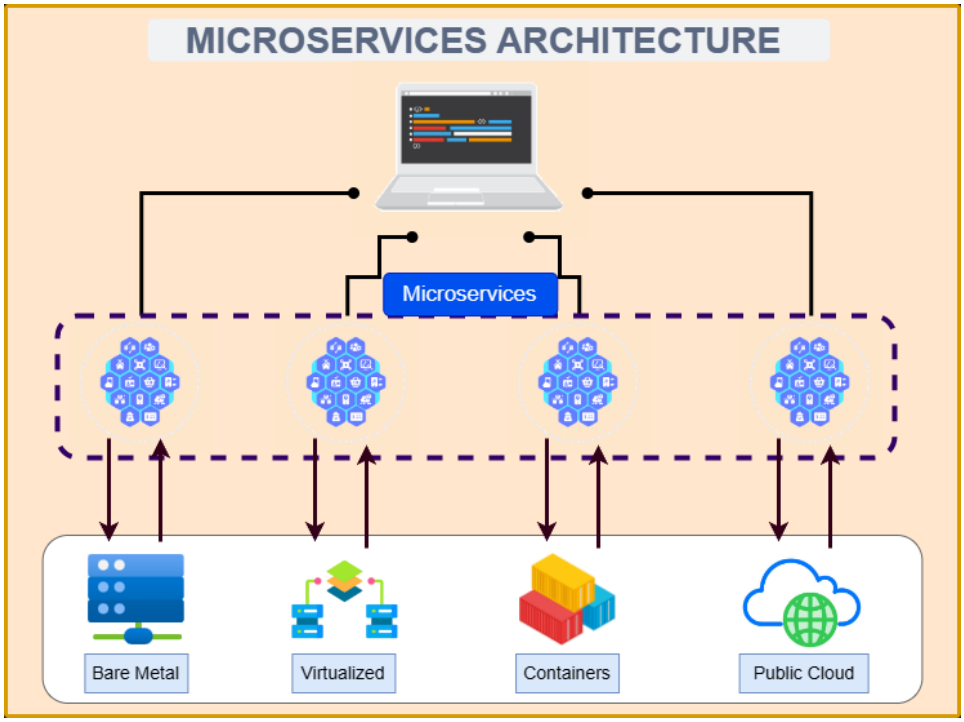


Σχήμα 3.3

3.4 Microservices Architecture

Ένα microservice είναι μια σύγχρονη αρχιτεκτονική ανάπτυξης λογισμικού που προσφέρει λύση στις προκλήσεις που αντιμετωπίζει η μονολιθική αρχιτεκτονική. Δίνει τη δυνατότητα της διάσπασης μιας μεγάλης, σύνθετης εφαρμογής σε μικρότερες, ανεξάρτητες υπηρεσίες με σαφείς αρμοδιότητες που μπορούν εύκολα να διαχειριστούν. Η πιο σημαντική πτυχή των μικρουπηρεσιών είναι η ικανότητά τους να παρέχουν επεκτασιμότητα, ευελιξία και ανθεκτικότητα στην εφαρμογή. Με τα microservices, κάθε υπηρεσία μπορεί να κλιμακωθεί ανεξάρτητα με βάση τις δικές της ανάγκες σε πόρους, επιτρέποντας τη βελτίωση της κλιμάκωσης της συνολικής εφαρμογής. Επιπλέον, τα microservices επιτρέπουν την ανάπτυξη και την ενημέρωση των υπηρεσιών ανεξάρτητα, παρέχοντας μεγαλύτερη ευελιξία και ευκινησία στην ανταπόκριση στις μεταβαλλόμενες προγραμματιστικές ανάγκες. Τα microservices ενισχύουν επίσης την ανθεκτικότητα της εφαρμογής, μειώνοντας τον αντίκτυπο των αποτυχιών και διευκολύνοντας την ανάκαμψη από αυτές. Ταυτόχρονα, εξασφαλίζουν ότι η αποτυχία μιας μονάδας δεν επηρεάζει την λειτουργία της συνολικής εφαρμογής. Επιπλέον, τα microservices διευκολύνουν τη συντήρηση της εφαρμογής, καθώς κάθε υπηρεσία μπορεί να αναπτυχθεί, να δοκιμαστεί και να αναπτυχθεί ανεξάρτητα. Τέλος, η τεχνολογική ποικιλομορφία, επιτρέπει στην χρήση διαφορετικών τεχνολογιών και εργαλείων για διαφορετικές υπηρεσίες, καθώς

κάθε υπηρεσία εκτελείται σε δική της διαδικασία και επικοινωνεί με άλλες υπηρεσίες χρησιμοποιώντας ελαφριά API.



Σχήμα 3.4

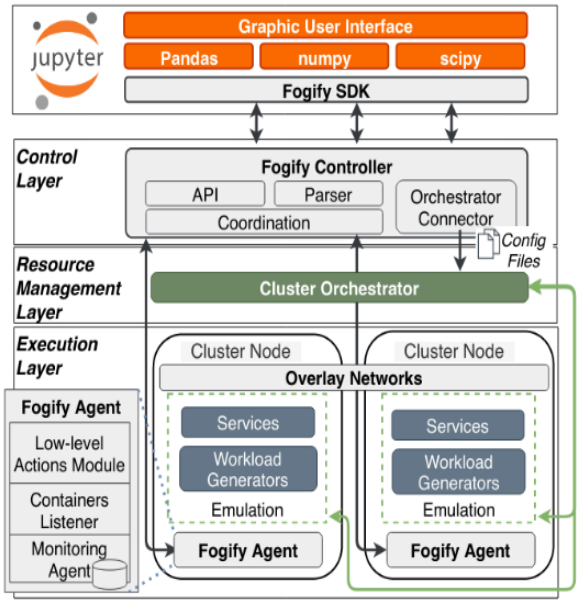
3.5 Emulator

Ο Emulator είναι ένας τύπος λογισμικού ή υλικού που επιτρέπει σε ένα σύστημα υπολογιστή, γνωστό ως κεντρικός υπολογιστής, να μιμείται τη συμπεριφορά, το υλικό και το λογισμικό του άλλου συστήματος ή συσκευής. Αυτό επιτρέπει στο σύστημα υποδοχής να εκτελεί λογισμικό ή να χρησιμοποιεί περιφερειακές συσκευές που έχουν σχεδιαστεί για το φιλοξενούμενο σύστημα. Οι εξομοιωτές, μπορούν να χρησιμοποιηθούν με διάφορους τρόπους στις διαδικασίες ανάπτυξης και δοκιμής λογισμικού επιταχύνοντας τη διαδικασία ανάπτυξης και την δοκιμή του λογισμικού χωρίς να χρειάζεται να αναπτύσσετε κάθε φορά στο φυσικό υλικό. Αυτό έχει τη δυνατότητα να μειώσει τον χρόνο αναμονής για τη διόρθωση σφαλμάτων και τη βελτίωση λειτουργιών. Είναι χρήσιμοι για την δημιουργία και δοκιμή λογισμικού για πολλαπλές πλατφόρμες χωρίς να χρειάζεται η αγορά και η συντήρηση αρκετών φυσικών συσκευών με αποτέλεσμα να είναι μια οικονομικά αποδοτική εναλλακτική λύση. Επιπρόσθετα, οι εξομοιωτές μπορούν να παρέχουν ένα ελεγχόμενο περιβάλλον για τη δοκιμή και

ανάπτυξη λογισμικού και υλικού, την αναπαραγωγή σφαλμάτων και τη διόρθωσή τους σε ελεγχόμενο περιβάλλον. Μπορούν επίσης να χρησιμοποιηθούν για να καταστήσουν το λογισμικό και τα συστήματα προσβάσιμα σε χρήστες που μπορεί να μην έχουν πρόσβαση στο φυσικό υλικό ή λογισμικό, όπως σε απομακρυσμένα ή εικονικά περιβάλλοντα. Τέλος, παρέχει επίσης ένα sandboxed περιβάλλον για την εκτέλεση δυνητικά κακόβουλου ή μη αξιόπιστου λογισμικού, μειώνοντας τον κίνδυνο παραβίασης ενός συστήματος υποδοχής.

3.6 Fogify

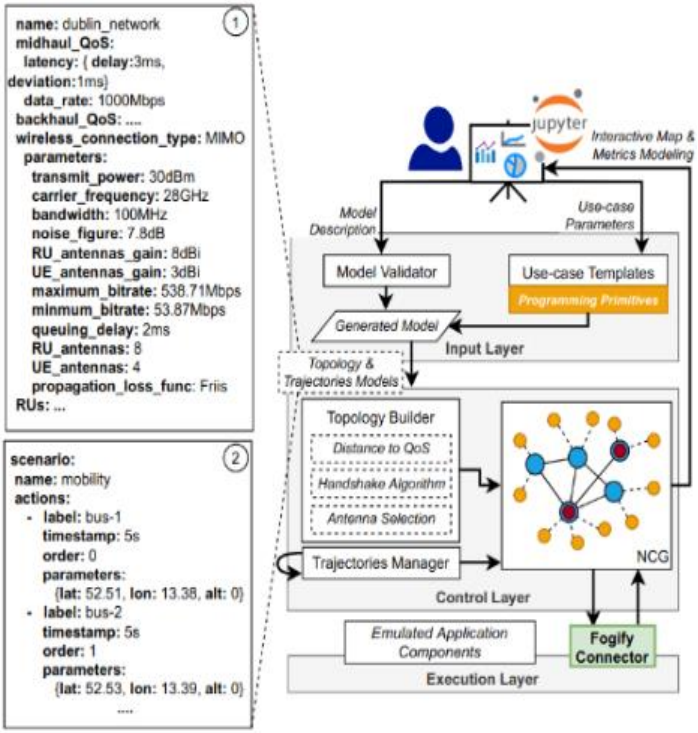
Προχωρώντας, το Fogify είναι ένα framework προσομοίωσης που επιτρέπει την μοντελοποίηση, την ανάπτυξη και τον πειραματισμό με εφαρμογές υπολογισμού ακραίων σημείων χωρίς την ανάγκη φυσικών συσκευών ακραίων σημείων αλλά και δοκιμαστικά fog environments με ρεαλιστική προσομοίωση στον πραγματικό κόσμο. Παρέχει μια σειρά εργαλείων για τη μοντελοποίηση σύνθετων fog topologies που αποτελούνται από ποικίλους πόρους, δυνατότητες δικτύου και κριτήρια ποιότητας υπηρεσιών. Το framework υποστηρίζει επίσης την ανάπτυξη των διαμορφωμένων υπηρεσιών και των περιγραφών υποδομής ως containerized code σε περιβάλλοντα cloud ή τοπικά περιβάλλοντα. Το Fogify προσφέρει εργαλεία για την προσομοίωση σύνθετων fog topologies και επιτρέπει επίσης τον πειραματισμό, την μέτρηση και αξιολόγηση της ανάπτυξης υπηρεσιών προσαρμόζοντας τη διαμόρφωση κατά την εκτέλεση για να δοκιμάσουν διαφορετικά διαμορφώσεις και σενάρια «τι θα γίνει αν». Το framework έχει την ικανότητα να προσομοιώνει πραγματικές συνθήκες edge computing, όπως η καθυστέρηση δικτύου και η αναξιόπιστη συνδεσιμότητα σε ένα ρεαλιστικό περιβάλλον. Επιπλέον, το Fogify εξαλείφει την ανάγκη για φυσικές συσκευές άκρων, μειώνοντας το συνολικό κόστος ανάπτυξης και δοκιμής εφαρμογών υπολογισμού άκρων. Τέλος η επεκτασιμότητα και τα χαρακτηριστικά ασφαλείας του Fogify αποτελούν επίσης σημαντικό κομμάτι του.



Σχήμα 3.6 (Fogify)[8]

3.7 5G - Slicer

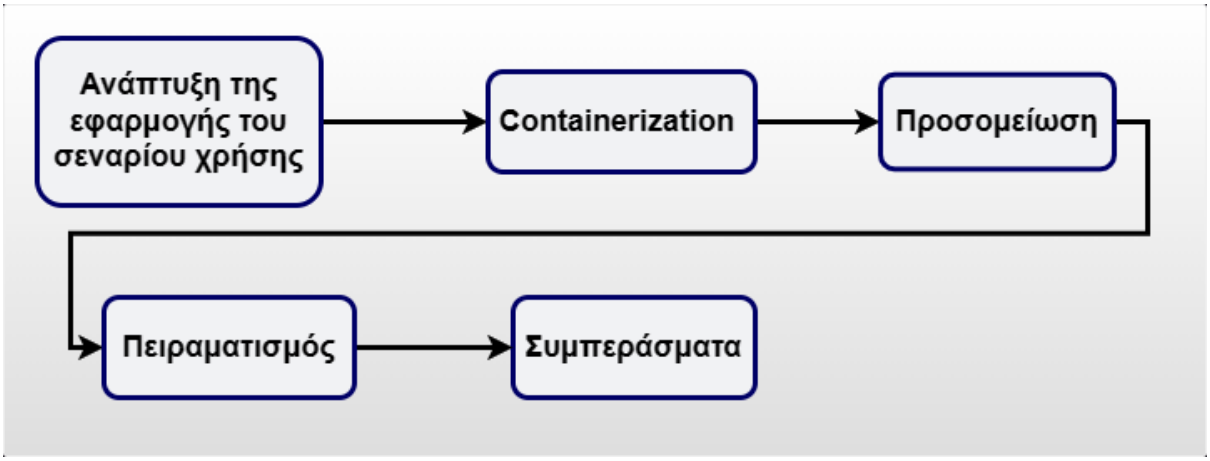
Η επέκταση 5G-Slicer του πλαισίου Fogify παρέχει έναν εξομοιωτή για κινητές εφαρμογές IoT που αναπτύσσονται σε φέτες δικτύου 5G χρησιμοποιώντας την τεχνική του network slicing. Ουσιαστικά, είναι ένα εργαλείο που διευκολύνει τη δημιουργία, παρέχει ευελιξία στην ανάπτυξη και τη διαχείριση τμημάτων δικτύου σε δίκτυα 5G, επιτρέποντας τη δημιουργία πολλαπλών εικονικών δικτύων που μπορούν να προσαρμοστούν ώστε να ανταποκρίνονται στις ειδικές απαιτήσεις διαφορετικών εφαρμογών και υπηρεσιών. Απλοποιεί τη διαδικασία δοκιμών των υπηρεσιών που υποστηρίζονται από το 5G, δημιουργώντας ένα προσομοιωμένο περιβάλλον που μοντελοποιεί radio units, κινητούς κόμβους και διαδρομές, μεταξύ άλλων, επιτρέποντας την δημιουργία και την διαχείριση των δυναμικών φετών δικτύου. Ο εξομοιωτής είναι σε θέση να παρέχει ρεαλιστικό QoS δικτύου μεταβάλλοντας δυναμικά την ισχύ του σήματος κατά την εκτέλεση με αποτέλεσμα να γίνετε η επικύρωση της λειτουργικότητας και της απόδοσης των φετών δικτύου πριν από την ανάπτυξή τους σε περιβάλλον παραγωγής. Περιλαμβάνει επίσης ένα ήδη υλοποιημένο σενάριο για μια εγκατάσταση σε κλίμακα πόλης, επιτρέποντας στους χρήστες να διαμορφώνουν και να δοκιμάζουν εύκολα τις εφαρμογές τους.



Σχήμα 3.7 (5G-Slicer) [9]

Κεφάλαιο 4

Μεθοδολογία



4.1 Ανάπτυξη της εφαρμογής του σεναρίου χρήσης

Η συγκεκριμένη ακολουθία εκτέλεσης προϋποθέτει μία microservice εφαρμογή. Στα πλαίσια της συγκεκριμένης έρευνας υλοποιήθηκε μια παραμετροποιήσιμη μελέτη περίπτωσης εφαρμογής (use-case application), ένας κώδικας που για την υλοποίηση του

ακολουθήθηκε η αρχιτεκτονική των microservices. Ποιο συγκεκριμένα, είναι ένα application το οποίο υποστηρίζει τρία διαφορετικά services. Το microservice του drone, του base station και του cloud. Η επικοινωνία μεταξύ drone – basestation και basestation – cloud επιτυγχάνετε με την χρήση restful API και συγκεκριμένα με το FastAPI framework. Η ροή του προγράμματος καθορίζετε από κάποια environmental variable στο microservice drone και base station τα οποία καθορίζονται είτε από το πρόγραμμα το ίδιο είτε κατά την χρήση των docker containers. Ο όρος "environmental variable" αναφέρεται σε τιμές που αποθηκεύονται στο περιβάλλον του λειτουργικού συστήματος και είναι προσπελάσιμες από διεργασίες που εκτελούνται στο εν λόγω σύστημα, επιτρέποντας τον έλεγχο διαφόρων πτυχών της συμπεριφοράς και της λειτουργικότητάς του. Παρέχει επίσης έναν τρόπο για τη μετάδοση πληροφοριών από το σύστημα στις διεργασίες που εκτελούνται σε αυτό.

Ένα τυπικό σενάριο ξεκινά με την εφαρμογή του drone microservice να αποστέλλει στο base station microservice τις εικόνες που έχει συλλέξει, αφού πρώτα εκτελέσει κάποια αρχική επεξεργασία των εικόνων ώστε να τοποθετηθούν σε tuples και το κάθε tuple με την εικόνα γίνετε append σε μια list η οποία θα σταλεί με HTTP Post Request μέσω μιας σύνδεσης 5G χαμηλής καθυστέρησης. Το base station microservice λαμβάνει τις εικόνες από το drone microservice και εκτελεί καταμέτρηση οχημάτων χρησιμοποιώντας ένα από τις δύο επιλογές αλγορίθμων υπολογιστικής όρασης. Μόλις ολοκληρωθεί η καταμέτρηση οχημάτων, το base station αποστέλλει τα αποτελέσματα στο cloud και αυτό με HTTP Post Request μέσω μιας σύνδεσης 5G χαμηλής καθυστέρησης και απαντά στο drone για την επιτυχία της διαδικασίας. Καταλήγοντας, το cloud microservice λαμβάνει τα αποτελέσματα καταμέτρησης οχημάτων από το base station και τα αποθηκεύει για τυχόν χρήση από άλλες οντότητες ή ίσως στο μέλλον να χρησιμοποιούνται για να ενημερώνει τους οδηγούς με αυτόνομα αυτοκίνητα σχετικά με τις πραγματικές συνθήκες κυκλοφορίας.

4.2 Containerization

Σε αυτό το βήμα ο χρήστης πρέπει να δημιουργήσει τα containerized services της εφαρμογής όπου του προσφέρονται δύο διαφορετικοί μέθοδοι για να το επιτεύξει. Στην δική μας περίπτωση, η πρώτη μέθοδος περιλαμβάνει τη δημιουργία των images κάθε υπηρεσίας ξεχωριστά και στη συνέχεια τη δημιουργία των αντίστοιχων containers μέσω

της γραμμής εντολών. Η δεύτερη μέθοδος εκτέλεσης των builds είναι η εκτέλεση του αρχείου docker-compose.yaml που βρίσκεται στο root του project, το οποίο επιτρέπει την κατασκευή όλων των απαραίτητων υπηρεσιών με μία μόνο εντολή.

Ο προαναφερθέντας κώδικας για τους σκοπούς της έρευνας τοποθετήθηκε μέσα σε containers. Συγκεκριμένα, δημιουργήθηκαν τρία containers ώστε το κάθε διαφορετικό microservice να εισαχθεί σε διαφορετικό container. Η εισαγωγή τους έγινε για το επιπλέον πλεονέκτημα της φορητότητας του προγράμματος αλλά επίσης και για την ενσωμάτωση του όλου προγράμματος στον emulator – 5G Slicer. Η απόφαση αυτή ελήφθη για πειραματικούς σκοπούς αλλά και επειδή οι προσομοιωτές υποδομών Edge συνήθως λειτουργούν με containers.

Για κάθε microservice δημιουργείτε πρώτα το image του, από το δικό του Dockerfile και αφού η διαδικασία γίνει και για τα τρία microservices, με την χρήση του docker-compose.yaml αρχείου δημιουργούμε τα τρία ξεχωριστά containers. Το docker-compose χρησιμοποιείτε για την εύκολη δημιουργία των containers αλλά και για τον καθορισμό των διαφορετικών environmental variables του κάθε microservice. Εν τέλει, προσφέρει την σύνδεση των τριών containers μέσω της δημιουργίας network.

4.3 Προσομοίωση

Η χρήση ενός εξομοιωτή για την εκτέλεση ενός προγράμματος περιλαμβάνει τη δημιουργία ενός εικονικού περιβάλλοντος που μιμείται τη συμπεριφορά των πραγματικών συσκευών. Αυτό μπορεί να επιτευχθεί μέσω της χρήσης εξειδικευμένου λογισμικού που εξομοιώνει τα στοιχεία υλικού και λογισμικού, επιτρέποντας στο πρόγραμμα να εκτελείται σε ένα εικονικό περιβάλλον που αντιπροσωπεύει μια πραγματική συσκευή. Για την χρήση του εξομοιωτή, εγκαταστάθηκε το κατάλληλο λογισμικό και διαμορφώθηκαν οι ρυθμίσεις του εξομοιωτή ώστε να ταιριάζουν με τις συσκευές. Το πρόγραμμα υποβλήθηκε σε δοκιμές και εφαρμόστηκε με τον ίδιο τρόπο που θα ήταν στον πραγματικό κόσμο, χρησιμοποιώντας φυσικές συσκευές.

Το πρόγραμμα εκτελείται όπως αναμένεται στις συσκευές, και δοκιμάζεται υπό διάφορες συνθήκες, συμπεριλαμβανομένων διαφορετικών περιβαλλόντων δικτύου, περισσότερων ή λιγότερων συσκευών και σεναρίων εισόδου/εξόδου. Επίσης παρακολουθείτε η απόδοση του προγράμματος και συλλέγονται δεδομένα σχετικά με τη συμπεριφορά του,

προκειμένου να εντοπιστούν περιοχές προς βελτίωση ή αν αποδίδει όπως αναμένεται στο συγκεκριμένο περιβάλλον.

4.4 Πειραματισμός

Αρχικά το πρόγραμμα ενσωματώνει διάφορες λειτουργίες για τη μέτρηση της καθυστέρησης σε διάφορα στάδια λειτουργίας, με τα timestamps να χρησιμοποιούνται για τον υπολογισμό της συνολικής διάρκειας κάθε λειτουργίας. Επίσης, αφού στο πρόγραμμα υλοποιήθηκαν δύο αλγόριθμοι καταμέτρησης οχημάτων αξιολογούνται επίσης ως προς την ακρίβειά τους, ενώ λαμβάνονται υπόψη οι πιθανές καθυστερήσεις εξυπηρέτησης στο service του base station. Επιπλέον, διεξάγεται πειραματική ανάλυση για τον προσδιορισμό της επίδρασης της απόστασης μεταξύ drone και σταθμού βάσης στην απόδοση, ενώ παρακολουθούνται επίσης οι διακυμάνσεις της υπηρεσίας. Για την αξιολόγηση της χρήσης των πόρων, κάθε υπηρεσία παρακολουθείται στενά καθ' όλη τη διάρκεια της εκτέλεσης του προγράμματος. Το πραγματικό κόστος μετάδοσης δεδομένων μέσω του δικτύου υπολογίζεται μετά την ολοκλήρωση της αποστολής των εικόνων από το drone στο cloud μέσω του σταθμού βάσης, παρέχοντας έτσι, πληροφορίες σχετικά με τις οικονομικές επιπτώσεις της λειτουργίας του προγράμματος.

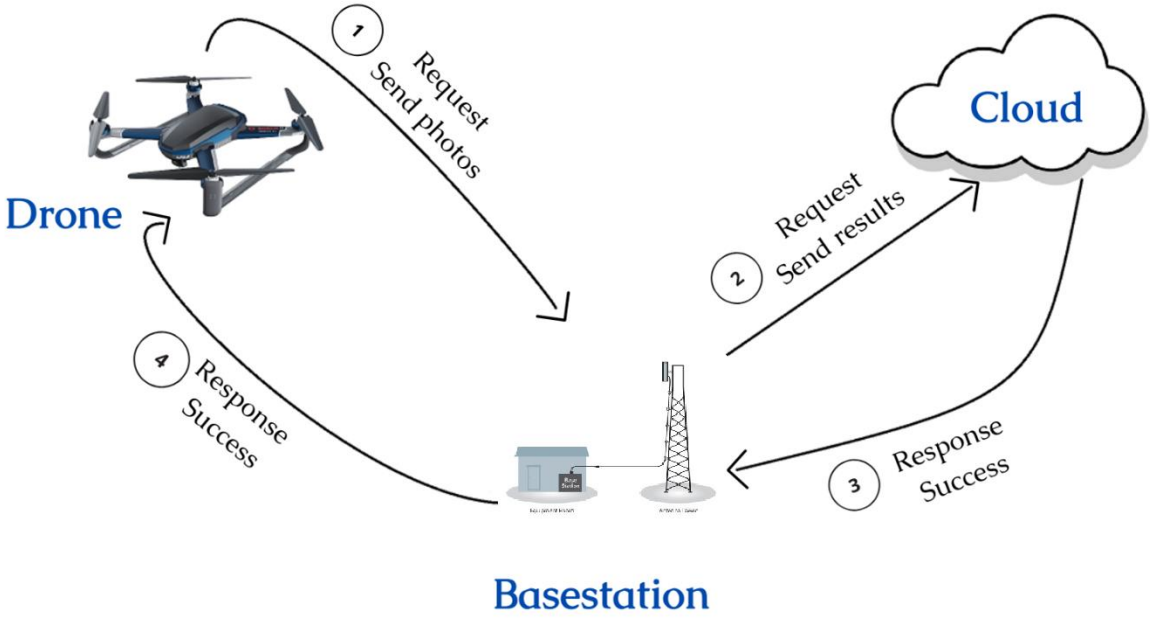
4.5 Συμπεράσματα

Η ολοκληρωμένη αξιολόγηση του προγράμματος παρέχει ανεκτίμητη πληροφορία και αποκαλύπτει τόσο τις δυνατότητες όσο και τις αδυναμίες σχετικά με την απόδοση και την ακρίβειά του. Αυτό διασφαλίζει ότι οι τελικοί χρήστες μπορούν να έχουν πλήρη εμπιστοσύνη στο πρόγραμμα για την παροχή ακριβών και έγκαιρων πληροφοριών. Με τη μέτρηση διαφόρων μετρήσεων, η αποδοτικότητα και η αποτελεσματικότητα του προγράμματος μπορούν να βελτιωθούν με την πάροδο του χρόνου, ενισχύοντας έτσι τη συνολική χρησιμότητά του. Η ανάλυση επιτρέπει επίσης την ακριβέστερη κατανόηση του τρόπου με τον οποίο το πρόγραμμα αποδίδει υπό διάφορες συνθήκες, επιτρέποντας στους χρήστες να προβλέπουν καλύτερα τυχόν πιθανά προβλήματα ή περιορισμούς.

Κεφάλαιο 5

Λεπτομέρειες Υλοποίησης

5.1 Ανάπτυξη της εφαρμογής του σεναρίου χρήσης



Σχήμα 5.1

Όπως παρατηρείτε στο πιο πάνω σχήμα, το πρόγραμμα χωρίζετε στα τρία microservices, drone, base station, cloud. Κάθε ένα από αυτά έχει σχεδιαστεί για να εκτελεί μια συγκεκριμένη εργασία και επικοινωνεί μεταξύ των άλλων microservices για να επιτευχθεί η συνολική λειτουργικότητα του προγράμματος.

Η χρήση της αρχιτεκτονικής των microservices, όπου διαίρει το πρόγραμμα σε τρία μικρότερα, ανεξάρτητα microservices δημιουργεί ένα πιο αρθρωτό, επεκτάσιμο και συντηρήσιμο πρόγραμμα. Η συγκεκριμένη αρχιτεκτονική επιτρέπει στα microservices να κλιμακωθούν ανεξάρτητα, δίνοντας τη δυνατότητα αναβάθμισης των συγκεκριμένων microservices που απαιτούν περισσότερους πόρους, χωρίς να επηρεάζουν τα υπόλοιπα. Επιπρόσθετα, τα microservices είναι «χαλαρά» συνδεδεμένα, επιτρέποντάς την ενημέρωση, την αντικατάσταση ή τον πολλαπλασιασμό ενός microservice χωρίς να επηρεαστούν τα υπόλοιπα. Επίσης, εάν ένα microservice αποτύχει, δεν θα επηρεάσει τα υπόλοιπα, επιτρέποντας στο υπόλοιπο πρόγραμμα να συνεχίσει να εκτελείται αφού το single point of failure δεν μπορεί να το εμποδίσει. Ακόμη, δεδομένου ότι τα τρία microservices αναπτύσσονται ανεξάρτητα και δοκιμάζονται ταχύτερα καθώς επίσης γίνετε και ευκολότερη η συντήρηση και η ενημέρωση τους μεμονωμένα.

Η εφαρμογή ξεκινά με το drone microservice να φορτώνει τις εικόνες από ένα φάκελο ο οποίος εμπεριέχει το dataset της έρευνας με φωτογραφίες που περιέχουν οχήματα για πειραματικούς σκοπούς αλλά και σκοπούς ανάπτυξης και επέκτασης του προγράμματος. Για το dataset δεν υπάρχει κάποιος περιορισμός έτσι ο κάθε χρήστης που θα θέλει να το δοκιμάσει με το dataset του, είναι επιτρεπτό. Κατά την φόρτωση τους εκτελείτε κάποια επεξεργασία των εικόνων ώστε να τοποθετηθούν σε tuples και το κάθε tuple με την εικόνα γίνετε append σε μια list. Αφού γίνει και η καταμέτρηση του αριθμού των εικόνων για αποστολή, καλείτε η συνάρτηση που είναι υπεύθυνη να τις στείλει στο basestation μέσω της χρήσης του FastAPI framework.

```yaml
# Environment variables
environment:
  ALL_TOGETHER: "False"
  NUMBER_OF_IMAGES: "2"
  BASESTATION_URL: "basestation:8000"
```

Σχήμα 5.2

Στην αρχή του προγράμματος καθορίζονται τρία environmental variable για να προσδιορίσουν τη μέθοδο αποστολής εικόνων.

- *ALL_TOGETHER*: είναι υπεύθυνη να καθορίσει κατά πόσο οι εικόνες θα αποσταλούν όλες μαζί σε ένα request του drone προς το basestation ή αν θα χωριστούν σε ισότιμα segments με το τελευταίο να αποστέλλει πάντα είτε ισότιμο segment με τα άλλα είτε τον εναπομείναντα αριθμό εικόνων. Αυτός ο διαχωρισμός μπορεί να θεωρηθεί ότι το drone τελειώνει την διαδρομή του και αποστέλλει όλες τις φωτογραφίες στο base station ή κατά την προκαθορισμένη διαδρομή του βρίσκει base stations στον δρόμο του και αποστέλλει ένα προκαθορισμένο αριθμό φωτογραφιών.
- *NUMBER_OF_IMAGES*: είναι ο αριθμός των εικόνων που καθορίζει τον μέγιστο αριθμό εικόνων που μπορούν να αποσταλούν σε κάθε segment.
- *BASESTATION_URL*: είναι η διεύθυνση URL του base station microservice στο οποίο το drone στέλνει τις εικόνες.

Τα environmental variables μπορούν να καθοριστούν με την χρήση του docker compose file ή αν δεν χρησιμοποιηθεί το συγκεκριμένο αρχείο για την δημιουργία των containers τότε μπορούν να αρχικοποιηθούν από την γραμμή εντολών με την δημιουργία των containers.

```
try:
    # Make an HTTP GET request to check if the connection to the basestation has been established
    response = requests.get(url=f"http://{BASESTATION_URL}/check/connection/basestation", timeout=10)
    if response.status_code == 200:
        # If the response status code is 200, log a message indicating that the request was successful
        logging.error("Request was successful")
        self.create_logs(txt_file, "Request was successful")

    else:
        # If the response status code is not 200, raise a SystemExit exception with an error message
        raise SystemExit(f"Request failed with status code: {response.status_code}")
except requests.exceptions.Timeout:
    # If the request times out, raise a SystemExit exception with an error message
    raise SystemExit("Request timed out")
except requests.exceptions.RequestException as e:
    # If an exception occurs, raise a SystemExit exception with an error message
    raise SystemExit(f"An error occurred: {e}")
```

Σχήμα 5.3

Αφού καθοριστούν, το drone στέλνει ένα request για να επιβεβαιώσει την σύνδεση του μαζί με το base station στο: */check/connection/basestation*, και όταν παραλάβει το response, περνά από τους ελέγχους για error και timeout.

```
drone        | ERROR:root:Request was successful with code: 200 and text: Connection established
```

Σχήμα 5.4

Εάν το status code είναι 200, τυπώνει ένα μήνυμα που υποδεικνύει ότι η αίτηση ήταν επιτυχής, εάν όχι δημιουργεί μια εξαίρεση SystemExit με ένα μήνυμα σφάλματος. Εάν προκύψουν εξαιρέσεις κατά τη διάρκεια της αίτησης ή εάν η αίτηση τερματιστεί, εγείρετε μια εξαίρεση SystemExit με ένα μήνυμα σφάλματος.

```
# Iterate over the list of images and group them in segments
for num_pic in range(0, total_num_of_imgs, imgs_in_segments):
    send_start_time = time.time()  # get the current time in seconds
    # Loop over the range of the segment and add images to the list_num_elements
    for x in range(0, imgs_in_segments):
        if num_pic + x < total_num_of_imgs:
            list_num_elements.append(self.images_list[num_pic + x])

    try:
        response = requests.post(
            # Endpoint URL to send images
            url=f"http://{BASESTATION_URL}/import/images/{num_pic}/{total_num_of_imgs}",
            files=list_num_elements,  # Images to be sent
            timeout=60
        )
```

Σχήμα 5.5

Ένα for loop είναι υπεύθυνο για τον διαχωρισμό των segments και με την χρήση του endpoint: */import/images/{num_pic}/{total_num_of_imgs}* αποστέλλετε με Http Post request η λίστα με τις εικόνες καθώς και κάποιες επιπρόσθετες πληροφορίες όπως ο αριθμός της θέσης της εικόνας στην αρχική λίστα αλλά και ο συνολικός αριθμός των εικόνων του dataset. Αφού λάβουμε την απάντηση από το basestation γίνονται οι σχετικοί ελέγχοι και αν είναι success τότε γίνετε ένα sleep των 5 δευτερολέπτων και συνεχίζει με την αποστολή του επόμενου segment εικόνων.

Σε κάθε επικοινωνία μεταξύ των microservices ,όπως βλέπουμε και πιο πάνω στον κώδικα, χρησιμοποιείτε ένα RESTful API που δημιουργήθηκε με FastAPI το οποίο απλοποιεί την επικοινωνία μεταξύ τους αλλά και παρέχει μια τυποποιημένη διαδικτυακή διεπαφή για την επίτευξη της διαλειτουργικότητας και αλληλεπίδρασης τους. Και τα τρία microservices στέλνουν αιτήματα HTTP ανά μεταξύ τους για να αποστέλλουν τα διάφορα δεδομένα όπως π.χ. εδώ είναι ένα απλό GET request και ένα POST request με τις εικόνες. Έπειτα, το FastAPI δημιουργεί γρήγορα τελικά σημεία για τα microservices.

Παρέχει εύκολη διαχείριση των εισερχόμενων αιτημάτων αλλά και διαχειρίζεται την αποστολή δεδομένων ή επιστροφή απάντησης σε τυποποιημένη μορφή, όπως JSON. Αυτό βοηθά στην δημιουργία ενός πιο επεκτάσιμου και ευέλικτου συστήματος.

Αφού ολοκληρωθεί η αποστολή όλων των εικόνων που περιλαμβάνονται στο dataset, οι σχετικές παράμετροι για τη χρονική διάρκεια της διαδικασίας αποθηκεύονται. Ωστόσο, επειδή το κομμάτι του κώδικα που καλεί τη συνάρτηση για την αποστολή των εικόνων βρίσκεται σε έναν infinity loop, η ίδια διαδικασία εκτελείται ξανά. Κατά τη διάρκεια αυτής της επανάληψης, υπάρχει η δυνατότητα να αλλάξει το dataset ή τα environmental variables.

Το base station microservice έχει τέσσερα διαφορετικά environmental variables :

```yaml
# Environment variables
environment:
  ALL_TOGETHER: "False"
  FIRST_COUNTER: "True"
  USE_FOLDER_FOR_SAVE: "True"
  CLOUD_URL: "cloud:8080"
```

Σχήμα 5.6

- *ALL_TOGETHER*: καθορίζει αν οι εικόνες που θα λάβει στο server κομμάτι του microservice θα είναι διαχωρισμένες σε segments ή όλες μαζί.
- *FIRST_COUNTER*: καθορίζει ποιος από τα δύο ML models, όπου θα αναλυθούν παρακάτω, θα χρησιμοποιηθεί για την καταμέτρηση των οχημάτων. Αν η τιμή είναι True τότε χρησιμοποιείτε ο πρώτος ενώ αν είναι False αυτόματα χρησιμοποιείτε ο δεύτερος.
- *USE_FOLDER_TO_SAVE*: καθορίζει αν θα χρησιμοποιηθεί ένας έξτρα φάκελος για αποθήκευση των εικόνων μόλις παραληφθούν και πριν επεξεργαστούν.
- *CLOUD_URL*: είναι η διεύθυνση URL του cloud microservice στο οποίο το base station στέλνει τα αποτελέσματα των καταμετρήσεων.

Στο πρόγραμμα είναι ενσωματωμένοι προς το παρόν μόνο δυο αλγόριθμοι για την καταμέτρηση των οχημάτων, όμως στο μέλλον θα μπορούσαν πολύ εύκολα να προστεθούν και άλλοι ανάλογα με τις ανάγκες που θα υπάρξουν. Πιο συγκεκριμένα, ο πρώτος αλγόριθμος χρησιμοποιεί τη βιβλιοθήκη OpenCV για την επεξεργασία εικόνων και την κλάση VehicleDetector από ένα ξεχωριστό module που χρησιμοποιεί ένα προεκπαιδευμένο νευρωνικό δίκτυο για τον εντοπισμό οχημάτων σε εικόνες.

```
for image_arr in images_details:
    # Generate a unique name for the image
    if not img_number:
        img_name = f"pic{counter}"
    else:
        img_name = f"pic{img_number + counter}"

    # Detect the vehicles in the image and count them
    vehicle_boxes = self.vd.detect_vehicles(image_arr)
    vehicle_count = len(vehicle_boxes)

    # Update total count
    vehicle_count_dict[img_name] = vehicle_count

    # Display the bounding boxes and count on the image
    self.count_vehicles(vehicle_boxes, image_arr, vehicle_count)

    counter += 1
```

Σχήμα 5.7

Αρχικά, η μέθοδος vehicle_count() δέχεται μια λίστα από numpy arrays και έναν προαιρετικό αριθμό εικόνας και επιστρέφει ένα dictionary που περιέχει τον αριθμό οχημάτων που καταμετρήθηκαν για κάθε εικόνα. Η συγκεκριμένη μέθοδος κάνει βρόχο σε κάθε εικόνα στη λίστα, ανιχνεύει τα οχήματα χρησιμοποιώντας την κλάση VehicleDetector που καλεί το function της, detect_vehicles(), και μετρά τον αριθμό των οχημάτων καλώντας τη μέθοδο count_vehicles() που σχεδιάζει τα πλαίσια οριοθέτησης. Η μέθοδος αυτή χρησιμοποιείται για την καταμέτρηση του αριθμού των οχημάτων σε κάθε εικόνα, τη σχεδίαση ενός οριοθετημένου πλαισίου γύρω από κάθε όχημα και την εμφάνιση του συνολικού αριθμού οχημάτων στην εικόνα.

Πιο αναλυτικά, όταν καλείτε η detect_vehicle(), συγκεκριμένα καλείτε η κλάση VehicleDetector και χρησιμοποιείται για την ανίχνευση οχημάτων σε εικόνες χρησιμοποιώντας το μοντέλο ανίχνευσης αντικειμένων YOLOv4.

```python
def __init__(self):
    # Load YOLOv4 object detection model
    net = cv2.dnn.readNet(
        "yolov4.weights",
        "yolov4.cfg"
    )
    self.model = cv2.dnn_DetectionModel(net)
    self.model.setInputParams(size=(832, 832), scale=1 / 255)

    # Define classes to detect (classes containing vehicles only)
    self.classes_allowed = [2, 3, 5, 6, 7]
```

Σχήμα 5.8

Το μοντέλο YOLOv4 φορτώνεται από τα αρχεία yolov4.weights και yolov4.cfg, τα οποία περιέχουν τα weights και τη διαμόρφωση του προ-εκπαιδευμένου μοντέλου YOLOv4. Η κλάση VehicleDetector διαθέτει την προαναφερόμενη μέθοδο detect_vehicles, η οποία δέχεται έναν numpy array ως είσοδο και επιστρέφει μια λίστα με τα πλαίσια οριοθέτησης των ανιχνευμένων οχημάτων. Η μέθοδος περνάει πρώτα την εικόνα μέσω του μοντέλου YOLOv4 χρησιμοποιώντας τη συνάρτηση cv2.dnn_DetectionModel.

```python
class_ids, scores, boxes = self.model.detect(img, nmsThreshold=0.4)
for class_id, score, box in zip(class_ids, scores, boxes):
    if score < 0.5:
        # Skip detection with low confidence
        continue

    if class_id in self.classes_allowed:
        # Add bounding box of vehicle to the list of detected vehicles
        vehicles_boxes.append(box)
```

Σχήμα 5.9

Στη συνέχεια, η μέθοδος φιλτράρει τα ανιχνευμένα αντικείμενα που δεν ταξινομούνται ως οχήματα και εκείνα με χαμηλή βαθμολογία εμπιστοσύνης.

Ο δεύτερος αλγόριθμος ορίζει μια κλάση που ονομάζεται VehicleCounting2, η οποία περιέχει μια στατική μέθοδο vehicle_count_2. Σκοπός αυτής της μεθόδου είναι να μετράει τον αριθμό των οχημάτων σε ένα σύνολο εικόνων εισόδου, χρησιμοποιώντας ένα προ-εκπαιδευμένο μοντέλο ανίχνευσης αντικειμένων RetinaNet.

Η μέθοδος δέχεται δύο ορίσματα: images_details, που είναι μια λίστα εικόνων εισόδου με τη μορφή PyTorch tensor, και img_number, που είναι ένας προαιρετικός ακέραιος που αντιπροσωπεύει τον αριθμό της αρχικής εικόνας.

Για κάθε εικόνα εισόδου, η μέθοδος καθορίζει πρώτα το όνομα της εικόνας με βάση τον τρέχοντα αριθμό εικόνας και τον αριθμό αρχικής εικόνας.

```
# Load pre-trained RetinaNet model
weights = RetinaNet_ResNet50_FPN_V2_Weights.DEFAULT
model = retinanet_resnet50_fpn_v2(weights=weights, score_thresh=0.35)
# Put the model in inference mode
model.eval()
```

Σχήμα 5.10

Στη συνέχεια, φορτώνει το προ-εκπαιδευμένο μοντέλο RetinaNet και το θέτει σε λειτουργία εξαγωγής συμπερασμάτων.

```
# Get the transforms for the model's weights
preprocess = weights.transforms()
# Preprocess the image using the transforms from our weights, create a batch and run inference
batch = [preprocess(img)]
prediction = model(batch)[0]
```

Σχήμα 5.11

Η εικόνα εισόδου προεπεξεργάζεται χρησιμοποιώντας τους μετασχηματισμούς που παρέχονται από τα weights του μοντέλου και στη συνέχεια περνάει από το μοντέλο για να λάβει ένα prediction.

```
# Extract predicted labels and count vehicles
labels = [weights.meta["categories"][i] for i in prediction["labels"]]
# Extract the labels with respect to the metadata from the weights
vehicle_count = len([i for i in labels if i == "car"])
```

Σχήμα 5.12

Εξάγονται οι ετικέτες των προβλεπόμενων αντικειμένων και ο αριθμός των οχημάτων στην εικόνα υπολογίζεται μετρώντας τις εμφανίσεις της ετικέτας "car".
Τέλος, ο αριθμός των οχημάτων για την τρέχουσα εικόνα προστίθεται στο vehicle_count_dict και ο μετρητής αυξάνεται. Η μέθοδος επιστρέφει το vehicle_count_dict που περιέχει τις μετρήσεις οχημάτων για όλες τις εικόνες εισόδου.

Επίσης το service χωρίζετε σε δύο αρχεία κώδικα τα οποία το ένα είναι η λειτουργία του server που ορίζει το κύριο τελικό σημείο της εφαρμογής χρησιμοποιώντας το διαδικτυακό πλαίσιο FastAPI.

```
@app.post("/import/images/{num_pic}/{num_of_repeats}", response_model=VehicleCountResponse)
async def drone(num_pic: int, num_of_repeats: int, files: List[UploadFile] = File(...)):
```

Σχήμα 5.13

Το endpoint λαμβάνει μια λίστα αρχείων εικόνας που αποστέλλονται από ένα drone, τα χωρίζει σε τμήματα και επεξεργάζεται κάθε τμήμα χρησιμοποιώντας την κλάση BasestationMain.

```
# If cloud response is successful, return a success response. Otherwise, return an error response.
if cloud_response:
    return VehicleCountResponse(
        success=True,
        message="Process completed successfully"
    )
else:
    return VehicleCountResponse(
        success=False,
        message="Process in cloud didn't complete successfully"
    )
```

Σχήμα 5.14

Η συνάρτηση επιστρέφει ένα αντικείμενο VehicleCountResponse με ένα success flag και ένα μήνυμα που δείχνει αν η διαδικασία ολοκληρώθηκε με επιτυχία ή όχι.

```
# If all images have been processed, send the updated response_cloud dictionary to cloud
# and update cloud_response variable
if num_pic + len(files) is num_of_repeats:
    send_start_time = time.time()  # get the current time in seconds
    cloud_response = await send_to_cloud(service, response_cloud)
    send_end_time = time.time()  # get the current time again
    print(f"Cloud response: {cloud_response}")
    create_logs(txt_file, f"Cloud response: {cloud_response}")
```

Σχήμα 5.15

Εάν έχουν υποστεί επεξεργασία όλες οι εικόνες, η συνάρτηση καλεί την συνάρτηση send_to_cloud που αποστέλλει το λεξικό response_cloud στο cloud και ενημερώνει τη μεταβλητή cloud_response.

Το τελικό σημείο check_conn χρησιμοποιείται για τον έλεγχο της σύνδεσης μεταξύ του drone και του σταθμού βάσης. Το drone στέλνει ένα αίτημα GET στο microservice base station για τον έλεγχο της σύνδεσης. Εάν ο κωδικός κατάστασης απόκρισης δεν είναι 200, η συνάρτηση δημιουργεί μια εξαίρεση SystemExit με τον κωδικό κατάστασης απόκρισης. Εάν η αίτηση λήξει χρονικά, η συνάρτηση εγείρει μια εξαίρεση SystemExit με ένα μήνυμα που υποδεικνύει ότι η αίτηση έληξε χρονικά.

```
# Create an instance of BasestationMain
service = BasestationMain()

# Divide images in segments and process each segment separately
response = service.divide_in_segments(files, num_pic)
# Update response_cloud dictionary with response
response_cloud.update(response)
```

Σχήμα Error! No text of specified style in document..**5.16**

Αφού παραλάβουμε το request από το drone, καλούμε το δεύτερο κομμάτι του microservice για να γίνει η επεξεργασία και η καταμέτρηση.

```
def divide_in_segments(self, files, num_pic):
```

Σχήμα 5.17

Η κύρια συνάρτηση του κώδικα είναι η divide_in_segments, η οποία δέχεται δύο ορίσματα: files και num_pic. Το files είναι μια λίστα εικόνων και το num_pic είναι ο αριθμός της πρώτης εικόνας στο συγκεκριμένο segment.

```
# Loop through all the files
for file in files:
    contents = file.file.read()
    image = Image.open(io.BytesIO(contents))

    # If using a folder for saving
    if USE_FOLDER_FOR_SAVE:  # Save in folder and make a list with.
        save = "Use folder to save | "
        # Save the image to a folder and append its file path to the images_list
        self.save_img(image, num_pic + count)

    # Otherwise
    else:  # Don't save and make a list with numpy ndarray each image.
        save = "No use folder to save | "
        if FIRST_COUNTER:
            # Convert the image to a numpy array and append it to the images_list
            image_array = np.array(image)
            images_list.append(image_array)
        else:
            # Apply the ToTensor transform to convert the image to a PyTorch tensor
            # and append it to the images_list
            images_list.append(transforms.ToTensor()(image))
    count += 1
```

Σχήμα 5.18

Κάνει βρόχο σε όλα τα αρχεία, ανοίγει κάθε εικόνα, την μετατρέπει σε bytes και την αποθηκεύει σε ένα φάκελο ή δημιουργεί μια λίστα ανάλογα με το αν η μεταβλητή USE_FOLDER_FOR_SAVE είναι true ή false. Μετά, παίρνουμε τις εικόνες, είτε από τον έξτρα φάκελο αποθήκευσης αν είναι αποθηκευμένες, είτε απευθείας από το request που της μετατρέψαμε σε bytes και ανάλογα με το ποιο ML model θα χρησιμοποιήθεί, δημιουργείται το κατάλληλο list με τον τύπο της εικόνας που δέχεται ο αλγόριθμος για να μπορέσουμε να τις τροφοδοτήσουμε σε αυτόν. Ο πρώτος αλγόριθμος λόγο του ότι χρησιμοποιεί το YOLOv4 neural network model απαιτεί ο τύπος της φωτογραφίας να είναι numpy array ενώ ο δεύτερος αντίστοιχα αφού χρησιμοποιεί το ResNet pre-trained torchvision model απαιτεί ο τύπος των εικόνων να είναι PyTorch tensor.

```
# Choose which counter to use
if FIRST_COUNTER:
    # Call the vehicle_counter_1's vehicle_count method with the images_list
    # and the number of the first image in the segment
    response = self.vehicle_counter_1.vehicle_count(images_list, num_pic)
    counter = "and the vehicle counting (First)."
else:
    # Call the vehicle_counter_2's vehicle_count_2 method with the images_list
    # and the number of the first image in the segment
    response = self.vehicle_counter_2.vehicle_count_2(images_list, num_pic)
    counter = "and the car recognition second (Second)."
```

Σχήμα 5.19

Στη συνέχεια, καλεί είτε τη μέθοδο vehicle_counter_1's vehicle_count είτε τη vehicle_counter_2's vehicle_count_2, όπου είναι οι αλγόριθμοι οι οποίοι περιεγράφηκαν πιο πάνω, ανάλογα με το αν η μεταβλητή FIRST_COUNTER είναι true ή false, περνώντας τη λίστα των εικόνων και τον αριθμό της πρώτης εικόνας στο module. Καταγράφει τις πληροφορίες και αδειάζει το φάκελο που χρησιμοποιείται για την αποθήκευση και επιστρέφει τα αποτελέσματα σε μορφή dictionary με key τον αριθμό της εικόνας και value τον αριθμό οχημάτων που μέτρησε από τις εικόνες.

```
def send_to_cloud(self, road_info):
    try:
        # Send a POST request to the cloud service with road_info data as the payload
        response = requests.post(
            url=f"http://{CLOUD_URL}/cloud/information",
            data=json.dumps(road_info),
            timeout=60
        )
```

Σχήμα 5.20

Αν ολοκληρωθούν οι εικόνες του dataset με την επεξεργασία, τότε το παραγόμενο dictionary αποστέλλετε στο cloud microservice με Http post request στο /cloud/information αφού πρώτα γίνει ο έλεγχος για την σύνδεση του base station – cloud και είναι επιτυχής. Αν δεν ολοκληρώθηκαν τότε ο server του base station περιμένει το επόμενο request με το επόμενο segment και η διαδικασία επαναλαμβάνετε ωσότου ολοκληρωθεί ο αριθμός των εικόνων.

```
# Defining a route for handling incoming requests to the '/cloud/information' endpoint
@app.post("/cloud/information")
async def cloud(road_info: dict = Body(...)):
```

Σχήμα 5.21

```
# Defining a route for checking the connection to the Cloud Server
@app.get("/check/connection/cloud")
async def check_connection():
    return "Connection established to Cloud Server"
```

Σχήμα 5.22

Το cloud microservice χωρίζετε και αυτό σε δύο αρχεία κώδικα όπως το base station. Το ένα είναι η λειτουργία του server που λαμβάνει requests στα δύο endpoints του και αποστέλλει πίσω response στο base station microservice. Ενώ το άλλο είναι αυτό που ασχολείται με την επεξεργασία των απεσταλμένων πληροφοριών και την αποθήκευσή τους.

```
# Creating an instance of the CloudMain class
service = CloudMain()
# Calling the 'road_check' method of the CloudMain instance with the received information
response = service.road_check(road_info)
```

Σχήμα 5.23

Όταν στέλνετε ένα Post request καλείτε η συνάρτηση cloud(), του πρώτου μέρους του microservice, η οποία στη συνέχεια δημιουργείται ένα instance της κλάσης CloudMain και καλείται η μέθοδος road_check αυτής της περίπτωσης με τα δεδομένα που λαμβάνονται. Τέλος, η απάντηση από τη ποιο πάνω μέθοδο επιστρέφεται στο microservice του base station.

Ποιο συγκεκριμένα, για το δεύτερο κομμάτι του cloud microservice, λαμβάνει τα αποτελέσματα της καταμέτρησης των οχημάτων και τα αποθηκεύει. Τα συγκεκριμένα αποτελέσματα θα μπορούσαν να χρησιμοποιηθούν είτε από την τροχαία είτε από ιδιωτικές εταιρείες για να ενημερώνουν τους οδηγούς με αυτόνομα αυτοκίνητα σχετικά με τις συνθήκες κυκλοφορίας σε πραγματικό χρόνο. Ταυτόχρονα με τα πιο πάνω, η λειτουργεία αποθηκεύει τα αποτελέσματα και κάποιες μετρήσεις σχετικά με το πρόγραμμα.

```
# Check if the "info" argument is a dictionary
if type(info) is dict:
    # Print the dictionary from basestation
    print(f"Dictionary from basestation with number of vehicles: \n{info}")
    logging.error(f"Dictionary from basestation with number of vehicles: \n{info}")
    self.create_logs(f"Dictionary from basestation with number of vehicles: {info}")
    self.save_dict_to_csv(csv_file, info)

    # Return True if "info" is a dictionary
    return True
# Return False if "info" is not a dictionary
return False
```

Σχήμα 5.24

Στο πρόγραμμα, η ενημέρωση δεν πραγματοποιείτε αλλά στην θέση της υλοποιήθηκε απλά ένας έλεγχος ότι το request από το base station είναι έγκυρο και έχει την μορφή του dictionary τότε επιστρέφει την τιμή True. Εάν το request δεν είναι dictionary, επιστρέφει False.

Αν όλα είναι εντάξει τότε το cloud απαντά με success στο base station, και αυτό στην συνέχεια απαντά στο drone για να γνωρίζει ότι η διαδικασία ήταν επιτυχείς και οι πληροφορίες των εικόνων έφτασαν στο cloud.

Η επικοινωνία που πραγματοποιείτε μεταξύ των microservices περιγράφετε κατάλληλα από την αρχιτεκτονική client – server, η οποία είναι η ιδανική για την επικοινωνία τους στο πρόγραμμα. Αυτή η αρχιτεκτονική βοηθά στην οργάνωση της επικοινωνίας μεταξύ των microservices και παρέχει ένα επεκτάσιμο και ευέλικτο σύστημα. Επιπλέον, παρέχει separation of concerns μεταξύ των διαφόρων microservices, καθιστώντας ευκολότερη τη διαχείριση και τη συντήρηση του πιο πάνω προγράμματος.

Συγκεκριμένα για το πρόγραμμα, το base station και το cloud microservice ενεργούν ως servers και το drone microservice ενεργεί ως client. Το drone microservice στέλνει requests στο base station για να στείλει τις φωτογραφίες που έχουν ληφθεί και το base station απαντά με response ότι η καταμέτρηση και η αποστολή της καταμέτρησης των οχημάτων στο cloud microservice ήταν επιτυχής. Το cloud microservice μπορεί να ενεργεί ως server για το base station microservice, λαμβάνοντας τα αποτελέσματα της καταμέτρησης οχημάτων και αποστέλλοντας τις πληροφορίες στα αυτόνομα αυτοκίνητα.

5.2 Containerization

```
FROM python:3.10

RUN pip3 install --upgrade pip

RUN mkdir /code
WORKDIR /code

ADD requirements.txt /code/

RUN pip3 install -r requirements.txt

ADD ./code/ /code/

CMD sleep 10 && python drone_main.py
```

```
FROM python:3.10

RUN pip3 install --upgrade pip


RUN mkdir /code
WORKDIR /code

ADD requirements.txt /code/

RUN pip install --cache-dir=./pip_cache -r requirements.txt

EXPOSE 8000

ADD ./code/ /code/

CMD ["python", "basestation_server.py" ]
```

```
FROM python:3.10

RUN pip3 install --upgrade pip

RUN mkdir /code
WORKDIR /code

ADD requirements.txt /code/
RUN pip3 install -r requirements.txt

EXPOSE 8080

ADD ./code/ /code/

CMD ["python", "cloud_server.py" ]
```

Σχήμα 5.25

Η χρήση των Docker Containers στο πιο πάνω σενάριο παρέχει πολλά πλεονεκτήματα. Τα παραπάνω αρχεία Docker χρησιμοποιούνται για τη δημιουργία container. Και για τα τρία microservices το docker file παραμένει το ίδιο με κάποιες μικρές διαφορές στο τελευταίο κομμάτι του. Το στοιχείο της απομόνωσης κάθε microservice που εκτελείται στο δικό της docker container διασφαλίζει ότι κάθε microservice έχει το δικό της περιβάλλον και δεν παρεμβαίνει με άλλες microservices ή το λειτουργικό σύστημα του κεντρικού υπολογιστή. Αυτό αυξάνει τη διαχειρισιμότητα των εξαρτήσεων και μειώνει τις συγκρούσεις μεταξύ των microservices. Επιπλέον, δεδομένου ότι κάθε microservice επικοινωνεί μόνο μέσω του δικτύου, η απομόνωση κάθε κομματιού που εκτελείται στο ίδιο σύστημα είναι κρίσιμη.

Το πρώτο βήμα, είναι να αντλήσουμε την τελευταία βασική εικόνα της Python 3.10. Στη συνέχεια, ο διαχειριστής πακέτων pip αναβαθμίζεται στην τελευταία έκδοση του. Στη συνέχεια, δημιουργείται ένα νέο directory με όνομα 'code' στο container και ορίζεται ως working directory. Το αρχείο requirements.txt, το οποίο περιέχει μια λίστα με όλα τα πακέτα Python που απαιτούνται από την εφαρμογή, αντιγράφεται στον κατάλογο /code του container. Αντίστοιχα, στο αρχείο του base station η εντολή 'RUN' εγκαθιστά τις εξαρτήσεις και αποθηκεύει τις εγκατεστημένες εξαρτήσεις στον κατάλογο 'pip_cache'. Τα απαιτούμενα πακέτα εγκαθίστανται στη συνέχεια με τη χρήση του pip. Στη συνέχεια, τα περιεχόμενα του καταλόγου ./code από το local machine αντιγράφονται και αυτά στον κατάλογο /code του container. Το Dockerfile του basestation και του cloud έχουν την

έξτρα εντολή 'EXPOSE' η οποία εκθέτει τη θύρα 8000 και 8080, αντίστοιχα, στον έξω κόσμο.

Επιπρόσθετα, για το image του drone, εκτελείται η εντολή 'sleep 10 && python drone_main.py', η οποία θα περιμένει 10 δευτερόλεπτα πριν εκτελέσει την εντολή 'python drone_main.py' στο container. Έχουμε αντίστοιχα τις εντολές Python 'basestation_server.py' και Python 'cloud_server.py' που θα εκτελεστούν κατά την εκκίνηση του docker container. Ο σκοπός του sleep είναι σε περίπτωση που ξεκινήσουν όλα τα containers μαζί, να δώσει χρόνο στους server του base station και του cloud να ξεκινήσουν πρώτοι για να διαχειριστούν τα requests από drone και μετά από base station αντίστοιχα.

Συνολικά, η χρήση των docker container για να τοποθετήσουμε τα microservices, πέραν από την απομόνωση, βοηθούν και στην φορητότητα η οποία επιτρέπει την εύκολη ανάπτυξη και εκτέλεση των microservices σε οποιαδήποτε πλατφόρμα που υποστηρίζει το Docker, διευκολύνοντας τη μετακίνηση τους μεταξύ διαφορετικών περιβαλλόντων και την κλιμάκωση του έργου ανάλογα με τις ανάγκες. Αυτό επιτρέπει ένα πιο σταθερό και αξιόπιστο περιβάλλον σε διαφορετικά μηχανήματα. Και με τα διαφορετικά μηχανήματα εννοούμε τον υπολογιστή που γίνετε η υλοποίηση του προγράμματος, και τον υπολογιστή ο οποίος θα χρησιμοποιηθεί σαν emulator για να τρέξουμε τα πειράματα. Τέλος, ακόμα ένα βοήθημα που προσφέρουν τα Docker Containers είναι η εύκολα διαχειρίσιμη έκδοση των microservices, καθώς κάθε microservice έχει το δικό της docker image όπως βλέπουμε και πιο πάνω, συμπεριλαμβανομένου του κώδικα του microservice και των εξαρτήσεων, απλοποιώντας τη διαχείριση αλλαγών και τις επαναφορές, ιδίως σε πειραματικά σημεία της έρευνας.

Σχήμα 5.26

Με το Docker Compose, διαχειριζόμαστε την ανάπτυξη και το συντονισμό των Docker containers που αποτελούν το πρόγραμμά, το οποίο χωρίζεται στα τρία πιο πάνω microservices. Το πιο πάνω αρχείο Docker Compose περιγράφει τις διάφορες υπηρεσίες που θα εκτελούνται σε περιβάλλον Docker. Το Docker Compose file μας παρέχει την δυνατότητα να διαχειριζόμαστε την ροή του προγράμματος και τις διαφορετικές καταμετρήσεις των παραμέτρων σε κάθε πείραμα κατά την εκτέλεση του, με την αλλαγή των environmental variables που έχει το κάθε service. Συνολικά μπορούμε να παράξουμε οκτώ διαφορετικές ροές του προγράμματος. Με το Docker Compose όχι μόνο ορίζουμε την αρχιτεκτονική και τις εξαρτήσεις του προγράμματός σε ένα μόνο αρχείο (docker-compose.yaml), αλλά και αναπτύσσουμε γρήγορα και εύκολα ολόκληρο το πρόγραμμά μόνο με μία εντολή. (Επειδή θα χρησιμοποιηθούν minimum 3 microservices και αυτό ευκολύνει την διαδικασία και είναι όλα οργανωμένα σε ένα αρχείο). Με το Docker Compose, μπορούμε να διαχειριστούμε και να κλιμακώσουμε μαζί όλα τα containers. Έπειτα, μπορούμε να καθορίσουμε και να μεταβάλουμε τον αριθμό των αντιγράφων για κάθε microservice στο αρχείο docker-compose.yml ανάλογα με τις ανάγκες μας και το Docker Compose θα διασφαλίσει ότι ο επιθυμητός αριθμός Docker Containers εκτελείται ανά πάσα στιγμή.

Αρχικά, ορίζονται οι υπηρεσίες που θα εκτελούνται στο περιβάλλον, συμπεριλαμβανομένων των base station, drone και cloud microservices. Το base station service ορίζεται με ένα όνομα 'basestation' και κατασκευάζεται από το Dockerfile αρχείο στο './BaseStation' directory. Η υπηρεσία ορίζει επίσης τέσσερις μεταβλητές

περιβάλλοντος και αντιστοιχίζει τη θύρα 8000 στο container με την θύρα 8000 στον κεντρικό υπολογιστή. Επιπλέον, η υπηρεσία προσαρτά τον κατάλογο './data' στον κατάλογο './data' στο container και το συνδέει στο δίκτυο με όνομα 'internet'.
Παρομοίως, η υπηρεσία drone κατασκευάζεται από το αντίστοιχο Dockerfile στο './Drone' directory. Και αυτή η υπηρεσία ορίζει τρεις μεταβλητές περιβάλλοντος, προσαρτά το './data' directory στο '/data' directory στο container και το συνδέει στο ίδιο δίκτυο με όνομα 'internet'. Επίσης, εξαρτάται από την υπηρεσία του basestation που πρέπει να είναι σε λειτουργία πριν ξεκινήσει να κτίζεται το συγκεκριμένο container.
Τέλος, το cloud service ορίζεται με όνομα 'cloud' και κατασκευάζεται από το Dockerfile της στο './Cloud' directory. Η υπηρεσία αντιστοιχίζει τη θύρα 8080 στο container με την θύρα 8080 στον κεντρικό υπολογιστή και προσαρτά το './data' directory στο '/data' directory του container. Το cloud service είναι επίσης συνδεδεμένο στο δίκτυο 'internet'.
Το Docker Compose δημιουργεί ένα κοινόχρηστο δίκτυο για τα Docker Containers, το οποίο τους επιτρέπει να επικοινωνούν εύκολα μεταξύ τους, αλλά και να αποστέλλουν διαφόρων τύπων δεδομένα, μεταξύ άλλων και εικόνες, όπως υλοποιήθηκε και στο πιο πάνω πρόγραμμα. Το αρχείο ολοκληρώνεται με τον ορισμό του δικτύου 'internet'. Οι πιο πάνω υπηρεσίες μπορούν να επικοινωνούν μεταξύ τους χρησιμοποιώντας τα ονόματα των containers τους. Αυτό είναι απαραίτητο για την απρόσκοπτη συνεργασία και επικοινωνία των microservices. Το δίκτυο αυτό αρχικοποιείται στο docker-compose.yaml πρέπει επίσης να αναφέρεται ξεχωριστά ως χρησιμοποιούμενο δίκτυο σε κάθε microservice του αρχείου.

5.3 Προσομοίωση

Για να εξασφαλιστεί η ομαλή και αποτελεσματική λειτουργία του έργου των τριών microservices, χρησιμοποιούνται διάφορα εργαλεία και τεχνολογίες. Ο emulator είναι ένα από αυτά τα εργαλεία, το οποίο χρησιμοποιείται για την προσομοίωση της συμπεριφοράς του φυσικού υλικού και του λογισμικού με το οποίο αλληλεπιδρούν τα microservices. Αυτό βοηθά στη δοκιμή και την ανάπτυξη του λογισμικού των τριών microservices σε ένα ελεγχόμενο περιβάλλον. Για παράδειγμα, μπορούμε να προσομοιώσουμε τη συμπεριφορά του drone και να δοκιμάσουμε πώς το base station microservice επεξεργάζεται τις εικόνες που συλλαμβάνει το drone. Μπορούμε επίσης να προσομοιώσουμε τη συμπεριφορά του base station μετά την ολοκλήρωση της

καταμέτρησης και να δοκιμάσουμε τον τρόπο με τον οποίο το cloud microservice επικοινωνεί με το base station για να λάβει τα αποτελέσματα και να ενημερώσει τους οδηγούς με την κυκλοφορία σε συγκεκριμένους δρόμους.

Ένα άλλο εργαλείο που χρησιμοποιήθηκε στην εφαρμογή είναι το Fogify, ένα εργαλείο προσομοίωσης και δοκιμής για εφαρμογές fog computing. Το Fogify επιτρέπει τη δημιουργία ενός εικονικού περιβάλλοντος που μιμείται ένα πραγματικό περιβάλλον υπολογιστών Fog. Σε αυτό το εικονικό περιβάλλον, οι συσκευές άκρων, όπως τα drones και τα base stations, συνεργάζονται με το cloud για την εκτέλεση σύνθετων εργασιών. Χρησιμοποιώντας το Fogify, μπορούμε να προσομοιώσουμε τις αλληλεπιδράσεις μεταξύ αυτών των συσκευών άκρων και του νέφους υπό διάφορα σενάρια και συνθήκες. Για παράδειγμα, μπορούμε να δημιουργήσουμε ένα εικονικό περιβάλλον όπου πολλά drones πετούν σε μια συγκεκριμένη περιοχή, συλλαμβάνουν εικόνες και τις στέλνουν στο base station, το οποίο στη συνέχεια τις επεξεργάζεται και στέλνει τα αποτελέσματα στο cloud. Στη συνέχεια, το cloud ενημερώνει τα αυτόνομα αυτοκίνητα για τις οδικές συνθήκες. Μέσω αυτών των δοκιμών, μπορούμε να εντοπίσουμε και να αντιμετωπίσουμε πιθανά προβλήματα ή σημεία συμφόρησης, καθώς και να βρούμε το ιδανικό σενάριο για την ανάπτυξη του προγράμματος σε ένα πραγματικό περιβάλλον.

Τέλος, το 5G-Slicer είναι ένα άλλο εργαλείο που χρησιμοποιείται στο έργο, το οποίο επιτρέπει το slicing του δικτύου και την κατανομή των απαιτούμενων δικτυακών πόρων στα διάφορα microservices. Για παράδειγμα, το drone microservice μπορεί να απαιτεί χαμηλή καθυστέρηση και υψηλό εύρος ζώνης για να διασφαλίσει τη λήψη και τη μετάδοση εικόνας σε πραγματικό χρόνο, ενώ το base station microservice μπορεί να απαιτεί υψηλότερη αξιοπιστία και ασφάλεια για να διασφαλίσει ότι οι μετρήσεις του οχήματος είναι ακριβείς και δεν διακυβεύονται. Το 5G-Slicer υποστηρίζει δυναμικό τεμαχισμό δικτύου, επιτρέποντάς του να προσαρμόζεται στις μεταβαλλόμενες συνθήκες του δικτύου και να προσαρμόζει ανάλογα τις παραμέτρους τεμαχισμού. Αυτό είναι χρήσιμο σε σενάρια όπου η ζήτηση για πόρους δικτύου μπορεί να αλλάξει με την πάροδο του χρόνου και του τόπου.

Με τη χρήση αυτών των εργαλείων και τεχνολογιών, η πιο πάνω εφαρμογή μπορεί να δοκιμαστεί και να βελτιστοποιηθεί σε ελεγχόμενο περιβάλλον. Αυτό μπορεί να συμβάλει στη βελτίωση της απόδοσης και της αξιοπιστίας του έργου, ιδίως σε περιβάλλον δικτύου 5G, και να διασφαλίσει ότι τα microservices συνεργάζονται αποτελεσματικά για την επίτευξη των αντίστοιχων στόχων τους.

Κεφάλαιο 6

Πειραματισμός

Για καλύτερη κατανόηση των μετρικών και των ρυθμίσεων των παραμέτρων που χρησιμοποιήθηκαν για την εξαγωγή των πιο κάτω γραφικών και συμπερασμάτων παρέχεται το πιο κάτω υπόμνημα με τις αντίστοιχες συντομογραφίες της κάθε μετρικής.

Υπόμνημα

Xms, Yms, ML model, Zcores, #GB, @Images

Xms D->B: Το network delay (round-trip time) μεταξύ των δύο συσκευών drone και base station είναι Xms κατά το συγκεκριμένο πείραμα.

Yms B->C: Το network delay (round-trip time) μεταξύ των δύο συσκευών base station και cloud είναι Yms κατά το συγκεκριμένο πείραμα.

1ο ML model/2ο ML model: Ποιο ML model από τα δύο του προγράμματος χρησιμοποιήθηκε κατά το συγκεκριμένο πείραμα.

Zcores: Ο αριθμός (Z) των cores της CPU οι οποίοι ήταν ενσωματωμένοι στο base station infrastructure κατά την εκτέλεση του συγκεκριμένου πειράματος.

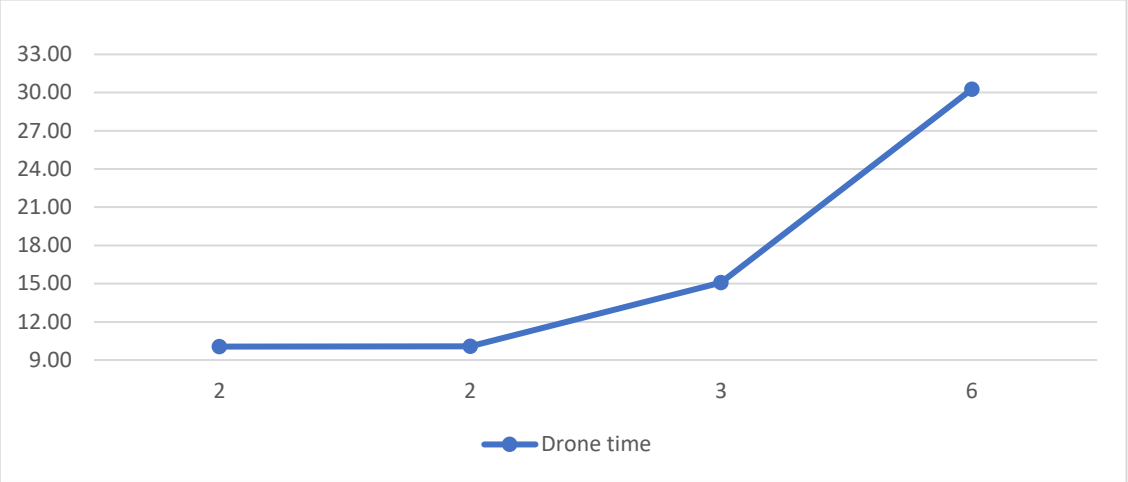
#GB: Ο αριθμός (#) των Gigabytes της μνήμης RAM η οποία ήταν ενσωματωμένη στο base station infrastructure κατά την εκτέλεση του συγκεκριμένου πειράματος.
@Images: Ο αριθμός (@) των εικόνων που αποστέλλονται από το drone προς το base station ανά request κατά την εκτέλεση του συγκεκριμένου πειράματος.
Υποσημείωση: Για τα παρακάτω πειράματα χρησιμοποιήθηκε ένα dataset 6 εικόνων με οχήματα στο drone microservice.
Οι τιμές που παίρνουν στο πείραμα οι παραπάνω μεταβλητές είναι οι εξής:

- X = 5, 10, 20, 30, 50, 100, 200
- Z = 2, 4
- @ = 1, 2, 3, 4
- Y = 5, 100
- # = 2, 4

6.1 Πείραμα 1

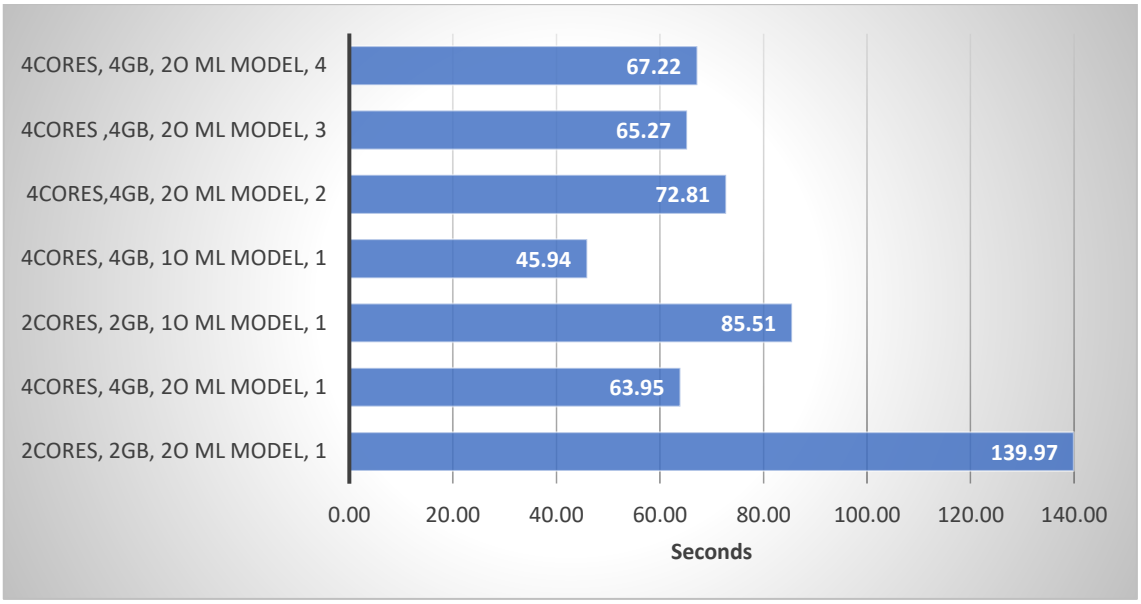
Οι μεταβλητές time markers παίζουν καθοριστικό ρόλο στο συγκεκριμένο πείραμα, καθώς χρησιμοποιούνται σε όλο τον κώδικα για να μετρήσουν με ακρίβεια το χρόνο που χρειάζεται κάθε microservice για να ολοκληρώσει την καθορισμένη εργασία της σε σχέση με τη συνολική εκτέλεση του προγράμματος. Αναλύοντας προσεκτικά τα time duration των time markers, μπορούμε να υπολογίσουμε την καθυστέρηση ολόκληρου του προγράμματος και των μεμονωμένων υπηρεσιών με βάση διάφορες παραμέτρους.
Με την σχολαστική ανάλυση των μεταβλητών time markers, στοχεύουμε αρχικά να μετρήσουμε τον χρόνο που χρειάζεται κάθε microservice για να ολοκληρώσει την εργασία της και το χρονικό διάστημα της συνολικής καθυστέρησης του προγράμματος. Επίσης, μπορούμε να μετρήσουμε τον τρόπο με τον οποίο μεταβάλλεται η καθυστέρηση σε διάφορες υπηρεσίες, λαμβάνοντας υπόψη διαφορετικές παραμέτρους του προγράμματος. Αποκτώντας γνώσεις σχετικά με τις πτυχές αυτές, μπορούμε να αξιολογήσουμε την αποδοτικότητα και την αποτελεσματικότητα της αρχιτεκτονικής των μικρουπηρεσιών.
Για τα παρακάτω πειράματα, όπου μετριέται το network delay μεταξύ των microservices drone και base station, το network delay αντικατοπτρίζει επίσης, την απόσταση μεταξύ των δύο αυτών συσκευών. Να σημειωθεί ότι στα πειράματα όπου το network delay μεταξύ των drone – base station είναι 100ms ή 200ms είναι επειδή το drone επικοινωνεί κατευθείαν με το cloud όπου είναι το base station υλοποιημένο και είναι και deploy ο ML

model αλγόριθμος και δεν υπάρχει καμία επικοινωνία μεταξύ drone και της υποδομής base station.



Experiment number	Images/request	Requests/execution	Drone time
1	3	2	10.06
2	3	2	10.07
3	2	3	15.07
4	1	6	30.26

Ο χρόνος που χρειάζεται το drone να επεξεργαστεί τις εικόνες χωρίς να τις αποστείλει, δεν εξαρτάται μόνο από τον αριθμό των εικόνων που μπαίνουν ανά request αλλά και αλυσιδωτά από τον αριθμό των requests που θα κάνει στο base station ανά execution. Αυτό παρατηρείτε από την πιο πάνω γραφική και πίνακα όπου ο χρόνος εκτέλεσης του drone στέλνοντας μια φωτογραφία ανά request, το οποίο συνεπάγετε με 6 requests ανά εκτέλεση, κυμαίνεται στα 30,26 seconds. Στο επερχόμενο πείραμα, όπου τα requests ανά εκτέλεση είναι 3, δηλαδή τα μισά του προηγούμενου, ο συνολικός χρόνος είναι το ½ του προηγούμενου χρόνου. Ανάλογα, όταν τα requests είναι 2 ανά εκτέλεση, ο χρόνος είναι το 1/3 του αρχικού χρόνου. Εδώ παρατηρείτε μια αναλογία μεταξύ του αριθμού των requests ανά εκτέλεση και του συνολικού χρόνου εκτέλεσης του drone. Επίσης παρατηρείτε μια αντιστρόφως ανάλογη σχέση με τις εικόνες ανά request και τον χρόνο εκτέλεσης του drone. Συμπερασματικά, με την μείωση των εικόνων ανά request αυξάνεται ο αριθμός των requests ανά εκτέλεση όπου και παρατηρείτε μία αυξητική τάση στον χρόνο εκτέλεσης του drone που αυτό συνεπάγεται και σε αύξηση του συνολικού χρόνου εκτέλεσης ολόκληρου του προγράμματος.



Experiment	Zcores & #GB & ML model & @Images	Base station time (sec)
1	2cores, 2GB, 2ο ML model, 1	139.97
2	4cores, 4GB, 2ο ML model, 1	63.95
3	2cores, 2GB, 1ο ML model, 1	85.51
4	4cores, 4GB, 1ο ML model, 1	45.94
5	4cores, 4GB, 2ο ML model, 2	72.81
6	4cores ,4GB, 2ο ML model, 3	65.27
7	4cores, 4GB, 2ο ML model, 4	67.22

Για το πιο πάνω πείραμα κρατάμε σταθερές τις παραμέτρους των network delays μεταξύ των τριών microservices και πιο συγκεκριμένα το network delay (round-trip time) μεταξύ των δύο συσκευών drone και base station είναι 5ms και το network delay (round-trip time) μεταξύ των δύο συσκευών base station και cloud είναι 100ms.

Τον μεγαλύτερο χρόνο του basestation μαζί με την επεξεργασία εικόνας που είναι μέρος του χρόνου, σημειώνετε όταν έχουμε 2 CPU cores με 2 GB μνήμη RAM και χρησιμοποιείτε το 2ο ML model.

1. Σε σύγκριση του πρώτου πειράματος με του δεύτερου, όπου χρησιμοποιείτε το δεύτερο ML model, παρατηρούμε ότι με τον διπλασιασμό των πυρήνων και της μνήμης ram ο χρόνος μειώνετε στον μισό περίπου του αρχικού. Από 139,97sec γίνετε 63.95sec.
2. Σε ίδια σύγκριση του τρίτου πειράματος με του τέταρτου όπου χρησιμοποιείτε το πρώτο ML model, παρατηρείτε και πάλι η μείωση του πρώτου χρόνου στον μισό, αφού ο χρόνος από 85.51sec γίνετε 45.94sec.

Βάση αυτών, συμπεραίνουμε ότι ο χρόνος λειτουργίας δεν εξαρτάτε αποκλειστικά από τον αλγόριθμο που χρησιμοποιείτε αλλά εξαρτάτε και από τα υπολογιστικά χαρακτηριστικά του infrastructure του base station.

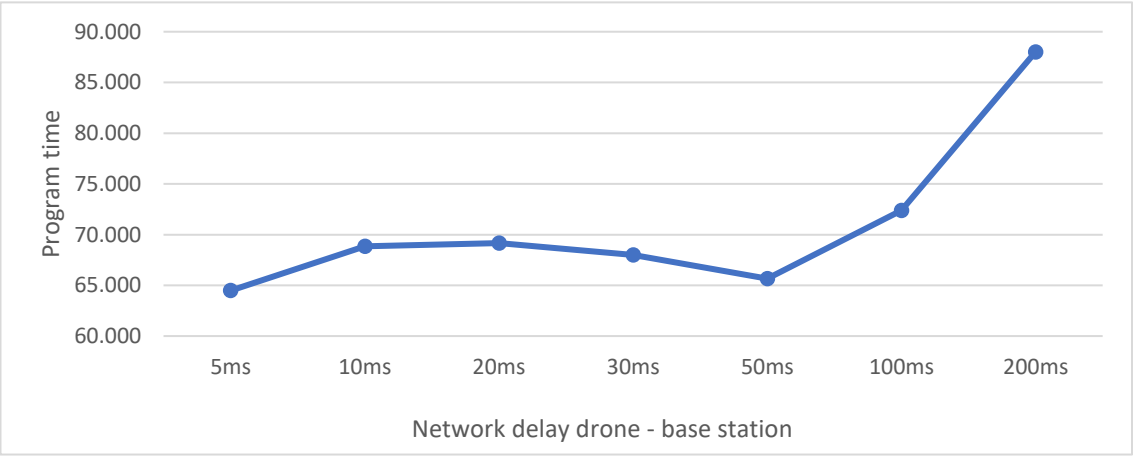
3. Στα πειράματα που κρατάμε σταθερούς τις παραμέτρους τον αριθμό των CPU cores, της μνήμης RAM και το δεύτερο ML model και μεταβάλουμε τον αριθμό εικόνων που αποστέλλονται σε κάθε segment από το drone (πειράματα 2,5,6,7) παρατηρούμε κάτι όχι και τόσο συνηθισμένο. Όπως αναμενόταν τον μικρότερο χρόνο τον έχει το πείραμα με την μία εικόνα. Αντίθετα όμως τον δεύτερο πιο μικρό χρόνο τον έχουν οι 3 εικόνες, τον αμέσως επόμενο οι 4 και τον μεγαλύτερο χρόνο του base station τον έχουν οι 2 εικόνες.

Cloud execution time

Experiment	Cloud Time (sec)	Experiment	Cloud Time (sec)
1	0.024706125259	6	0.017634510994
2	0.012591600418	7	0.019204616500
3	0.023541331291	8	0.016892512333
4	0.007694244385	9	0.023968816000
5	0.008229255333	10	0.031468153000

Αυτές οι τιμές, αντιπροσωπεύουν τον χρόνο εκτέλεσης του cloud στα πρώτα 10 πειράματα. Με αυτές, συμπεράνουμε ότι ο χρόνος εκτέλεσης του cloud είναι αμελητέος ή ελάχιστος σε σύγκριση με τις άλλες μετρικές. Αυτό υποδηλώνει ότι η εμπλοκή του νέφους στην εκτέλεση του προγράμματος είχε ελάχιστη έως μηδαμινή επίδραση στο συνολικό χρόνο εκτέλεσης του προγράμματος.

Σύγκριση network delay μεταξύ drone και base station



Να σημειωθεί ότι πέραν από το υλοποιημένο base station που υπάρχει σαν υποδομή, υλοποιείτε και ένα base station στο cloud το οποίο χρησιμοποιείται στο πείραμα στα δύο τελευταία πειράματα. Επιπρόσθετα, το network delay αντικατοπτρίζει και την απόσταση που έχουν οι δύο συσκευές μεταξύ τους κατά την ώρα της μεταφοράς των δεδομένων. Για το πιο πάνω πείραμα, το network delay μεταξύ base station – cloud δεν μας ενδιαφέρει έτσι, κρατάμε σταθερές τις παραμέτρους του base station και πιο συγκεκριμένα το ML model που χρησιμοποιείτε είναι το δεύτερο, τα CPU cores είναι 4, η μνήμη ram είναι 4Gb και ο αριθμός εικόνων που αποστέλλετε σε κάθε request από το drone στο basestation είναι μία. Στο συγκεκριμένο πείραμα μεταβάλουμε το network delay μεταξύ drone και basestation για να δούμε αν το delay αυτό επηρεάζει την διαδικασία καταμετρήσεις των οχημάτων από τις εικόνες αλλά όχι για κάθε μια εικόνα ξεχωριστά, αλλά μέχρι να επεξεργαστεί όλο το σύνολο των εικόνων που έχει το drone να αποστείλει. Βάση το πιο πάνω σχήμα, παρατηρούμε ότι το network delay μεταξύ drone και basestation όταν κυμαίνεται από 5ms μέχρι και 100ms ο χρόνος της διαδικασίας καταμέτρησης παραμένει σχετικά ο ίδιος με μόνο κάποιες μικρές αυξομειώσεις της κλίμακας των 10sec, μεταξύ των 64sec και 72sec. Αντίθετα όμως, όταν το drone στέλνει τις εικόνες στο basestation στο cloud, δηλαδή όταν το network delay ξεπεράσει τα 100ms η διαδικασία καταμέτρησης επηρεάζεται αρνητικά με τον χρόνο να αρχίζει να αυξάνετε απότομα όπως βλέπουμε στα δύο τελευταία σημεία της γραφικής.

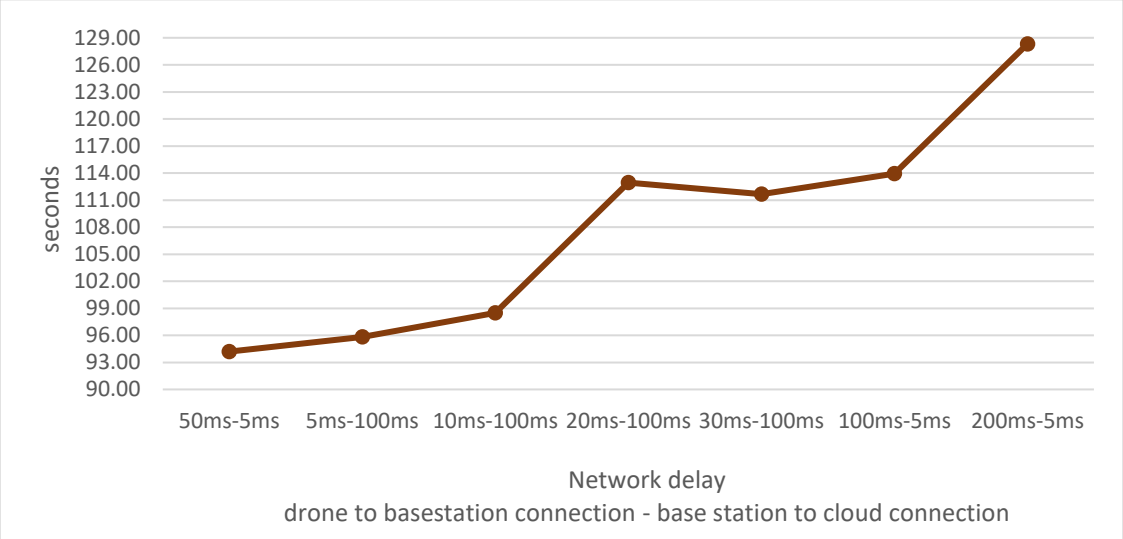
Σύγκριση network delay μεταξύ base station και cloud

Experiment	Yms B->C, @Images	Cloud Time
1	100ms, 1	1.260452269684
2	5ms, 1	0.127711841867
3	100ms, 2	1.331473350525
4	100ms, 3	1.323522150517
5	100ms, 4	1.288117000035

Ο χρόνος αποστολής των εικόνων από το basestation και η επεξεργασία των αποτελεσμάτων στο cloud είναι ανάλογη της αυξομείωσης του network delay της σύνδεσης τους. Παρατηρούμε όμως με όλες τις άλλες παραμέτρους σταθερές, ότι με την μείωσή των 100ms network delay στο 1/20 τους, δηλαδή στα 5ms, ο χρόνος του αποστολής των δεδομένων προς το cloud και απάντησης του cloud πίσω, δεν πέφτει στο 1/20 αλλά στο 1/10 και από 1.292033475169sec κατά μέσο όρο πάει στα

0.127711841867 sec. Αυτό μπορεί όμως να οφείλετε και στις καθυστερήσεις που έχουν οι αλγόριθμοι ξεχωριστά.

Επιπρόσθετα, ο χρόνος της διαδικασίας πιο πάνω με 100ms καθυστέρηση στην σύνδεση base station – cloud φαίνεται να μην επηρεάζει κάτι αν το basestation λαμβάνει από το drone και επεξεργάζεται εικόνες πάνω από μία σε κάθε request με τον χρόνο της διαδικασίας να παραμένει στα τρία πειράματα 4, 5, 6 γύρο στα 1.314 sec κατά μέσο όρο.



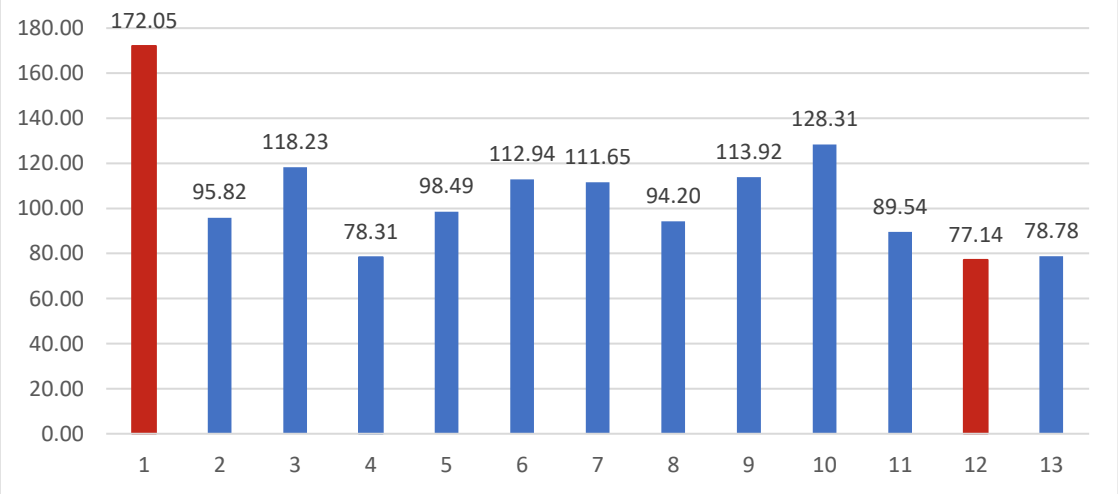
Experiment	Network delay Drone to basestation connection – base station to cloud connection	Total time (sec)
1	50ms D->B - 5ms B->C	94.20
2	5ms D->B - 100ms B->C	95.82
3	10ms D->B - 100ms B->C	98.49
4	20ms D->B - 100ms B->C	112.94
5	30ms D->B - 100ms B->C	111.65
6	100ms D->B - 5ms B->C	113.92
7	200ms D->B - 5ms B->C	128.31

Για το πιο πάνω πείραμα, κρατήθηκαν σταθερές οι παράμετροι του base station και πιο συγκεκριμένα το ML model που χρησιμοποιείτε είναι το δεύτερο, τα CPU cores είναι 4, η μνήμη RAN είναι 4Gb και ο αριθμός εικόνων που αποστέλλετε σε κάθε request από το drone στο basestation είναι μία. Παρατηρούμε ότι στον συνολικό χρόνο εκτέλεσης του προγράμματος υπάρχει μια αυξητική τάση του χρόνου ανάλογα με την αύξηση του network delay (round-trip time) από το drone στο base station και από το base station στο cloud. Έτσι συμπεραίνουμε ότι το συνολικό network delay που υπάρχει κατά την εκτέλεση του προγράμματος είναι ανάλογο στον συνολικό χρόνο εκτέλεσης του προγράμματος.

Network delay Drone to basestation connection - base station to cloud connection	Total time
5ms - 100ms	95.82 sec
100ms - 5ms	113.92 sec

Για το πιο πάνω πείραμα, κρατήθηκαν σταθεροί οι παράμετροι του base station και πιο συγκεκριμένα το ML model που χρησιμοποιείτε είναι το δεύτερο, τα CPU cores είναι 4, η μνήμη RAM είναι 4Gb και ο αριθμός εικόνων που αποστέλλετε σε κάθε request από το drone στο basestation είναι μία. Από τα αποτελέσματα που προέκυψαν από τον πίνακα, παρατηρούμε ότι όταν η μικρή καθυστέρηση είναι μεταξύ της σύνδεσης drone – base station και η μεγάλη μεταξύ base station – cloud, ο συνολικός χρόνος εκτέλεσης του προγράμματος είναι πιο μικρός από το να είναι η μεγάλη καθυστέρηση μεταξύ της σύνδεσης drone – base station και η μικρή μεταξύ base station – cloud. Ένας από τους κύριους παράγοντες που συμβαίνει αυτό είναι ότι κατά την εκτέλεση του προγράμματος, για ένα μεγάλο ποσοστό επαναλήψεων των εκτελέσεων, το drone στέλνει request παραπάνω από μια φορά ανά εκτέλεση εκτός και αν σταλούν όλες οι εικόνες σε ένα request ώστε να μετρήσει ο χρόνος καθυστέρησης μεταξύ drone base station μια φορά, αλλιώς η συγκεκριμένη καθυστέρηση θα πολλαπλασιάζετε με τον αριθμό των requests. Και όπως γνωρίζουμε, όσο πιο μεγάλος είναι ο αριθμός που τον πολλαπλασιάζουμε με τον αριθμό των requests ανά εκτέλεση τότε θα είναι και ανάλογα μεγάλο το χρονικό διάστημα εκτέλεσης ολόκληρου του προγράμματος.

Total time



A/A	Xms, Yms, ML model, Zcores, #GB, @Images	Times	A/A	Xms, Yms, ML model, Zcores, #GB, @Images	TImes
1	5ms D->B, 100ms B->C, 2o ML model, 2cores, 2GB, 1	172.05	8	50ms D->B, 5ms B->C, 2o ML model, 4cores, 4GB, 1	94.20
2	5ms D->B, 100ms B->C, 2o ML model, 4cores, 4GB, 1	95.82	9	100ms D->B, 5ms B->C, 2o ML model, 4cores, 4GB, 1	113.92
3	5ms D->B, 100ms B->C, 1o ML model, 2cores, 2GB, 1	118.23	10	200ms D->B, 5ms B->C, 2o ML model, 4cores, 4GB, 1	128.31
4	5ms D->B, 100ms B->C, 1o ML model, 4cores, 4GB, 1	78.31	11	5ms D->B, 100ms B->C, 2o ML model, 4cores, 4GB, 2	89.54
5	10ms D->B, 100ms B->C, 2o ML model, 4cores, 4GB, 1	98.49	12	5ms D->B, 100ms B->C, 2o ML model, 4cores, 4GB, 3	77.14
6	20ms D->B, 100ms B->C, 2o ML model, 4cores, 4GB, 1	112.94	13	5ms D->B, 100ms B->C, 2o ML model, 4cores, 4GB, 4	78.78
7	30ms D->B, 100ms B->C, 2o ML model, 4cores, 4GB, 1	111.65			

Το εύρος των χρόνων του προγράμματος κυμαίνεται από το 77,14s μέχρι και 172.05s. Παρατηρούμε, ότι ένα σύνολο από μικρότερους χρόνους εκτέλεσης του συνολικού προγράμματος επιτυγχάνονται όταν το drone αποστέλλει παραπάνω από μια εικόνα ανά request και ταυτόχρονα έχουμε και την μικρότερη διάρκεια εκτέλεση όλου του προγράμματος με 77,14sec όταν έχουμε network round-trip delay 5 ms από drone σε basestation, 100 ms από base station σε cloud, χρησιμοποιούμε το δεύτερο ML model με 4 CPU cores και 4 GB RAM στο infrastructure του basestation, και αποστέλλουμε 3 εικόνες σε κάθε request. Ο μεγαλύτερος χρόνος επιτυγχάνεται στο πρώτο πείραμα όπου το network round-trip delay και στις δυο συνδέσεις είναι το ίδιο με πιο πάνω αλλά το infrastructure του base station έχει 2 CPU cores και 2GB RAM και αποστέλλουμε μία εικόνα σε κάθε request.

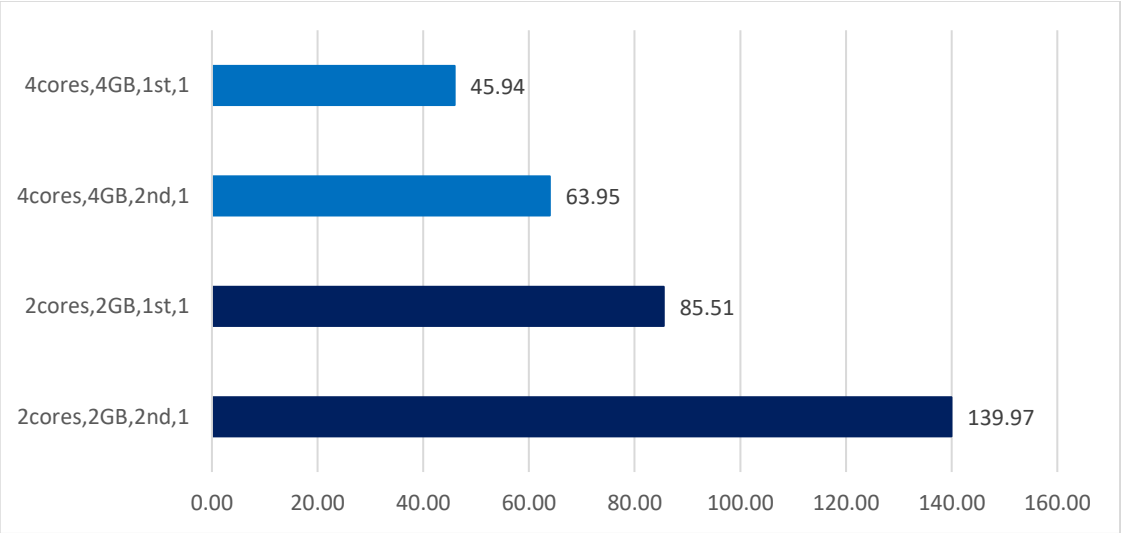
Με βάση τις παρατηρήσεις από τα πειράματα, μπορούν να εξαχθούν διάφορα συμπεράσματα. Πρώτον, η ελαχιστοποίηση της καθυστέρησης round-trip time του δικτύου, όπως με καθυστέρηση 5 ms από το drone στο σταθμό βάσης και 100 ms από το σταθμό βάσης στο νέφος, βελτιώνει σημαντικά την απόδοση του προγράμματος, όπως αποδεικνύεται από το συντομότερο χρόνο εκτέλεσης 77,14 δευτερολέπτων που επιτεύχθηκε υπό αυτές τις συνθήκες. Δεύτερον, η διάθεση επαρκών πόρων υλικού, συγκεκριμένα η χρήση 4 πυρήνων CPU και 4GB RAM στην υποδομή του σταθμού βάσης, οδηγεί σε συντομότερους χρόνους εκτέλεσης σε σύγκριση με μια διαμόρφωση με 2 πυρήνες CPU και 2GB RAM. Τέλος, η αποστολή πολλαπλών εικόνων σε κάθε αίτηση οδηγεί σε συντομότερους συνολικούς χρόνους εκτέλεσης, συμβάλλει στην ταχύτερη

επεξεργασία και στη μείωση των χρόνων εκτέλεσης, όπως αποδεικνύεται από το πείραμα με τρεις εικόνες ανά αίτηση σε σχέση με το σενάριο με μία εικόνα ανά αίτηση. Αυτό υποδεικνύει την αποτελεσματικότητα που επιτυγχάνεται από τη συγκέντρωση και την επεξεργασία πολλαπλών εικόνων μαζί.

6.2 Πείραμα 2

Η υποδομή του σταθμού βάσης ενσωματώνει δύο διαφορετικούς αλγορίθμους ανίχνευσης αντικειμένων με μοντέλο μηχανικής μάθησης (ML) για εργασίες όρασης υπολογιστή. Ένα αξιοσημείωτο χαρακτηριστικό αυτής της εγκατάστασης είναι η δυνατότητα δυναμικής επιλογής του αλγορίθμου του μοντέλου ML κατά τη διάρκεια της εκτέλεσης. Αυτή η προσέγγιση διευκολύνει μια ολοκληρωμένη ανάλυση τόσο της ακρίβειας όσο και της αποδοτικότητας του προγράμματος, επιτρέποντάς μας να βελτιστοποιήσουμε αποτελεσματικά την απόδοσή του.

Πρωταρχικός στόχος, είναι να αξιολογηθεί και να συγκριθεί η ακρίβεια και η καθυστέρηση του προγράμματος κατά τη χρήση κάθε αλγορίθμου. Έχοντας την ευελιξία να επιλέγουμε μεταξύ των μοντέλων ML κατά την εκτέλεση του προγράμματος, μπορούμε να αξιολογήσουμε τις επιμέρους επιδόσεις τους και να καθορίσουμε ποιο από τα δύο παρέχει καλύτερα αποτελέσματα τόσο από πλευράς ακρίβειας όσο και από πλευράς αποδοτικότητας. Επιπρόσθετα αποσκοπεί στον προσδιορισμό του αλγορίθμου του μοντέλου ML που αποδίδει καλύτερα βάσει συγκεκριμένων απαιτήσεων.



Experiment	ZCores & #GB & ML model & @Images	Base station time
1	2cores,2GB,2nd,1	139.97
2	2cores,2GB,1st,1	85.51
3	4cores,4GB,2nd,1	63.95
4	4cores,4GB,1st,1	45.94

Όπως βλέπουμε και από τα αποτελέσματα πιο πάνω, παρατηρούμε ότι ο πρώτος αλγόριθμος και στα 2 CPU cores, 2GB RAM αλλά και στα 4 CPU cores, 4GB RAM έχει αρκετά πιο μικρό χρονικό διάστημα εκτέλεσης από τον δεύτερο αλγόριθμο. Ο πρώτος αλγόριθμος, υπερτερεί του δεύτερου αλγορίθμου όσον αφορά το χρόνο εκτέλεσης. Ο πρώτος αλγόριθμος, επιδεικνύει σημαντικά μικρότερους χρόνους εκτέλεσης, περίπου το μισό του χρόνου εκτέλεσης του δεύτερου αλγορίθμου, ανεξάρτητα από τη διαμόρφωση του υλικού. Αυτό υποδηλώνει ότι ο πρώτος αλγόριθμος είναι αποδοτικότερος και αποδίδει καλύτερα από άποψη χρονικής απόδοσης σε σύγκριση με τον δεύτερο αλγόριθμο. Επομένως, όταν ο χρόνος εκτέλεσης θεωρείται κρίσιμος παράγοντας, ο πρώτος αλγόριθμος θα πρέπει να προτιμάται έναντι του δεύτερου αλγορίθμου για βέλτιστη απόδοση.

Accuracy

Όσον αφορά την ακρίβεια, τόσο το YOLOv4 όσο και το RetinaNet έχουν επιδείξει ισχυρές επιδόσεις. Το YOLOv4, είναι γνωστό για την ανταγωνιστική ακρίβειά του σε μεγάλα αντικείμενα και τη συνολική απόδοση ανίχνευσης, ενώ το RetinaNet υπερέχει στην ανίχνευση μικρών αντικειμένων και στο χειρισμό προβλημάτων ανισορροπίας κλάσεων. Η συγκεκριμένη ακρίβεια που επιτυγχάνεται από κάθε αλγόριθμο εξαρτάται από παράγοντες όπως η ποιότητα του συνόλου δεδομένων, η διαμόρφωση της εκπαίδευσης και ο συντονισμός των υπερπαραμέτρων.

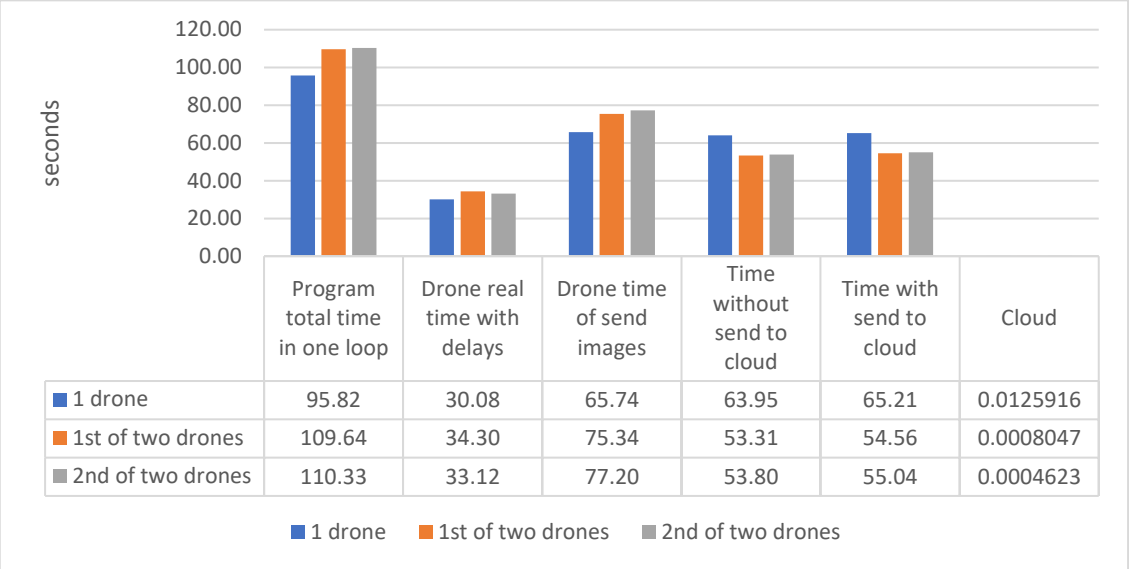
Τελικά, η επιλογή μεταξύ του YOLOv4 και του RetinaNet εξαρτάται από τις απαιτήσεις της εφαρμογής. Ο YOLOv4 προσφέρει ανίχνευση σε πραγματικό χρόνο με καλή συνολική ακρίβεια, καθιστώντας τον κατάλληλο για σενάρια που δίνουν προτεραιότητα στην ταχύτητα. Από την άλλη πλευρά, το RetinaNet αποτελεί ισχυρή επιλογή όταν η ακριβής ανίχνευση μικρών αντικειμένων και η αντιμετώπιση της ανισορροπίας των κλάσεων είναι κρίσιμοι παράγοντες

Ο πρώτος αλγόριθμος είναι ο YOLOv4 και επιτυγχάνει 80,77% mAP accuracy με real-time speed στα 65fps. Παρόμοια ο δεύτερος αλγόριθμος που είναι ο RetinaNet

επιτυγχάνει ένα ποσοστό accuracy στα 82,89% mAP αλλά αντίθετα με τον YOLOv4 δεν μπορεί να υποστηρίξει τόσο την real-time επεξεργασία αφού το real-time speed του κυμαίνεται στα 17 fps. Έτσι, ο YOLOv4 έχει και τους μικρότερους χρόνους λόγω του real-time speed.

6.3 Πείραμα 3

Για να ενισχυθεί η πολυπλοκότητα και η κλίμακα του πειράματος, ενσωματώθηκαν περισσότερα από ένα αντίγραφα drone τα οποία εκτελούν ταυτόχρονα τον κώδικα του drone και αποστέλλουν συλλογικά εικόνες στον σταθμό βάσης για επεξεργασία. Αυτή η διάταξη επιτρέπει την αξιολόγηση της επεκτασιμότητας του προγράμματος και της ικανότητάς του να διαχειρίζεται σημαντικούς όγκους δεδομένων σε πραγματικό περιβάλλον.
Στόχος, είναι να προσδιοριστεί πόσο καλά το πρόγραμμα διαχειρίζεται μεγαλύτερους όγκους δεδομένων και απαιτήσεις επεξεργασίας όταν πολλά drones μεταδίδουν εικόνες στο σταθμό βάσης. Το πείραμα αξιολογεί την ικανότητα του προγράμματος να επεξεργάζεται αποτελεσματικά το αυξημένο φορτίο δεδομένων που δημιουργείται και εξετάζει κατά πόσον το πρόγραμμα μπορεί να διατηρήσει αποδεκτά επίπεδα απόδοσης και να χειριστεί αποτελεσματικά τις πρόσθετες υπολογιστικές απαιτήσεις. Με την προσομοίωση ενός πιο ρεαλιστικού σεναρίου με πολλαπλά μη επανδρωμένα αεροσκάφη, το πείραμα αποσκοπεί στη μίμηση ενός πραγματικού περιβάλλοντος. Αξιολογεί τις επιδόσεις του προγράμματος υπό συνθήκες που μοιάζουν με πραγματικά επιχειρησιακά σενάρια, παρέχοντας πληροφορίες σχετικά με την ικανότητά του να χειρίζεται πρακτικές πολυπλοκότητες και φόρτους εργασίας. Επιπλέον, επιδιώκει τον εντοπισμό πιθανών περιοχών για βελτιστοποίηση. Στοχεύει στην ανακάλυψη σημείων συμφόρησης, στη βελτίωση της αποδοτικότητας ή σε στρατηγικές κατανομής πόρων που μπορούν να βελτιώσουν τη συνολική απόδοση και την επεκτασιμότητα του προγράμματος.



	Program total time in one loop	Drone real time with delays	Drone time of send images	Time without send to cloud	Time with send to cloud	Cloud
1 drone	95.82	30.08	65.74	63.95	65.21	0.0125916
1st of two drones	109.64	34.30	75.34	53.31	54.56	0.0008047
2nd of two drones	110.33	33.12	77.20	53.80	55.04	0.0004623

Για το πιο σχήμα, διατηρήθηκαν σταθερές οι παρακάτω παράμετροι. Το network delay (round-trip-time) μεταξύ drone και basestation είναι 5ms, το network delay (round-trip-time) μεταξύ basestation και cloud είναι 100ms, το ML model που χρησιμοποιείτε είναι το δεύτερο, τα CPU cores είναι 4, η μνήμη ram είναι 4Gb και ο αριθμός εικόνων που αποστέλλετε από το drone στο basestation είναι μία ανά request.

Μεταξύ των τριών drone, το drone με τον μικρότερο συνολικό χρόνο σε μια εκτέλεση είναι το πρώτο drone, το οποίο κατέγραψε χρόνο 95,82 sec. Συγκριτικά, τα δύο drones που αποστέλλουν μαζί τις εικόνες επιτυγχάνουν τον ίδιο χρόνο εκτέλεσης. Ο χρόνος που εκτελούνται οι υπόλοιπες διεργασίες του drone πέραν της διεργασίας που αποστέλλει τις εικόνες είναι και στα 3 drones σχεδόν ο ίδιος με πάλι λίγο μικρότερο χρόνο εκτέλεσης το drone που αποστέλλει μόνο του τις εικόνες. Επίσης και στην διεργασία που αποστέλλει τις εικόνες το drone το οποίο τις αποστέλλει μόνο του σημειώνει κατά 10sec μικρότερο χρόνο από τα άλλα δυο. Ωστόσο, το drone που στέλνει εικόνες μόνο του επιτυγχάνει μικρότερο χρόνο εκτέλεσης κατά τη διαδικασία αποστολής εικόνων, υποδεικνύοντας την πιθανή αποδοτικότητα που κερδίζεται από την αποστολή εικόνων μεμονωμένα.

Στους χρόνους του base station και του cloud παρατηρούμε ότι οι χρόνοι των δυο drone τα οποία αποστέλλουν μαζί τις εικόνες τους, είναι μικρότεροι από το πρώτο drone. Και συγκεκριμένα ο χρόνος του base station να επεξεργαστεί τις εικόνες με το 2ο ML model και να αποστείλει τα δεδομένα στο cloud, και ο χρόνος του cloud να παραλάβει τα δεδομένα να τα επεξεργαστεί και να απαντήσει πίσω στο basestation είναι μικρότεροι για

το σενάριο με τα δυο drones κατά 10sec από το πρώτο drone. Αυτό υποδηλώνει ότι η ταυτόχρονη αξιοποίηση πολλών drone μπορεί να μειώσει τους χρόνους επεξεργασίας και επικοινωνίας μεταξύ του σταθμού βάσης και του νέφους, οδηγώντας σε συνολική βελτιστοποίηση του χρόνου

Με την εκτέλεση του πειράματος, και πιο συγκεκριμένα στην πρώτη προσπάθεια των 2 drones να στείλουν τις εικόνες τους μαζί, παρουσιάστηκε ένα πρόβλημα το οποίο εμπόδισε το δεύτερο drone να παραλάβει τα response από το base station και συνάμα να μην καταφέρει να καταγράψει τους χρόνους εκτέλεσης της διεργασίας αυτής. Παρόλα αυτά οι εικόνες αποστάλθηκαν και παραλήφθηκαν στο base station όπου έγινε η επεξεργασία τους και η παραγωγή των δεδομένων από τον ML model αλγόριθμο.

Μετά από βαθιά ανάλυση και σύγκριση των σχετικών αποτελεσμάτων του πειράματος με το κομμάτι των αποτελεσμάτων του δεύτερου drone που δεν καταγράφηκε, ανακαλύφθηκε ότι το πρόβλημα ήταν οι μεγάλοι χρόνοι εκτέλεσης της συγκεκριμένης διεργασίας και στο δεύτερο drone ο χρόνος αναμονής του response από το base station ξεπερνούσε το όριο του timeout του request που στέλνει τις εικόνες. Συγκεκριμένα, το πρόβλημα προήλθε διότι το base station δεν επεξεργάζεται παράλληλα εικόνες έτσι η ταυτόχρονη αποστολή εικόνων από τα 2 drones επεξεργάζεται σειριακά. Έτσι, όταν οι εικόνες του πρώτου drone επεξεργάζονταν το timeout του δεύτερου drone με αποτέλεσμα να ξεπεραστεί το προκαθορισμένο timeout και να μην αποθηκευτούν τα δεδομένα.

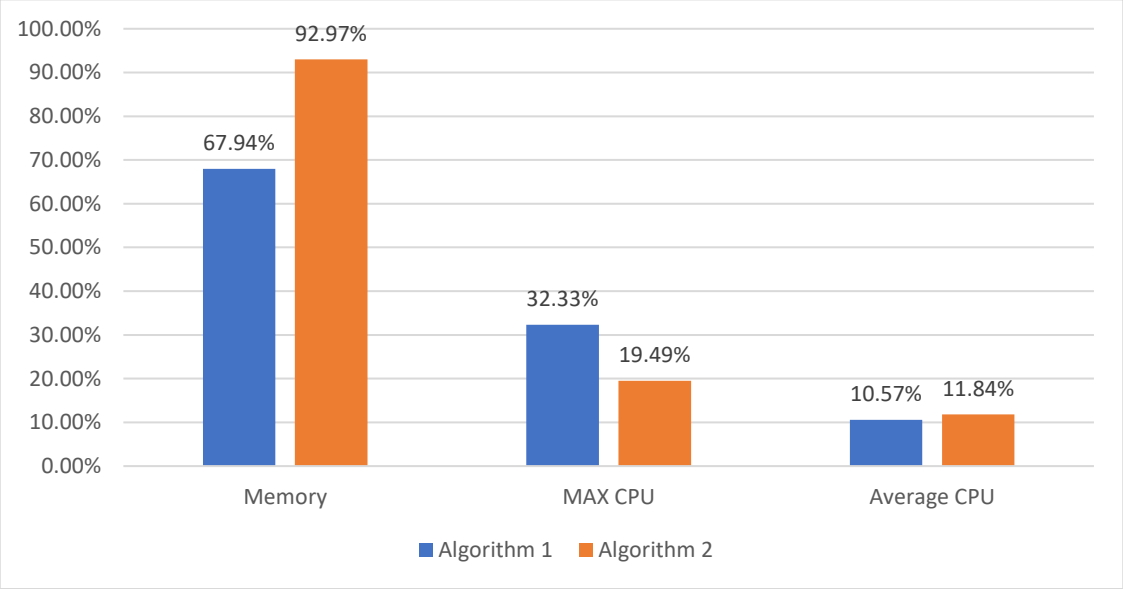
Για το συγκεκριμένο πρόβλημα μπορούν να πραγματοποιηθούν κάποιες αλλαγές είτε σε βελτιστοποίηση κώδικα είτε στο infrastructure του base station. Και πιο συγκεκριμένα μπορούμε να αυξήσουμε το timeout των drones όταν περιμένουν response από το base station ή να προσθέσουμε παραπάνω resources στο basestation, αντίστοιχα.

6.4 Πείραμα 4

Προκειμένου να προσδιοριστεί το κατάλληλο υλικό για τα τρία microservices του προγράμματος, είναι ζωτικής σημασίας να αξιολογηθεί η μεμονωμένη χρήση μνήμης και CPU, καθώς και η συνολική απόδοση του συστήματος.

Το πείραμα αυτό, αποσκοπεί στην απάντηση ερωτημάτων που σχετίζονται με την απόδοση και τη χρήση των πόρων διαφορετικών διαμορφώσεων υλικού στο πλαίσιο της εκτέλεσης συγκεκριμένων αλγορίθμων ή μικρουπηρεσιών. Επίσης, βοηθά στον εντοπισμό τυχόν πιθανών σημείων συμφόρησης ή προβλημάτων απόδοσης,

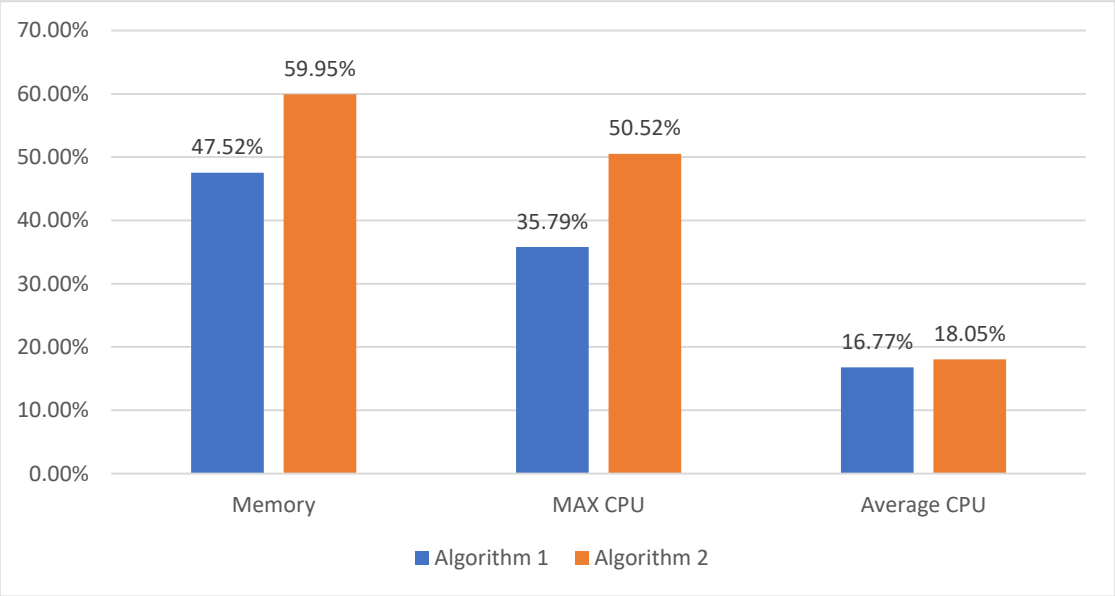
διασφαλίζοντας τη βέλτιστη λειτουργία του συστήματος. Συγκρίνοντας τα αποτελέσματα που προκύπτουν από την εκτέλεση των αλγορίθμων σε διαφορετικές διαμορφώσεις υλικού, το πείραμα επιδιώκει να προσδιορίσει την επίδραση των διαφορετικών πόρων υλικού, όπως οι πυρήνες CPU και η μνήμη RAM, σε παράγοντες όπως η χρήση της μνήμης, η χρήση της CPU και η συνολική απόδοση του συστήματος.



Σε σύγκριση των δύο ML model αλγορίθμων που τρέχουν στο base station με 2 CPU cores και 2GB RAM παρατηρούμε ότι το ο δεύτερος αλγόριθμος επιτυγχάνει 92,97% memory utilization ενώ αντίστοιχα ο πρώτος επιτυγχάνει 67,94% memory utilization. Κατά τις μετρήσεις του CPU performance ο πρώτος αλγόριθμος φτάνει μέχρι και το 32,33% χρήσης της CPU ενώ αντίθετα ο δεύτερος αλγόριθμος φτάνει μόνο μέχρι το 19,49% αλλά ο μέσος όρος και των δυο αλγορίθμων στην καταμέτρηση του CPU performance είναι 10,57% και 11,84% αντίστοιχα.

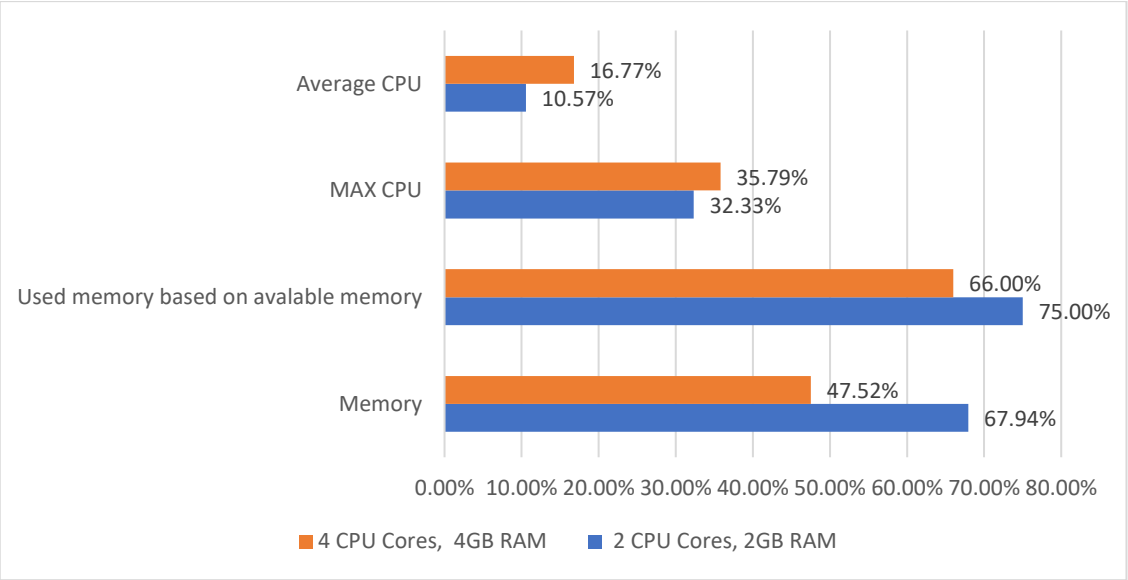
Ο πρώτος αλγόριθμος, επιτυγχάνει χαμηλότερη χρήση μνήμης 67,94% σε σύγκριση με την υψηλή χρήση 92,97% του δεύτερου αλγορίθμου. Η χαμηλότερη χρήση μνήμης του πρώτου αλγορίθμου υποδηλώνει αποτελεσματικότερη διαχείριση της μνήμης, μειώνοντας την πιθανότητα εμφάνισης προβλημάτων που σχετίζονται με τη μνήμη και βελτιώνοντας ενδεχομένως τη συνολική απόδοση. Παρόλο που ο δεύτερος αλγόριθμος έχει ελαφρώς υψηλότερο μέσο αριθμό επιδόσεων CPU (11,84% σε σύγκριση με 10,57% για τον πρώτο αλγόριθμο), αξίζει να σημειωθεί ότι ο πρώτος αλγόριθμος φτάνει σε υψηλότερη μέγιστη χρήση CPU, 32,33%, σε σύγκριση με μόλις 19,49% για τον δεύτερο

αλγόριθμο. Αυτό δείχνει ότι ο πρώτος αλγόριθμος είναι σε θέση να χρησιμοποιεί τους διαθέσιμους πόρους της CPU πιο αποτελεσματικά όταν χρειάζεται. Ωστόσο, και οι δύο αλγόριθμοι εξακολουθούν να περνούν σημαντικό χρονικό διάστημα με χαμηλότερη χρήση της CPU λόγω περιόδων αδράνειας.



Σε σύγκριση των δύο ML model αλγορίθμων που τρέχουν στο base station με 4 CPU cores και 4GB RAM παρατηρούμε ότι το ο δεύτερος αλγόριθμος επιτυγχάνει 59,95% memory utilization ενώ αντίστοιχα ο πρώτος επιτυγχάνει 47,52% memory utilization. Αντίστοιχα το CPU performance του δεύτερου αλγόριθμος φτάνει μέχρι και το 50,52% χρήσης της CPU ενώ αντίθετα ο πρώτος αλγόριθμος φτάνει μέχρι το 35,79% αλλά ο μέσος όρος και των δυο αλγορίθμων στην καταμέτρηση του CPU performance είναι 18,05% και 16,77% αντίστοιχα.

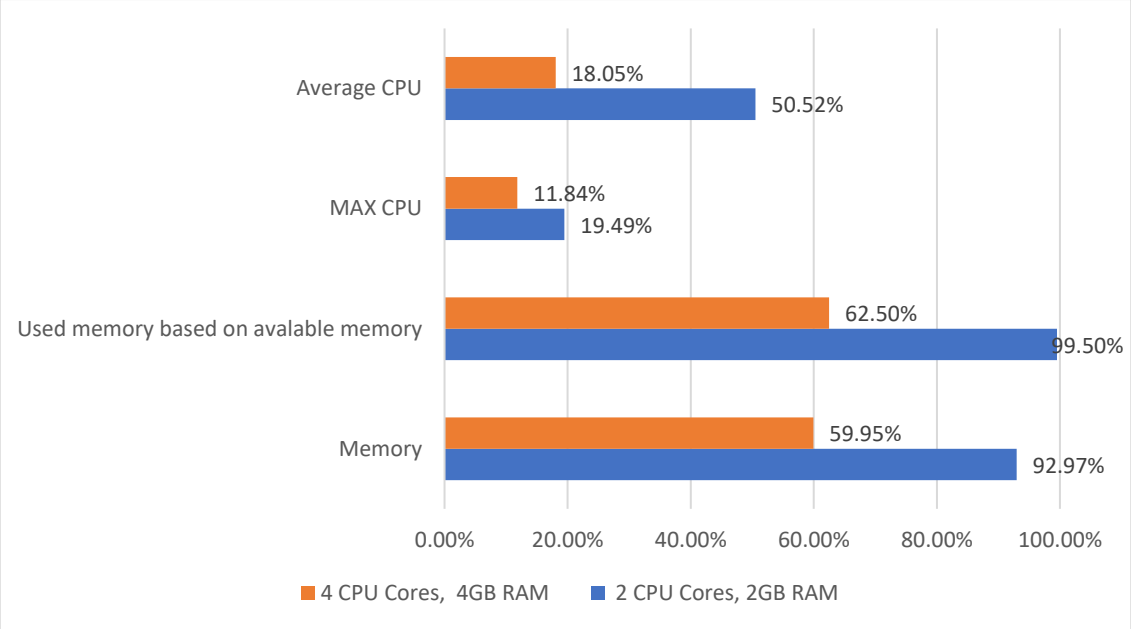
Συνολικά, με βάση τις παρεχόμενες μετρήσεις, ο δεύτερος αλγόριθμος φαίνεται να διαχειρίζεται καλύτερα τους διαθέσιμους πόρους (πυρήνες CPU και RAM), χρησιμοποιώντας τους πιο αποτελεσματικά και επιτυγχάνοντας υψηλότερα επίπεδα χρήσης. Ωστόσο, αξίζει να σημειωθεί ότι και οι δύο αλγόριθμοι εξακολουθούν να περνούν σημαντικό χρονικό διάστημα με χαμηλότερη χρήση της CPU λόγω περιόδων αδράνειας, υποδεικνύοντας πιθανές ευκαιρίες για περαιτέρω βελτιστοποίηση και βελτίωση της χρήσης των πόρων.



Κρατώντας σταθερό τον αλγόριθμο 1 και συγκρίνοντας τις αποδόσεις του αλγόριθμου στο base station infrastructure σε 2 CPU Cores με 2GB RAM και 4 CPU Cores με 4GB RAM παρατηρούμε ότι όταν ο αλγόριθμος εκτελείτε στα 2 CPU Cores με 2GB RAM το memory utilization είναι 20,42% περισσότερο από όταν εκτελείτε στα 4 CPU Cores με 4GB RAM. Και πιο συγκριμένα είναι στα 67,94% και 47,52% αντίστοιχα. Επίσης, στο δεύτερο πείραμα το memory utilization είναι πιο σταθερές οι μετρήσεις του με πολύ μικρές αυξομειώσεις σε σχέση με το πρώτο πείραμα. Επιπρόσθετα, στο πρώτο πείραμα ο αλγόριθμος χρησιμοποιεί κατά μέσο όρο 1,5 GB από τα 2GB (75%) ενώ στο δεύτερο χρησιμοποιεί κατά μέσο όρο μόνο τα 2GB από τα 4GB (66%). Κατά τις μετρήσεις του CPU performance στο πρώτο πείραμα φτάνει το 32.33% χρήσης της CPU με μέσο όρο 10.57%, ενώ στο δεύτερο πείραμα φτάνει το 35.79% με μέσο όρο 16.77%.

Η χαμηλότερη χρήση μνήμης στην υποδομή με 4 CPU cores και 4GB RAM υποδηλώνει ότι αποτελεί καλύτερη επιλογή πόρων για τον αλγόριθμο. Υποδεικνύει ότι ο αλγόριθμος μπορεί να λειτουργήσει πιο αποτελεσματικά και ενδεχομένως να έχει βελτιωμένη απόδοση λόγω της διαθεσιμότητας πρόσθετης μνήμης. Το δεύτερο πείραμα στην υποδομή με 4 πυρήνες CPU και 4GB RAM καταδεικνύει πιο σταθερή χρήση της μνήμης με ελάχιστες διακυμάνσεις σε σύγκριση με το πρώτο πείραμα. Η σταθερότητα στη χρήση της μνήμης υποδηλώνει καλύτερο έλεγχο και συνέπεια στη χρήση των πόρων, γεγονός που είναι επωφελές για την απόδοση του αλγορίθμου και τη συνολική σταθερότητα του συστήματος.

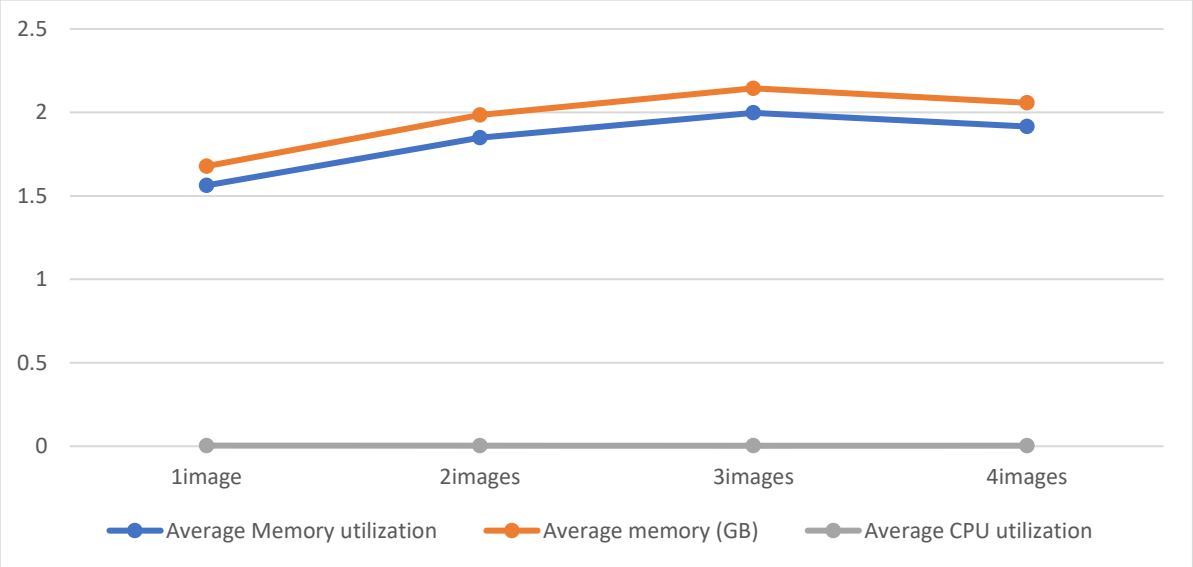
Τα αποτελέσματα χρήσης της μνήμης RAM βάση της διαθέσιμης μνήμης υποδεικνύουν ότι ο αλγόριθμος έχει υψηλότερες απαιτήσεις μνήμης στο πρώτο πείραμα και χρησιμοποιεί μεγαλύτερο μέρος των διαθέσιμων πόρων. Επομένως, η υποδομή με 4 πυρήνες CPU και 4GB RAM παρέχει μεγαλύτερο περιθώριο και είναι καταλληλότερη για να διαχειριστεί τις απαιτήσεις μνήμης του αλγορίθμου. Παρόλο που το δεύτερο πείραμα παρουσιάζει ελαφρώς υψηλότερη χρήση της CPU, η διαφορά δεν είναι σημαντική. Αυτό σημαίνει ότι και οι δύο διαμορφώσεις υποδομής μπορούν να χειριστούν αποτελεσματικά τις απαιτήσεις του αλγορίθμου σε CPU.



Κρατώντας σταθερό τον αλγόριθμο 2 και συγκρίνοντας τις αποδόσεις του αλγόριθμου στο base station infrastructure σε 2 CPU Cores με 2GB RAM και 4 CPU Cores με 4GB RAM παρατηρούμε ότι όταν ο αλγόριθμος εκτελείτε στα 2 CPU Cores με 2GB RAM το memory utilization είναι 33,01% περισσότερο από όταν εκτελείτε στα 4 CPU Cores με 4GB RAM. Και πιο συγκριμένα είναι στα 92,97% και 59,95% αντίστοιχα. Επίσης, και σε αυτές τι μετρήσεις το memory utilization του δεύτερου πειράματος είναι πιο σταθερές με πολύ μικρές αυξομειώσεις σε σχέση με το πρώτο πείραμα. Επιπρόσθετα, στο πρώτο πείραμα ο αλγόριθμος χρησιμοποιεί κατά μέσο όρο 1,99 GB από τα 2GB (99,5%) ενώ στο δεύτερο χρησιμοποιεί κατά μέσο όρο μόνο τα 2,5GB από τα 4GB(62,5%). Κατά τις μετρήσεις του CPU performance στο πρώτο πείραμα φτάνει το 19,49% χρήσης της CPU με μέσο όρο 11,84%, ενώ στο δεύτερο πείραμα φτάνει το 50,52% με μέσο όρο 18,05%.

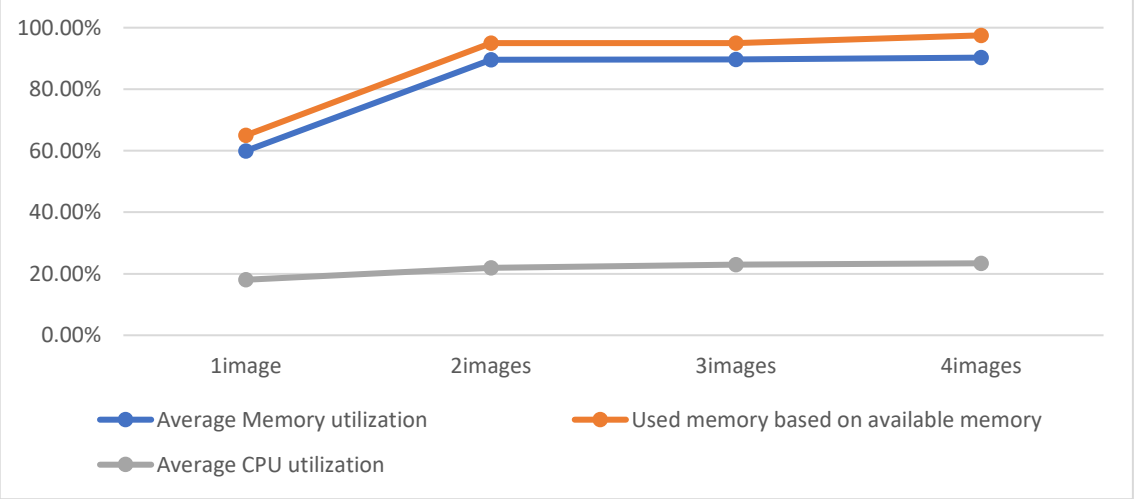
Η χαμηλότερη χρήση μνήμης στην υποδομή με 4 πυρήνες CPU και 4GB RAM υποδηλώνει ότι αποτελεί καλύτερη επιλογή πόρων για τον αλγόριθμο. Υποδεικνύει ότι ο αλγόριθμος μπορεί να λειτουργήσει πιο αποτελεσματικά και ενδεχομένως να έχει βελτιωμένη απόδοση λόγω της διαθεσιμότητας πρόσθετης μνήμης. Το δεύτερο πείραμα στην υποδομή επιδεικνύει πιο σταθερή χρήση μνήμης με ελάχιστες διακυμάνσεις σε σύγκριση με το πρώτο πείραμα. Αυτό υποδηλώνει καλύτερο έλεγχο και συνέπεια στη χρήση των πόρων. Ο αλγόριθμος έχει υψηλότερες απαιτήσεις μνήμης στο πρώτο πείραμα και χρησιμοποιεί μεγαλύτερο μέρος των διαθέσιμων πόρων. Επομένως, η υποδομή με 4 πυρήνες CPU και 4GB RAM παρέχει μεγαλύτερο περιθώριο και είναι καταλληλότερη για να διαχειριστεί τις απαιτήσεις μνήμης του αλγορίθμου. Το δεύτερο πείραμα παρουσιάζει υψηλότερη αξιοποίηση της CPU, γεγονός που υποδηλώνει καλύτερη αξιοποίηση των πόρων της CPU και ενδεχομένως ταχύτερους χρόνους εκτέλεσης. Ωστόσο, η διαφορά στη χρήση της CPU μεταξύ των δύο πειραμάτων δεν είναι σημαντική.

Drone



	1image	2images	3images	4images
Average Memory utilization	1.562740928	1.848515293	1.997228254	1.916088974
Average memory	16779802.9	19848281.82	21445075.09	20573848.7
Average CPU utilization	0.003902	0.003333	0.002951179	0.003792

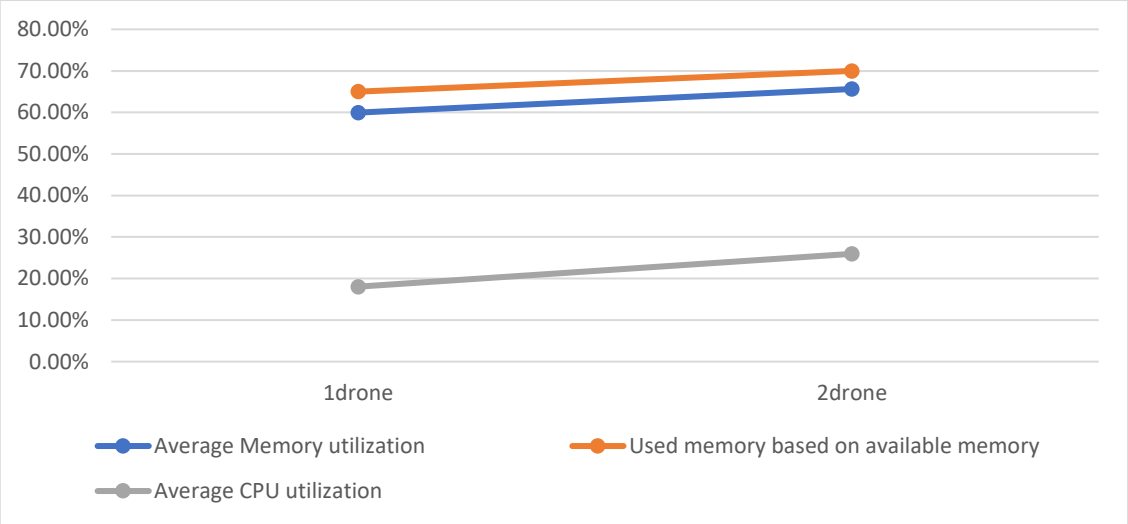
Οι πιο πάνω μετρήσεις αντιπροσωπεύουν το drone με σταθερό network delay από drone σε base station στα 5ms το οποίο στέλνει εικόνες προς το base station είτε 1,2,3,4 ανά request. Βάση των μετρήσεων παρατηρείτε, ότι καθώς αυξάνεται ο αριθμός των εικόνων, αυξάνεται και η μέση χρήση της μνήμης παραμένοντας όμως σε πολύ χαμηλά επίπεδα. Αυτό υποδεικνύει ότι απαιτούνται περισσότεροι πόροι μνήμης για τη διαχείριση των πρόσθετων εικόνων. Η μέση χρήση μνήμης σε bytes αυξάνεται επίσης καθώς αυξάνεται ο αριθμός των εικόνων φτάνοντας σχεδόν στα 2GB μνήμης RAM. Αυτό ευθυγραμμίζεται με την τάση που παρατηρείται στη μέση χρήση μνήμης, επιβεβαιώνοντας ότι απαιτείται περισσότερη μνήμη για την επεξεργασία μεγαλύτερου αριθμού εικόνων. Η μέση χρήση της KME παραμένει σχετικά σταθερή και πάρα πολύ μικρή σε διαφορετικές ποσότητες εικόνων.



	1image	2images	3images	4images
Average Memory utilization (%)	59.95%	89.60%	89.64%	90.26%
Average memory (GB)	2.6	3.8	3.8	3.9
Used memory based on available memory	65%	95%	95%	97.5%
Max CPU utilization (%)	50.52%	58.11%	36.20%	36.43%
Average CPU utilization (%)	18.05%	21.93%	22.94%	23.40%

Οι πιο πάνω μετρήσεις, αντιπροσωπεύουν το base station infrastructure με σταθερό network delay από drone σε base station στα 5ms, χρήση του 2ου ML model αλγόριθμου, 4 CPU cores και 4 GB RAM το οποίο λαμβάνει εικόνες από το drone είτε 1,2,3,4 ανά request. Βάση των δεδομένων πιο πάνω παρατηρούμε ότι καθώς αυξάνεται ο αριθμός των εικόνων, αυξάνεται και το average memory utilization. Αυτό δείχνει ότι ο αλγόριθμος απαιτεί περισσότερους πόρους μνήμης για την επεξεργασία μεγαλύτερου

αριθμού εικόνων. Το average memory σε bytes αυξάνεται επίσης καθώς αυξάνεται ο αριθμός των εικόνων. Αυτό ευθυγραμμίζεται με την τάση που παρατηρείται στο average memory utilization, υποδεικνύοντας μεγαλύτερη ζήτηση πόρων μνήμης με την αύξηση του αριθμού των εικόνων. Το ποσοστό χρησιμοποιημένης μνήμης με βάση τη διαθέσιμη μνήμη αυξάνεται καθώς αυξάνεται ο αριθμός των εικόνων. Αυτό υποδηλώνει ότι ένα μεγαλύτερο μέρος της διαθέσιμης μνήμης χρησιμοποιείται για την αντιμετώπιση του αυξημένου φόρτου εργασίας επεξεργασίας εικόνων. Η μέγιστη χρήση της CPU παραμένει σχετικά σταθερή σε διαφορετικές ποσότητες εικόνων. Δεν υπάρχει σαφής συσχέτιση μεταξύ του αριθμού των εικόνων και της μέγιστης χρήσης της CPU, αν και μπορούμε να προσέξουμε ότι ξεπερνώντας τις 2 εικόνες ανά επεξεργασία το μέγιστο utilization της CPU παραμένει μικρό και σταθερό σε αντίθεης με την τα προηγούμενα πειράματα. Η μέση χρήση της CPU παρουσιάζει μια μικρή αύξηση καθώς αυξάνεται ο αριθμός των εικόνων. Αυτό υποδηλώνει ότι απαιτούνται περισσότεροι υπολογιστικοί πόροι για την επεξεργασία μεγαλύτερου αριθμού εικόνων, με αποτέλεσμα ελαφρώς υψηλότερη μέση χρήση CPU.



	1 drone	2 drones
Average Memory utilization	59.95%	65.64%
Average memory (GB)	2.6	2.8
Used memory based on available memory	65%	70%
Average CPU utilization	18.05%	25.95%

Αυτές οι μετρήσεις, αντιπροσωπεύουν το base station infrastructure με σταθερό network delay από drone σε base station στα 5ms, χρήση του 2ου ML model αλγόριθμου, 4 CPU

cores και 4 GB RAM το οποίο λαμβάνει εικόνες είτε από ένα drone είτε από 2 drones. Βάση των πιο πάνω δεδομένων, όταν αυξάνονται τα drones ανάλογα αυξάνονται και οι απαιτήσεις του base station στα resources γιατί οι εικόνες που πρέπει να επεξεργαστεί αναλογούν στον αριθμό drones επί των αριθμό των εικόνων που έχουν να στείλουν. Επειδή όμως δεν υπάρχει παραλληλία στο base station οι εικόνες των drones επεξεργάζονται με την σειρά, δηλαδή πρώτα όλες οι εικόνες του ενός drone και μετά οι εικόνες του επομένου. Με ένα μικρό network delay μεταξύ drones και base station, το base station δεν θα έχει μεγάλα χρονικά αδράνειας και οι πόροι θα χρησιμοποιούνται συνεχώς αυξάνοντας έτσι τα utilization των resources. Για αυτό παρατηρείτε και η πιο πάνω αύξηση στα 2 drones έναντι του ενός.

6.5 Πείραμα 5

Για να εκτιμηθούν οι οικονομικές επιπτώσεις της χρήσης του δικτύου για το έργο, διενεργήθηκε ανάλυση για τον προσδιορισμό του πραγματικού κόστους που συνδέεται με τη μετάδοση δεδομένων μέσω του δικτύου. Η αξιολόγηση αυτή έλαβε υπόψη παράγοντες όπως ο όγκος δεδομένων και η χρήση του δικτύου, ώστε να αποκτηθούν γνώσεις σχετικά με την οικονομική σκοπιμότητα και τον αντίκτυπο της ανάπτυξης του προγράμματος σε ένα πραγματικό σενάριο.

Η αξιολόγηση αυτή, παρέχει πολύτιμες πληροφορίες σχετικά με τις οικονομικές εκτιμήσεις και της οικονομικής βιωσιμότητας που αφορούν τη λειτουργία του προγράμματος. Η κατανόηση των οικονομικών επιπτώσεων επιτρέπει στους ενδιαφερόμενους να λαμβάνουν τεκμηριωμένες αποφάσεις σχετικά με τον προϋπολογισμό του προγράμματος και την κατανομή των πόρων.

1 εικόνα ανά request

drone to basestation	base station to cloud	cloud to basestation	Basestation to drone
6728155.5	31075	4567.5	3392632

2 εικόνες ανά request

drone to basestation	base station to cloud	cloud to basestation	basestation to drone
2259754.5	3162.5	3162.5	2257549.5

Κατά την ανάλυση των παρεχόμενων πινάκων, παρατηρούμε ότι ο πρώτος πίνακας έχει συνολικά 10.156.430 bits, ενώ ο δεύτερος πίνακας έχει συνολικά 4.523.629 bits.

Εκτιμώντας τους συνολικούς χρόνους εκτέλεσης των προγραμμάτων, κατά την αποστολή 1 εικόνας ανά request και 2 εικόνων ανά request, μπορούμε να υπολογίσουμε το average rate των bits ανά δευτερόλεπτο. Ο πρώτος πίνακας, παρουσιάζει average rate 105.995,8 bits/sec, ενώ ο δεύτερος πίνακας παρουσιάζει μέσο ρυθμό 50.521,42 bits/sec.
Επιπλέον, λαμβάνοντας υπόψη τη δομή τιμολόγησης, διαπιστώνουμε ότι η αποστολή 10.156.430 bits συνεπάγεται χρέωση 0,18€, ενώ η αποστολή 4.523.629 bits συνεπάγεται χρέωση 0,079€. Με βάση το μέσο ρυθμό ανά δευτερόλεπτο, μπορούμε να εκτιμήσουμε ότι το κόστος είναι 0,0019€/sec για το πρώτο πείραμα και 0,00089€/sec για το δεύτερο πείραμα.

*Να σημειωθεί ότι στην Κύπρο, το μέσο κόστος για 1GB δεδομένων είναι περίπου 18,88€.

Κεφάλαιο 7

Συμπεράσματα

7.1 Συμπεράσματα Πειραμάτων

Τα πειράματα που διεξήχθησαν παρέχουν πολύτιμες πληροφορίες σχετικά με την απόδοση, τη χρήση των πόρων και των αλγορίθμων στην υποδομή του σταθμού βάσης. Από την πιο πάνω ανάλυση προέκυψαν διάφορα βασικά ευρήματα:

Σύγκριση αλγορίθμων: Ο πρώτος αλγόριθμος υπερτερεί σταθερά έναντι του δεύτερου αλγορίθμου όσον αφορά τον χρόνο εκτέλεσης. Ανεξάρτητα από τη διαμόρφωση του υλικού, ο πρώτος αλγόριθμος επιδεικνύει σημαντικά μικρότερους χρόνους εκτέλεσης, καθιστώντας τον πιο αποδοτικό ως προς το χρόνο.
Επίδραση των requests: Ο συνολικός χρόνος εκτέλεσης του προγράμματος εξαρτάται από τον αριθμό των requests που αποστέλλει το drone στο σταθμό βάσης και όχι τόσο από τον αριθμό των εικόνων που αποστέλλονται. Η αύξηση του αριθμού των requests ανά εκτέλεση, οδηγεί σε μεγαλύτερο συνολικό χρόνο εκτέλεσης του drone και, κατά συνέπεια, σε αύξηση του συνολικού χρόνου εκτέλεσης του προγράμματος.

Υποδομή base station: Τα υπολογιστικά χαρακτηριστικά της υποδομής του σταθμού βάσης επηρεάζουν σημαντικά τον χρόνο εκτέλεσης των αλγορίθμων. Η διαθεσιμότητα επαρκών πόρων υλικού, όπως πυρήνες CPU και μνήμη RAM, διαδραματίζει καθοριστικό ρόλο στην επίτευξη βέλτιστης απόδοσης.

Καθυστέρηση δικτύου: Η συνολική καθυστέρηση δικτύου κατά την εκτέλεση του προγράμματος είναι ανάλογη του συνολικού χρόνου εκτέλεσης του προγράμματος. Η ελαχιστοποίηση του network delay (round-trip time) μπορεί να οδηγήσει σε συντομότερους χρόνους εκτέλεσης του προγράμματος.

Αποδοτικότητα πόρων: Ο πρώτος αλγόριθμος επιδεικνύει καλύτερες δυνατότητες διαχείρισης πόρων όσον αφορά τη χρήση της μνήμης και την απόδοση της CPU. Επιτυγχάνει χαμηλότερη χρήση μνήμης, ενώ επιτυγχάνει συγκρίσιμη ή υψηλότερη χρήση CPU, γεγονός που υποδηλώνει αποδοτικότερη κατανομή πόρων.

7.2 Τελικοί στόχοι

Ο πρωταρχικός στόχος αυτού του ερευνητικού έργου ήταν η διεξαγωγή μιας ολοκληρωμένης αξιολόγησης των επιδόσεων, με τη δημιουργία σημείων αναφοράς σε ένα σενάριο κινητών κόμβων. Η αξιολόγηση αυτή αποσκοπούσε στην παροχή αξιόπιστης εκτίμησης της εμπιστοσύνης και της αποδοτικότητας του έργου, εξασφαλίζοντας τη βιωσιμότητά του για μελλοντική χρήση. Επιπλέον, το έργο αποσκοπούσε στη συλλογή δεδομένων και τη δημιουργία πληροφοριών οδικής κυκλοφορίας σε πραγματικό χρόνο προς όφελος των οδηγών.

Το ερευνητικό έργο επικεντρώθηκε στην ανάπτυξη μιας προσαρμόσιμης εφαρμογής που χρησιμοποιούσε ένα drone, έναν σταθμό βάσης και μια υποδομή cloud για την αποτελεσματική καταμέτρηση οχημάτων. Η διαδικασία περιελάμβανε τη λήψη εικόνων με τη χρήση του drone και τη χρήση μοντέλων μηχανικής μάθησης στο σταθμό βάσης για την ακριβή καταμέτρηση οχημάτων σε κάθε εικόνα. Αξιοποιώντας τις δυνατότητες της τεχνολογίας 5G, η εφαρμογή διευκόλυνε την απρόσκοπτη επικοινωνία μεταξύ των διαφόρων στοιχείων. Αν και το στοιχείο του cloud παρέμεινε θεωρητικό, οραματίστηκε να παρέχει έγκαιρες ειδοποιήσεις στους οδηγούς με βάση τις συγκεκριμένες διαδρομές τους.

Για να μετρηθεί η αποτελεσματικότητα της εφαρμογής, πραγματοποιήθηκαν εκτεταμένα πειράματα σε προσομοιωμένο περιβάλλον. Τα πειράματα αυτά περιλάμβαναν τη

μέτρηση διαφόρων μετρικών για την ενδελεχή αξιολόγηση της αποδοτικότητας και της αποτελεσματικότητας της εφαρμογής. Οι πρωταρχικοί στόχοι ήταν να αξιολογηθεί η απόδοση της εφαρμογής και να καθοριστούν σημεία αναφοράς απόδοσης για μελλοντική αναφορά.

Επιπλέον, το έργο αποσκοπούσε στην αξιοποίηση των συλλεχθέντων δεδομένων για τη δημιουργία πληροφοριών οδικής κυκλοφορίας σε πραγματικό χρόνο. Με τον τρόπο αυτό, η εφαρμογή σκόπευε να παρέχει πολύτιμες πληροφορίες στους οδηγούς, δίνοντάς τους τη δυνατότητα να λαμβάνουν ενημερωμένες πληροφορίες σχετικά με τις οδικές συνθήκες και τη ροή της κυκλοφορίας. Αυτός ο πρόσθετος στόχος αποσκοπούσε στην ενίσχυση της ασφάλειας των οδηγών και στη βελτιστοποίηση της ταξιδιωτικής εμπειρίας.

7.3 Γενικά Συμπεράσματα

Σκοπός της διεξαγωγής αυτών των πειραμάτων είναι να παράσχει πολύτιμες πληροφορίες σχετικά με τις παραμέτρους που πρέπει να ληφθούν υπόψη κατά την εφαρμογή του προγράμματος στην πραγματικότητα και να εντοπιστούν ευκαιρίες για τη βελτιστοποίηση της αποτελεσματικότητάς του. Με την ανάλυση διαφόρων παραγόντων, τα πειράματα ρίχνουν φως στο πώς το πρόγραμμα μπορεί να γίνει πιο βέλτιστο και αποδοτικό. Τα ευρήματα αυτά χρησιμεύουν ως πολύτιμος οδηγός για τη λήψη τεκμηριωμένων αποφάσεων σχετικά με την αρχιτεκτονική, τις παραμέτρους του συστήματος και την κατανομή των πόρων, οδηγώντας τελικά σε μια βελτιωμένη εφαρμογή του προγράμματος από άποψη επιδόσεων και αποτελεσματικότητας.

Συμπερασματικά, η βελτιστοποίηση της υποδομής του base station και των παραμέτρων του συστήματος είναι ζωτικής σημασίας για την επίτευξη συντομότερων χρόνων εκτέλεσης του προγράμματος και βελτιωμένων επιδόσεων. Οι βασικές εκτιμήσεις περιλαμβάνουν τη μείωση του network delay (round-trip time) μεταξύ των microservices, τη διασφάλιση επαρκών πόρων υλικού στην υποδομή του base station (όπως 4 πυρήνες CPU και 4 GB RAM) και τη συγκέντρωση και αποστολή πολλαπλών εικόνων μαζί ανά request. Αυτή η προσέγγιση συμβάλλει στη μεγιστοποίηση της αποδοτικότητας και της χρήσης των πόρων.

Επιπλέον, η σημασία της υποδομής του base station παίζει σημαντικό αντίκτυπο στην συνολική καθυστέρηση και απόδοση του προγράμματος. Η κατανομή των κατάλληλων υπολογιστικών πόρων στο σταθμό βάσης και όχι στο σύννεφο ή στο drone είναι κρίσιμη

για την επίτευξη βέλτιστης απόδοσης. Επιπρόσθετα, η αποστολή πολλαπλών εικόνων ανά request μπορεί να οδηγήσει σε μείωση των γενικών εξόδων και βελτιωμένη απόδοση, με αποτέλεσμα χαμηλότερη χρήση πόρων και καλύτερη συνολική απόδοση του προγράμματος.

Συνολικά, τα αποτελέσματα παρέχουν πληροφορίες σχετικά με τη βέλτιστη διαμόρφωση του υλικού και τις παραμέτρους του συστήματος για τους αλγορίθμους στην υποδομή του σταθμού βάσης, συμβάλλοντας στην καλύτερη διαχείριση των πόρων, στη μείωση των χρόνων εκτέλεσης και στη βελτίωση της αποδοτικότητας. Τα ευρήματα αυτά μπορούν να καθοδηγήσουν τις διαδικασίες λήψης αποφάσεων για τη βελτιστοποίηση της αρχιτεκτονικής και των παραμέτρων του συστήματος για την ανάπτυξη σε πραγματικό κόσμο, βελτιώνοντας τελικά την απόδοση και την οικονομική σκοπιμότητα του προγράμματος.

7.4 Μελλοντικές Έρευνες

Αρχικά, θα μπορούσε να εξεταστεί η ενσωμάτωση άλλων αλγορίθμων μοντέλων μηχανικής μάθησης (ML) στην υποδομή του σταθμού βάσης. Με την εισαγωγή μιας ποικιλίας μοντέλων ML, μπορεί να αξιολογηθεί η αποτελεσματικότητα τους για διαφορετικές εργασίες. Επιπλέον, η επέκταση της υπηρεσίας νέφους και η αξιοποίηση των αποτελεσμάτων της για διάφορους σκοπούς αποτελεί έναν ενδιαφέροντα τομέα για περαιτέρω διερεύνηση. Η επέκταση των δυνατοτήτων της υπηρεσίας νέφους μπορεί να ανοίξει ευκαιρίες για προηγμένη ανάλυση δεδομένων, λήψη αποφάσεων και παραγωγή πολύτιμων πληροφοριών για διάφορους ενδιαφερόμενους. Επιπρόσθετα, για να διασφαλιστεί η ευρωστία και η επεκτασιμότητα του προγράμματος, η αντιμετώπιση των υφιστάμενων προβλημάτων του κώδικα και η προσαρμογή του για την υποστήριξη πολλαπλών drones και σταθμών βάσης είναι ζωτικής σημασίας. Αυτό θα περιλαμβάνει τη βελτίωση της ποιότητας του κώδικα, τη βελτιστοποίηση των μηχανισμών συγχρονισμού και τη διασφάλιση του απρόσκοπτου συντονισμού μεταξύ των συσκευών. Επιπλέον, η ενσωμάτωση τεχνικών παράλληλης επεξεργασίας στην υποδομή του σταθμού βάσης μπορεί να βελτιώσει σημαντικά τις επιδόσεις και την απόδοση. Με την αξιοποίηση του παραλληλισμού, πολλαπλές εργασίες μπορούν να υποβάλλονται σε ταυτόχρονη επεξεργασία, βελτιώνοντας τη συνολική απόδοση και μειώνοντας τους

χρόνους εκτέλεσης. Μπορεί επίσης να διερευνηθεί η δυνατότητα ταυτόχρονης παραλαβής εικόνων, επιτρέποντας την ταχύτερη παραλαβή και επεξεργασία δεδομένων. Αυτές οι μελλοντικές ερευνητικές κατευθύνσεις υπόσχονται βελτίωση των επιδόσεων, της επεκτασιμότητας και της αποδοτικότητας του προγράμματος. Με τη διερεύνηση αυτών των τομέων μπορούν να αποκτηθούν πολύτιμες γνώσεις, συμβάλλοντας στην ανάπτυξη ενός πιο προηγμένου και βελτιστοποιημένου συστήματος που θα ανταποκρίνεται στις απαιτήσεις των εφαρμογών του πραγματικού κόσμου.

Βιβλιογραφία

[1] A.Edenbrandt, Hong, C. H., & Varghese, B. (2019). Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms. ACM Computing Surveys (CSUR), 52(5), 1-37.

[2] Coutinho, A., Greve, F., Prazeres, C., & Cardoso, J. (2018, May). Fogbed: A rapid-prototyping emulation environment for fog computing. In 2018 IEEE International Conference on Communications (ICC) (pp. 1-7). IEEE.

[3] Hasenburg, J., Grambow, M., Grünewald, E., Huk, S., & Bermbach, D. (2019, June). Mockfog: Emulating fog computing infrastructure in the cloud. In 2019 IEEE International Conference on Fog Computing (ICFC) (pp. 144-152). IEEE.

[4] Hong, C. H., & Varghese, B. (2019). Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms. ACM Computing Surveys (CSUR), 52(5), 1-37.

[5] Lantz, B., Heller, B., & McKeown, N. (2010, October). A network in a laptop: rapid prototyping for software-defined networks. In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (pp. 1-6).

[6] Mayer, R., Graser, L., Gupta, H., Saurez, E., & Ramachandran, U. (2017, October). Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures. In 2017 IEEE Fog World Congress (FWC) (pp. 1-6). IEEE.

[7] Naqvi, N. Z., Vansteenkiste-Muylle, T., & Berbers, Y. (2015, July). Benchmarking leading-edge mobile devices for data-intensive distributed mobile

cloud applications. In 2015 IEEE Symposium on Computers and Communication (ISCC) (pp. 50-57). IEEE.

[8] Symeonides, M., Georgiou, Z., Trihinas, D., Pallis, G., & Dikaiakos, M. D. (2020, November). Fogify: A fog computing emulation framework. In 2020 IEEE/ACM Symposium on Edge Computing (SEC) (pp. 42-54). IEEE.

[9] Symeonides, M., Trihinas, D., Pallis, G., Dikaiakos, M. D., Psomas, C., & Krikidis, I. (2022, May). 5g-slicer: An emulator for mobile iot applications deployed over 5g network slices. In 2022 IEEE/ACM Seventh International Conference on Internet-of-Things Design and Implementation (IoTDI) (pp. 115-127). IEEE.

[10] Varghese, B., Subba, L. T., Thai, L., & Barker, A. (2016, May). DocLite: A docker-based lightweight cloud benchmarking tool. In 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid) (pp. 213-222). IEEE.

[11] Zeng, Y., Chao, M., & Stoleru, R. (2019, June). Emuedge: A hybrid emulator for reproducible and realistic edge computing experiments. In 2019 IEEE International Conference on Fog Computing (ICFC) (pp. 153-164). IEEE.

Παράρτημα Α

Πιο κάτω παρατίθεται το αρχείο το οποίο συμπεριλαμβάνει τα αποτελέσματα των πειραμάτων του κεφαλαίου 6.

https://drive.google.com/file/d/1WXrApVdjYl5LZYyJdQEfUlxRMnFEOysq/view?usp=sharing

Πιο κάτω παρατίθεται το αρχείο του κώδικα της εφαρμογής του σεναρίου χρήσης.

https://github.com/DimitrisGrigoras/ADE.git