



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ  
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ  
ΥΠΟΛΟΓΙΣΤΩΝ  
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

## Stackelberg Game-Based Resource Management of Edge Computing Systems

Αντώνιος Α. Καρτέρης  
Α.Μ. : 03112076

Επιβλέπων : Δημήτριος Ι. Σούντρης  
Καθηγητής ΕΜΠ

Αθήνα  
Ιούλιος 2021





ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ  
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ  
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

# Stackelberg Game-Based Resource Management of Edge Computing Systems

Αντώνιος Α. Καρτέρης  
Α.Μ. : 03112076

Επιβλέπων : Δημήτριος Ι. Σούντρης  
Καθηγητής ΕΜΠ

Τριμελής Επιτροπή Εξέτασης

(Υπογραφή)

(Υπογραφή)

(Υπογραφή)

.....  
Δημήτριος Σούντρης  
Καθηγητής  
ΕΜΠ

.....  
Παναγιώτης Τσανάκας  
Καθηγητής  
ΕΜΠ

.....  
Σωτήριος Ξύδης  
Επίκουρος Καθηγητής  
ΧΠΑ

Ημερομηνία Εξέτασης:  
16 Ιουλίου 2021

Copyright © - All rights reserved Καρτέρης Αντώνιος, 2021.  
Με επιφύλαξη παντός δικαιώματος.

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

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

(Υπογραφή)

.....  
**Αντώνιος Καρτέρης**  
©2021 - All rights reserved.

# Περίληψη

Οι αδιάκοπες τεχνολογικές εξελίξεις και η εντυπωσιακή άνθηση του Διαδικτύου των Πραγμάτων (Internet of Things) έχουν εκτοξεύσει τον αριθμό των συσκευών που συνδέονται και επικοινωνούν μεταξύ τους στο διαδίκτυο. Παράλληλα, η ραγδαία υιοθέτηση εφαρμογών τεχνητής νοημοσύνης σε συσκευές IoT έχει προκαλέσει κατακόρυφη αύξηση των δεδομένων που χρήζουν επεξεργασίας, ενώ η παραδοσιακή λύση της μεταφοράς και επεξεργασίας δεδομένων στο cloud έχει αποδειχθεί ελλιπής. Η επικρατούσα λύση του edge computing, παρά τα πλεονεκτήματα που προσφέρει, δημιουργεί νέες προκλήσεις, με σημαντικότερη ίσως την διαχείριση των περιορισμένων υπολογιστικών πόρων που διαθέτουν οι συσκευές IoT.

Στην παρούσα διπλωματική εργασία θα ασχοληθούμε με ένα σύστημα edge computing αποτελούμενο από ένα σύνολο από συσκευές IoT και κόμβους edge. Οι συσκευές καλούνται να φέρουν εις πέρας απαιτητικές εργασίες μέσα σε αυστηρές προθεσμίες, ενώ οι κόμβοι συμβάλλουν προσφέροντας τους υπολογιστικούς τους πόρους στο σύστημα. Προκύπτει, επομένως, η ανάγκη αποδοτικής διαχείρισης των πόρων του συστήματος, προκειμένου να επιτυγχάνονται οι λειτουργικές απαιτήσεις των εργασιών και να αξιοποιούνται αποδοτικά οι περιορισμένοι πόροι των συσκευών.

Για τον σκοπό αυτό, σχεδιάζουμε τον αλγόριθμο SGRM, έναν αποκεντροποιημένο αλγόριθμο διαχείρισης πόρων σε συστήματα edge computing βασισμένο σε στοιχεία της θεωρίας παιγνίων. Πιο συγκεκριμένα, ο αλγόριθμός μας μοντελοποιεί το πρόβλημα της κατανομής εργασιών μιας συσκευής edge ως ένα παίγνιο Στάκελμπέργκ μεταξύ της συσκευής και των κόμβων edge και διεξάγει μια κλειστή δημοπρασία δεύτερης τιμής (γνωστή και ως δημοπρασία Vickrey) για την επιλογή του βέλτιστου κόμβου εκφόρτωσης.

Μετά από μια σύντομη εισαγωγή στα πρότυπα IoT, edge, και στο θεωρητικό υπόβαθρο της θεωρίας παιγνίων, παρουσιάζουμε τον αλγόριθμο SGRM, περιγράφουμε λεπτομερώς την λειτουργία του και αξιολογούμε τις επιδόσεις του, πραγματοποιώντας μια εκτενή πειραματική μελέτη και συγκρίνοντάς τον με κατάλληλα επιλεγμένους αλγορίθμους αναφοράς.

**Λέξεις Κλειδιά** — Cloud, Edge, Fog, IoT, Διαχείριση Πόρων, Κατανομή Εργασιών, Θεωρία Παιγνίων, Παίγνιο Στάκελμπέργκ, Δημοπρασία Βίκρεϊ, SGRM



# Abstract

The incessant technological advancements and the unprecedented surge in popularity of the Internet of Things have catapulted the number of devices connected and communicating through the Internet, while the rapid adoption of artificial intelligence applications in IoT devices has led the amount of data that requires processing to skyrocket. The edge computing paradigm, albeit effective in addressing the challenges that the cloud computing solution fails to meet, presents its own challenges, with the resource management and task allocation challenge being of prime importance.

In this bachelor thesis, we will describe an edge computing architecture consisting of low-powered edge devices that are tasked with carrying out demanding tasks within strict deadlines, and mid-powered edge nodes that offer their limited computational resources to the edge devices. Thus, the need to effectively manage the limited resources of these devices and optimally allocate the tasks among them to achieve the required objectives arises.

To that end, we put forward the SGRM algorithm, a distributed resource management algorithm for edge computing systems based in theorems of game and auction theory. More precisely, our algorithm models the task allocation problem of an edge device as a Stackelberg game between the device and the nodes, and conducts a sealed-bid second-price (or Vickrey) auction to select the optimal offloading target node.

After a brief introduction to the basics of the edge and IoT paradigms and the theoretical background of our algorithm, we present the SGRM algorithm, describe its operation in detail and evaluate its performance through an extensive comparative study.

**Keywords** — Cloud, Edge, Fog, Computing, IoT, Resource Management, Task Offloading, Game Theory, Stackelberg Game, Vickrey Auction, SGRM





# Ευχαριστίες

Αρχικά, ευχαριστώ θερμά τον καθηγητή μου, κ. Δημήτριο Σούντρη, για την εμπιστοσύνη που μου έδειξε από την πρώτη μας επικοινωνία μέχρι και σήμερα. Επίσης, ένα μεγάλο ευχαριστώ στους υποψήφιους διδάκτορες κ.κ. Μασούρο και Κατσαραγάκη για την εξαιρετική συνεργασία μας και την ακούραστη καθοδήγηση που μου προσέφεραν. Τέλος, ευχαριστώ απο καρδιάς τους γονείς μου, την αδερφή μου, τους φίλους μου και την κοπέλα μου για την απεριόριστη στήριξη και ώθηση που μου έδωσαν καθ' όλη την ακαδημαϊκή μου πορεία.



# Contents

Περίληψη	5
Abstract	7
Ευχαριστίες	9
Contents	11
Εκτεταμένη Περίληψη	17
<b>1 Introduction</b>	<b>35</b>
1.1 Internet of Things and Cloud Computing . . . . .	35
1.2 Edge Computing . . . . .	36
1.3 Resource Management in Edge Computing . . . . .	37
1.4 Approaches to Resource Management . . . . .	38
1.5 Thesis Overview . . . . .	39
<b>2 Related Work</b>	<b>41</b>
2.1 Game & Market Theory Based Approaches . . . . .	41
2.2 Stackelberg Game Approaches . . . . .	41
<b>3 Game Theory and Models</b>	<b>43</b>
3.1 Game Theory Fundamentals . . . . .	43
3.2 Game Theoretic Models . . . . .	43
3.2.1 Non-Cooperative Games . . . . .	44
3.2.2 Stackelberg Games . . . . .	44
3.2.3 Sealed-Bid Auctions . . . . .	45
3.2.4 Other Approaches . . . . .	45
<b>4 SGRM: Stackelberg Game-Based Resource Management</b>	<b>47</b>
4.1 System & Problem Formulation . . . . .	47
4.1.1 System Formulation . . . . .	47
4.1.2 Problem Formulation . . . . .	49
4.1.3 Incentive Definition . . . . .	49

4.2	Theoretical Fundamentals . . . . .	50
4.2.1	Stackelberg Equilibrium . . . . .	51
4.3	Modules . . . . .	52
4.3.1	Edge Device Module . . . . .	53
4.3.2	Edge Gateway Module . . . . .	53
4.3.3	Task Evaluation Module . . . . .	54
<b>5</b>	<b>Technical Implementation</b>	<b>59</b>
5.1	Applications . . . . .	59
5.1.1	Deepspeech . . . . .	59
5.1.2	Facenet . . . . .	60
5.1.3	Lanenet . . . . .	61
5.1.4	Retain . . . . .	62
5.2	Application Workbench . . . . .	63
5.3	Algorithmic Baselines . . . . .	64
5.3.1	Offload None & Offload All . . . . .	64
5.3.2	Oracle . . . . .	65
<b>6</b>	<b>Experimental Evaluation</b>	<b>67</b>
6.1	Experimental Setup . . . . .	67
6.1.1	Raspberry Pi 4 Model B . . . . .	67
6.1.2	Nvidia Jetson TX1 . . . . .	68
6.1.3	Nvidia Jetson Nano . . . . .	69
6.1.4	64-bit Virtual Machine . . . . .	70
6.2	Comparative Study . . . . .	70
6.2.1	Burst Workloads . . . . .	70
6.2.2	Sustained Workloads . . . . .	71
6.3	Implementation Specifics . . . . .	73
6.3.1	Deadline Selection . . . . .	73
6.3.2	Overhead . . . . .	74
6.3.3	Scalability . . . . .	75
6.3.4	Prediction Mechanism Evaluation . . . . .	76
6.4	Aggregate Evaluation . . . . .	77
<b>7</b>	<b>Conclusions</b>	<b>79</b>
7.1	Thesis Summary . . . . .	79
7.2	Future Work . . . . .	79

# List of Figures

1	Αρχιτεκτονική Συστήματος Edge Computing . . . . .	17
2	Οργάνωση του SGRM . . . . .	22
3	Βελτίωση Εκτίμησης Χρόνου Εκτέλεσης . . . . .	24
4	Πλατφόρμα edgebench . . . . .	26
5	‘Ριπές’ Εργασιών: Σύστημα . . . . .	28
6	‘Ριπές’ Εργασιών: Αποτελέσματα . . . . .	28
7	Παρατεταμένες Ακολουθίες Εργασιών: Σύστημα . . . . .	29
8	Παρατεταμένες Ακολουθίες Εργασιών: Αποτελέσματα . . . . .	29
9	Επίδραση Προθεσμιών στα Αποτελέσματα των Πειραμάτων . . . . .	30
10	Χρονικό Κόστος Αλγορίθμου . . . . .	31
11	Υπολογιστικό Κόστος Αλγορίθμου . . . . .	31
12	Επίδραση σύνδεσης περισσότερων κόμβων . . . . .	31
13	Επίδραση ταυτόχρονης άφιξης περισσότερων εργασιών . . . . .	32
14	Ποσοστό Λάθους Μηχανισμού Πρόβλεψης . . . . .	32
1.1	Expected Adoption Growth of IoT Devices [1] . . . . .	35
1.2	Edge Computing Architecture [2] . . . . .	37
1.3	Resource Management Cycle . . . . .	38
3.1	Taxonomy of Games . . . . .	44
3.2	Stackelberg Game Cycle . . . . .	45
4.1	Target system architecture . . . . .	50
4.2	SGRM Organization . . . . .	52
4.3	Time Prediction Improvement . . . . .	55
5.1	Deepspeech function . . . . .	59
5.2	Deepspeech Resource Utilization . . . . .	60
5.3	Facenet function . . . . .	60
5.4	Facenet Resource Utilization . . . . .	61
5.5	Lanenet function . . . . .	61
5.6	Lanenet Resource Utilization . . . . .	62
5.7	Retain function . . . . .	62
5.8	Retain Resource Utilization . . . . .	63
5.9	Edgebench organization . . . . .	63

6.1	Raspberry Pi 4 Model B . . . . .	68
6.2	Jetson TX1 Developer Kit . . . . .	69
6.3	Jetson Nano Developer Kit . . . . .	69
6.4	Burst Scenario System . . . . .	71
6.5	Burst Scenario Results . . . . .	71
6.6	Sustained Scenario System . . . . .	72
6.7	Sustained Scenario Results . . . . .	73
6.8	Deadline Choice Effect Demonstration . . . . .	74
6.9	Time Overhead of SGRM . . . . .	74
6.10	Resource Overhead of SGRM . . . . .	75
6.11	Effect of Generating Tasks Simultaneously . . . . .	76
6.12	Effect of Additional Gateway Participants to Decision Times . . . . .	76
6.13	Prediction Module Error Rate . . . . .	77

# List of Tables

1	Μεταβλητές Συστήματος . . . . .	19
2	Επεξήγηση Χρονικών Μεταβλητών . . . . .	24
3	‘Ριπές’ Εργασιών: Παράμετροι . . . . .	27
4	Παρατεταμένες Ακολουθίες Εργασιών: Παράμετροι . . . . .	29
4.1	Key System Model Parameters . . . . .	48
4.2	Time Variable Definitions . . . . .	57
6.1	Burst Scenarios: Parameters . . . . .	70
6.2	Sustained Scenarios: Parameters . . . . .	72
6.3	Task Deadlines . . . . .	73

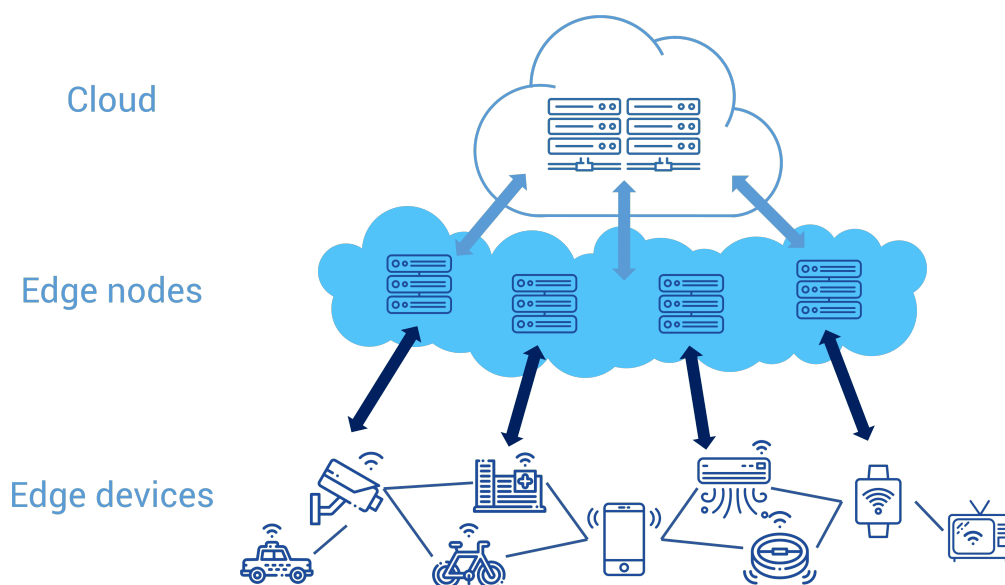




# Εκτεταμένη Περίληψη

## Εισαγωγή

Οι αδιάκοπες τεχνολογικές εξελίξεις, η πρόσφατη εισαγωγή του δικτυακού προτύπου 5G και η εκτεταμένη διάδοση εφαρμογών τεχνητής νοημοσύνης έχουν εκτοξεύσει την δημοτικότητα του Διαδικτύου των Πραγμάτων στα ύψη. Η ραγδαία υιοθέτηση και αξιοποίηση του προτύπου IoT έχει προκαλέσει δραματική αύξηση του αριθμού των συσκευών που συνδέονται στο διαδίκτυο, ενώ το πρότυπο cloud computing έχει αποδειχθεί ελλιπές για τις αυξημένες ανάγκες επεξεργασίας δεδομένων που παρουσιάζουν τα σύγχρονα υπολογιστικά συστήματα. Ταυτόχρονα, πολυάριθμες καινοτόμες εφαρμογές παρουσιάζουν νέες απαιτήσεις, όπως επεξεργασία δεδομένων σε πραγματικό χρόνο ή αυξημένη ιδιωτικότητα και ασφάλεια δεδομένων, οι οποίες δεν πληρούνται επαρκώς από τις υπάρχουσες λύσεις. Αυτές οι ανάγκες οδήγησαν στην δημιουργία του προτύπου Edge Computing, όπου μεγάλο μέρος της επεξεργασίας δεδομένων που απαιτείται δεν πραγματοποιείται σε απομακρυσμένους cloud servers, αλλά εκτελείται σε κόμβους edge (ή edge nodes/-gateways) που βρίσκονται κοντά στις συσκευές IoT. Μια τέτοια αρχιτεκτονική edge παρουσιάζεται στην εικόνα 1.



Εικόνα 1: Αρχιτεκτονική Συστήματος Edge Computing

Η λύση του edge computing, αν και αποτελεσματική στην ικανοποίηση των

παραπάνω απαιτήσεων, εισάγει πολυάριθμες νέες ερευνητικές προκλήσεις. Στην παρούσα διπλωματική θα ασχοληθούμε με το πρόβλημα διαχείρισης πόρων και κατανομής εργασιών, διατυπωμένο ως εξής: Σε αντίθεση με τους πανίσχυρους cloud servers, οι συσκευές IoT διαθέτουν εξαιρετικά περιορισμένους υπολογιστικούς πόρους, τους οποίους καλούνται να διαχειριστούν αποτελεσματικά, προκειμένου να φέρουν επαρκώς εις πέρας τις εργασίες που τους έχουν ανατεθεί. Παράλληλα, πολλές από αυτές τις εργασίες έχουν αυστηρές χρονικές προθεσμίες ολοκλήρωσης, ενώ τα τοπικά δίκτυα που συνδέουν τις συσκευές μεταξύ τους διαθέτουν περιορισμένο εύρος ζώνης. Για αυτόν τον σκοπό σχεδιάζουμε τον αλγόριθμο SGRM, έναν κατανεμημένο και κλιμακώσιμο αλγόριθμο διαχείρισης πόρων και κατανομής εργασιών σε συστήματα edge computing. Στις ακόλουθες ενότητες θα παρουσιάσουμε το θεωρητικό υπόβαθρο του αλγορίθμου, θα περιγράψουμε τον σχεδιασμό του, θα αναλύσουμε την τεχνική του υλοποίηση και θα αξιολογήσουμε πειραματικά τις επιδόσεις του μέσω εκτενούς συγκριτικής μελέτης.

## Διατύπωση του Προβλήματος

Στην παρούσα διπλωματική εργασία εξετάζουμε ένα σύστημα edge, αποτελούμενο από συσκευές δύο ειδών:

- **Συσκευές Edge:** Συσκευές IoT περιορισμένης υπολογιστικής ισχύος, επιφορτισμένες με την εκτέλεση εργασιών που ξεπερνούν τις ικανότητές τους.
- **Κόμβοι Edge (Edge Nodes/Gateways):** Συσκευές υψηλότερης υπολογιστικής ισχύος που προτίθενται να αναλάβουν και να εκτελέσουν εργασίες των συσκευών edge.

Η αρχιτεκτονική του συστήματος που θα μελετήσουμε αποτελείται από ένα σύνολο  $X$  συσκευών edge,  $Y$  κόμβων και  $Z$  εργασιών, οι οποίες παράγονται στις συσκευές edge και μπορούν είτε να εκτελεστούν επί τόπου, είτε να εκφορτωθούν στους κόμβους του δικτύου. Τόσο οι συσκευές όσο και οι κόμβοι που συμμετέχουν στο σύστημα διαθέτουν περιορισμένους υπολογιστικούς πόρους, ενώ το δίκτυο που συνδέει τις συσκευές διαθέτει περιορισμένο εύρος ζώνης για ανταλλαγή μηνυμάτων και δεδομένων.

Κάθε συσκευή edge που συμμετέχει στο σύστημα ορίζεται μονοσήμαντα από ένα αναγνωριστικό  $x \in \{1, \dots, X\}$ , και περιγράφεται από την πλειάδα  $D_x = \{C_x, M_x, N_x, t_x\}$ . Οι μεταβλητές  $C_x$ ,  $M_x$  και  $N_x$  υποδεικνύουν την διαθέσιμη επεξεργαστική ισχύ, χωρητικότητα μνήμης και εύρος ζώνης αντίστοιχα, ενώ η μεταβλητή  $t_x$  είναι ίση με τον αριθμό των εργασιών που εκτελούνται στην συσκευή. Παρόμοια, κάθε edge gateway που συμμετέχει στο σύστημα ορίζεται από ένα αναγνωριστικό  $y \in \{1, \dots, Y\}$  και μια πλειάδα  $G_y = \{C_y, M_y, N_y, t_y\}$ .

Συμβολισμός	Περιγραφή	Τιμές
$x$	Αναγνωριστικό Συσκευής	$1, \dots, X$
$y$	Αναγνωριστικό Κόμβου	$1, \dots, Y$
$C_x$	Διαθέσιμη Επεξεργαστική Ισχύς	$0 - 100\%$
$M_x$	Διαθέσιμη Χωρητικότητα Μνήμης	$0 - 100\%$
$N_x$	Διαθέσιμο Εύρος Ζώνης	$0 - 100\%$
$t_x$	Τρέχοντες Εργασίες	$\mathbb{N}$
$z$	Αναγνωριστικό Εργασίας	$1, \dots, Z$
$w_z$	Είδος Εργασίας	$\mathbb{N}$
$\Omega_z$	Προθεσμία Εργασίας	$\mathbb{R}_+^*$
$r_z$	Αμοιβή εργασίας	$\mathbb{R}_+^*$
$c_z^i$	Μέγιστη επεξεργαστική ισχύς	$0 - 100\%$
$m_z^i$	Μέγιστη χρήση μνήμης	$0 - 100\%$
$n_z^i$	Μέγιστη χρήση εύρους ζώνης	$0 - 100\%$
$x_z$	Συσκευή Παραγωγής	$1, \dots, X$
$y_z$	Συσκευή Εκτέλεσης	$0, \dots, Y$

Πίνακας 1: Μεταβλητές Συστήματος

Αντίστοιχα, οι εργασίες που παράγονται στο σύστημα ορίζονται από μονοσήμαντα αναγνωριστικά της μορφής  $z \in \{1, \dots, Z\}$ . Κάθε εργασία περιγράφεται από την πλειάδα  $T_z = \{w_z, \Omega_z, r_z, c_z^i, m_z^i, n_z^i, x_z, y_z\}$  ως εξής: Οι μεταβλητές  $\Omega_z$  και  $r_z$  ορίζουν την προθεσμία της εργασίας και την προσφερόμενη αμοιβή, αντίστοιχα. Οι μεταβλητές  $c_z^i$ ,  $m_z^i$  και  $n_z^i$  ορίζουν την μέγιστη απαιτούμενη επεξεργαστική ισχύ, χωρητικότητα μνήμης και εύρος ζώνης για την εκτέλεση της εργασίας στην συσκευή  $i$ , ενώ οι μεταβλητές  $x_z$  και  $y_z$  υποδεικνύουν την συσκευή στην οποία παράχθηκε η εργασία και την συσκευή στην οποία εκτελέστηκε. Σε περίπτωση τοπικής εκτέλεσης της εργασίας, θέτουμε  $y_z = 0$ . Τέλος, η μεταβλητή  $w_z$  κατηγοριοποιεί την εργασία για τις ανάγκες του μηχανισμού εκτίμησης εργασίας που περιγράφουμε σε επόμενη ενότητα.

Οι προαναφερθείσες μεταβλητές παρουσιάζονται εν συντομία στον πίνακα 1.

Έχοντας ορίσει το σύστημα, ορίζουμε τον βασικό στόχο του συστήματος:

$$\text{ελαχιστοποίησε } \sum_{z=1}^Z \delta_z \quad (1)$$

$$\text{όπου } \delta_z = \begin{cases} 1 & \text{αν η εργασία } z \text{ είναι ληξιπρόθεσμη} \\ 0 & \text{αλλιώς} \end{cases}$$

Στην συνέχεια, ορίζουμε τους περιορισμούς που διέπουν το σύστημα:

$$\text{Περιορισμοί Συσκευών : } \forall x, z \text{ τ.ω. } (x_z, y_z) = (x, 0) : \sum_z c_z^x < C_x \quad (2)$$

$$\forall x, z \text{ τ.ω. } (x_z, y_z) = (x, 0) : \sum_z m_z^x < M_x \quad (3)$$

$$\forall x, z \text{ τ.ω. } (x_z, y_z) = (x, 0) : \sum_z n_z^x < N_x \quad (4)$$

$$\text{Περιορισμοί Κόμβων : } \forall y, z \text{ τ.ω. } y_z = y : \sum_z c_z^y < C_y \quad (5)$$

$$\forall y, z \text{ τ.ω. } y_z = y : \sum_z m_z^y < M_y \quad (6)$$

$$\forall y, z \text{ τ.ω. } y_z = y : \sum_z n_z^y < N_y \quad (7)$$

## SGRM: Stackelberg Game-Based Resource Management

Προκειμένου να αντιμετωπίσουμε επιτυχώς το πρόβλημα διαχείρισης πόρων που περιγράψαμε παραπάνω, αναπτύσσουμε και υλοποιούμε τον αλγόριθμο SGRM, έναν κατανεμημένο αλγόριθμο κατανομής εργασιών βασισμένο σε θεμελιώδη θεωρήματα και μοντέλα της θεωρίας παιγνίων.

### Θεωρητική Θεμελίωση του SGRM

Η κεντρική ιδέα του αλγορίθμου SGRM είναι η μοντελοποίηση της διαδικασίας κατανομής μιας εργασίας σε ένα σύστημα edge computing ως ένα παίγνιο Στάκελμπεργκ.

Ως παίγνιο Στάκελμπεργκ ορίζεται ένα μη-συνεργατικό παιχνίδι, στο οποίο οι παίκτες διακρίνονται σε ‘ηγέτες’ (leaders) και ‘ακόλουθους’ (followers) [3]. Με βάση αυτή την διάκριση, το παιχνίδι παίζεται σε γύρους ως εξής:

1. Οι ηγέτες αποφασίζουν την στρατηγική τους,
2. Οι ηγέτες ανακοινώνουν την στρατηγική τους,
3. Οι ακόλουθοι ενημερώνονται για την στρατηγική των ηγετών και αποφασίζουν την στρατηγική τους,
4. Οι ακόλουθοι ανακοινώνουν την στρατηγική τους.

Στην περίπτωση του αλγορίθμου SGRM, οι συσκευές edge δρουν ως ηγέτες, αξιολογώντας μια εργασία και αποφασίζοντας την εκφόρτωσή της ή την πραγματοποίησή της τοπικά. Στην συνέχεια, οι κόμβοι edge, δρώντας ως ακόλουθοι στο

παίγνιο Στάκελμπεργκ, αξιολογούν με την σειρά τους την εργασία και ανακοινώνουν τις εκτιμήσεις τους στην συσκευή. Τέλος, η συσκευή διεξάγει μια κλειστή δημοπρασία δεύτερης τιμής (γνωστή και ως δημοπρασία Βίκρεϊ), προκειμένου να διακρίνει τον παίκτη που αξιολογεί μέγιστα την εν λόγω εργασία. Η εργασία εκφορτώνεται στην συσκευή ή κόμβο με την μέγιστη αξιολόγηση, ο οποίος καταθέτει ποσό αξίας ίσης με την δεύτερη μεγαλύτερη προσφορά της δημοπρασίας (ή εισπράττει, σε περίπτωση αρνητικών προσφορών). Μόλις ολοκληρωθεί η εκτέλεση της εργασίας, ο παίκτης ανταμοίβεται με ποσό ίσο με την αρχική αμοιβή  $r_z$  της εργασίας.

Σε αυτό το σημείο, είναι σημαντικό να ορίσουμε την έννοια της ανταμοιβής ή κινήτρου στην περίπτωση του αλγόριθμου μας. Το κίνητρο αυτό μπορεί να παίρνει την μορφή χρηματικής αμοιβής ή φήμης (reputation) [4], και η χρήση του στον αλγόριθμό μας καθιστά δυνατή την εφαρμογή του σε περιβάλλοντα όπου οι συμμετέχουσες συσκευές δεν συμμερίζονται τον κεντρικό στόχο του συστήματος που αναγράφουμε στην εξίσωση 1, αλλά δρουν με βάση το προσωπικό τους συμφέρον.

## Μηχανισμοί του SGRM

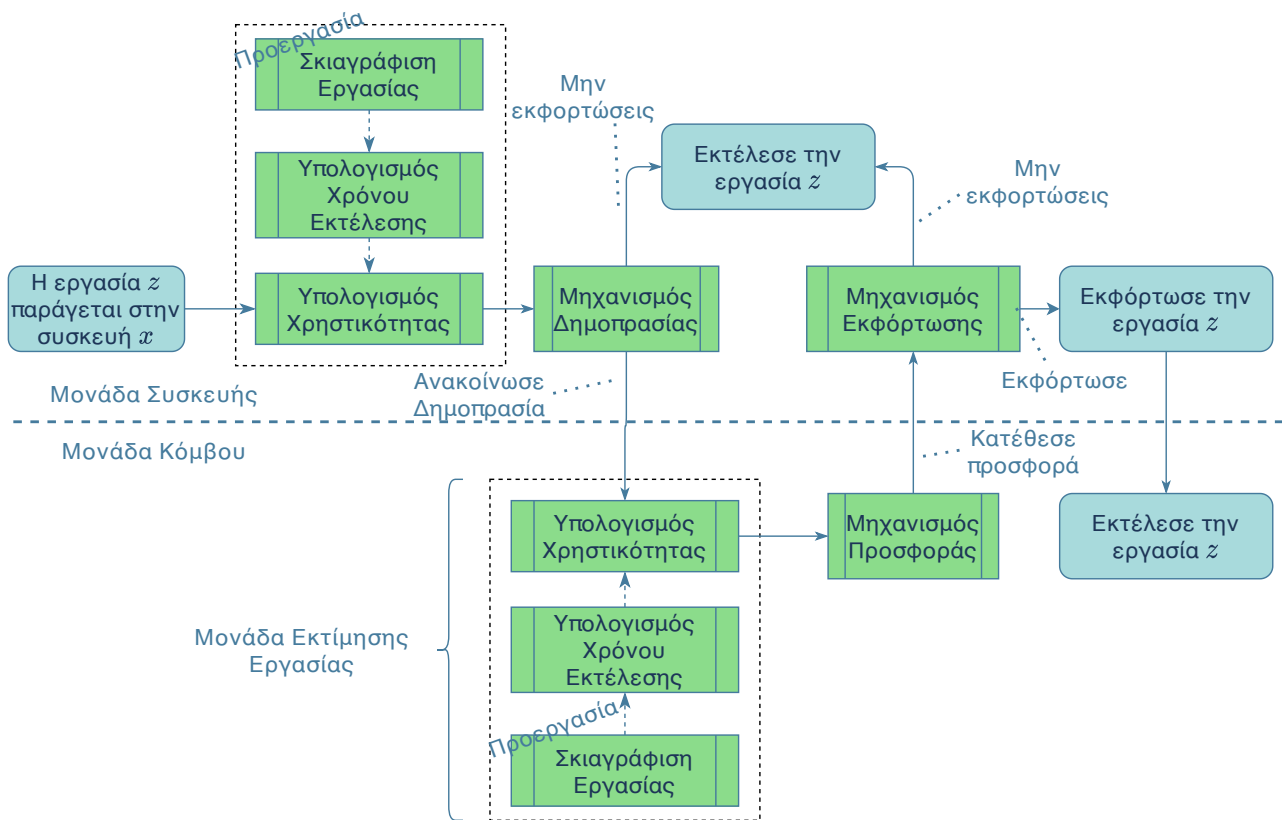
Για την υλοποίηση του SGRM κατασκευάζουμε ένα κατανεμημένο, δυναμικό σύστημα, αποτελούμενο από τις ακόλουθες μονάδες:

- **Μονάδα Συσκευής:** Περιλαμβάνει τους μηχανισμούς δημοπρασίας και εκφόρτωσης εργασιών.
- **Μονάδα Κόμβου:** Περιλαμβάνει τον μηχανισμό κατάθεσης προσφοράς.
- **Μονάδα Εκτίμησης Εργασίας:** Αποτελείται από έναν μηχανισμό σκιαγράφησης εργασιών (task profiling), έναν μηχανισμό εκτίμησης χρόνου εκτέλεσης εργασίας και έναν μηχανισμό υπολογισμού χρηστικότητας εργασίας (utility function).

Στο σχήμα 2 παρουσιάζεται η διασύνδεση και οργάνωση των μονάδων, ενώ παρακάτω περιγράφουμε αναλυτικά την λειτουργία τους.

### Μονάδα Συσκευής

Δεδομένης μιας νέας εργασίας, η μονάδα συσκευής (ή edge device module) αξιολογεί την εργασία χρησιμοποιώντας την μονάδα εκτίμησης εργασίας που περιγράφουμε παρακάτω. Εφόσον αυτή η εκτίμηση είναι εντός κάποιων προκαθορισμένων ορίων, ο μηχανισμός δημοπρασίας ανακοινώνει την πρόθεση εκφόρτωσης της εργασίας στους διασυνδεδεμένους κόμβους, και λαμβάνει τις προσφορές τους. Μόλις λάβει όλες τις προσφορές, ο μηχανισμός εκφόρτωσης τις συγκρίνει και εκφορτώνει την εργασία στον νικητή της δημοπρασίας.



Εικόνα 2: Οργάνωση του SGRM

## Μονάδα Κόμβου

Η μονάδα κόμβου (ή edge gateway module) ενεργοποιείται όταν ο κόμβος συνδεθεί σε δίκτυο edge που εκτελεί τον αλγόριθμο SGRM, και αρχικά ανακοινώνει την σύνδεση του κόμβου και την πρόθεσή του να αναλάβει την εκτέλεση εργασιών. Όταν λάβει μια πρόθεση εκφόρτωσης από κάποια συσκευή, ο μηχανισμός προσφοράς αξιολογεί την εργασία και καταθέτει την αξιολόγηση ως προσφορά στον πλειστηριασμό που διεξάγει η συσκευή. Αν ο κόμβος αποδειχθεί νικητής, αποδέχεται την εργασία και την εκτελεί τοπικά.

## Μονάδα Εκτίμησης Εργασίας

Όπως αναφέραμε παραπάνω, η δυνατότητα αξιολόγησης εργασιών είναι απαραίτητη για την σωστή λειτουργία του αλγορίθμου. Για τον σκοπό αυτό, σχεδιάζουμε μια μονάδα εκτίμησης εργασίας, αποτελούμενη από τους ακόλουθους μηχανισμούς:

### Μηχανισμός Σκιαγράφησης Εργασιών (Task Profiling)

Προκειμένου μια συσκευή να μπορέσει να συμμετάσχει σε ένα edge σύστημα που εκτελεί τον αλγόριθμο SGRM, απαιτείται πρώτα να περάσει από μια διαδικασία 'σκιαγράφησης', κατά την οποία θα ποσοτικοποιηθούν οι δυνατότητες της συσκευής και οι επιδόσεις της στην εκτέλεση των εφαρμογών που εκτελούνται στο σύστημα.

Η εν λόγω διαδικασία πραγματοποιείται από τον μηχανισμό σκιαγράφησης ως εξής:

1. **Εκτίμηση Τεμαχίων (Chunk Gauging):** Αρχικά, κάθε υποψήφια εφαρμογή εκτελείται απομονωμένα στην συσκευή και μετράται ο συνολικός χρόνος εκτέλεσής της. Ως ‘τεμάχιο’ (ή chunk) ορίζεται το τμήμα της εργασίας που εκτελείται σε 1 δευτερόλεπτο - για παράδειγμα, μια εφαρμογή που απαιτεί 12 δευτερόλεπτα για να εκτελεστεί στην συσκευή αποτελείται από 12 τεμάχια:

$$chunks_{w,i} = wcet_{w,i}(1) \quad (8)$$

2. **Υπολογισμός Χρόνου Εκτέλεσης Χειρότερης Περίπτωσης (WCET Estimation):** Ο μηχανισμός εκτελεί συνδυασμούς των υποψηφίων εφαρμογών, και για κάθε συνδυασμό εργασιών εκτιμά τον χρόνο εκτέλεσης σύμφωνα με την εξίσωση:

$$wcet_{w,i}(t_i) = chunks_{w,i} \times t_i \quad (9)$$

Αφού εκτελεστεί ο εν λόγω συνδυασμός και μετρηθεί ο πραγματικός χρόνος εκτέλεσης της εφαρμογής, υπολογίζεται και καταχωρείται το ποσοστό λάθους μεταξύ της εκτίμησης και της πραγματικής μέτρησης.

3. **Αντιστάθμιση Λάθους (Error Compensation):** Σύμφωνα με τα ποσοστά λάθους του προηγούμενου βήματος, ο μηχανισμός υπολογίζει ένα λογαριθμικό trendline των ποσοστών συναρτήσει του πλήθους εργασιών που εκτελούνται:

$$compf_{w,i}(t_i) = \alpha_{w,i} \times \ln t_i + \beta_{w,i} \quad (10)$$

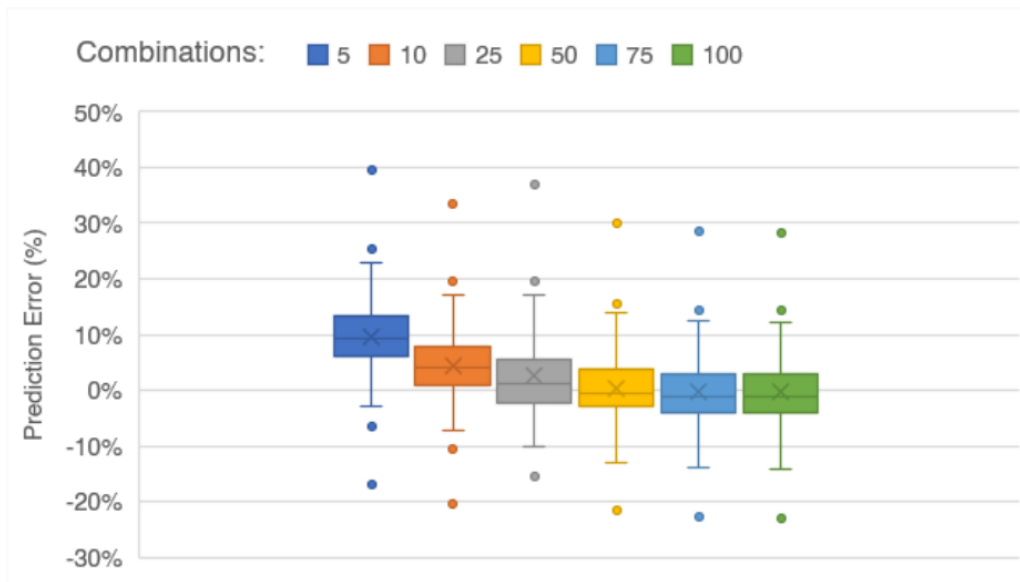
Ός αποτέλεσμα της παραπάνω προεργασίας, προκύπτει μια πλειάδα της μορφής  $(chunks_{w,i}, \alpha_{w,i}, \beta_{w,i})$  για κάθε συνδυασμό συσκευής  $i$  και τύπου εργασίας  $w$ , την οποία χρησιμοποιούν οι επόμενοι μηχανισμοί της μονάδας εκτίμησης που περιγράφονται παρακάτω. Όσον αφορά το απαιτούμενο πλήθος συνδυασμών εργασιών που πρέπει να εξετασθούν από τον μηχανισμό, στο σχήμα 3 φαίνεται πως αρκούν περίπου 25 συνδυασμοί εργασιών για την παραγωγή ικανοποιητικών προβλέψεων.

### Μηχανισμός Εκτίμησης Χρόνου Εκτέλεσης Εργασίας

Έχοντας ολοκληρώσει την σκιαγράφηση των εργασιών, ο μηχανισμός εκτίμησης, οπλισμένος με τις πλειάδες  $(chunks_{w,i}, \alpha_{w,i}, \beta_{w,i})$ , υπολογίζει τον χρόνο εκτέλεσης μιας εργασίας τύπου  $w$  σύμφωνα με τον τύπο:

$$wcet_{w,i}(t_i) = chunks_{w,i} \times t_i - compf_{w,i}(t_i) \quad (11)$$

$$\text{όπου } compf_{w,i}(t_i) = \alpha_{w,i} \times \ln t_i + \beta_{w,i} \quad (12)$$



Εικόνα 3: Βελτίωση Εκτίμησης Χρόνου Εκτέλεσης

### Μηχανισμός Υπολογισμού Χρηστικότητας Εργασίας (Utility Function)

Προκειμένου να ποσοτικοποιήσουμε την αξία μιας υποψήφιας εργασίας, σχεδιάζουμε μια συνάρτηση χρηστικότητας (utility function) ως εξής:

Πρώτα, υπολογίζεται μια συνάρτηση κύρωσης (penalty function), η οποία ελατώνει την χρηστικότητα της εργασίας ανάλογα με τον χρόνο εκτέλεσής της:

$$l_{z,x} = \frac{\Omega_z}{\Omega_z - wcet_{w,z,x}(t_x) - wt_{z,x}} \quad (13)$$

Μια ελαφρώς παραλλαγμένη συνάρτηση κύρωσης υπολογίζεται στους κόμβους edge:

$$l_{z,x,y} = \frac{\Omega_z}{\Omega_z - wcet_{w,z,y}(t_y) - tt_{z,x,y} - wt_{z,y}} \quad (14)$$

Ο πίνακας 2 προσφέρει μια σύντομη περιγραφή των μεταβλητών των παραπάνω συναρτήσεων.

Συμβολισμός	Περιγραφή
$\Omega_z$	Προθεσμία εργασίας $z$
$wcet_{w,i}$	Χρόνος εκτέλεσης χειρότερης περίπτωσης εργασίας τύπου $w$ στην συσκευή $i$
$tt_{z,x,y}$	Χρόνος μεταφοράς της εργασίας $z$ από την συσκευή $x$ στον κόμβο $y$
$wt_{z,i}$	Χρόνος αναμονής της εργασίας $z$ στην συσκευή $i$

Πίνακας 2: Επεξήγηση Χρονικών Μεταβλητών

Στην συνέχεια, υπολογίζεται μια δεύτερη συνάρτηση κύρωσης, η οποία ρυθμίζει την αξία της εργασίας ανάλογα με τον αριθμό των εργασιών που εκτελούνται ήδη στην συσκευή:



$$p_x = \begin{cases} c^{t_x} & \text{αν } l_{z,x} < 0 \\ \frac{1}{c^{t_x}} & \text{αν } l_{z,x} > 0 \end{cases} \quad (15)$$

Αντίστοιχα, για τον κόμβο η δεύτερη συνάρτηση κύρωσης υπολογίζεται σύμφωνα με τον τύπο:

$$p_{z,y} = \begin{cases} c^{t_y} & \text{αν } l_{z,x,y} < 0 \\ \frac{1}{c^{t_y}} & \text{αν } l_{z,x,y} > 0 \end{cases} \quad (16)$$

Ο αριθμός  $c$  αποτελεί μια σταθερά μεγαλύτερη της μονάδας.

Τέλος, υπολογίζουμε την συνάρτηση χρηστικότητας στην συσκευή edge:

$$u_{z,x} = \frac{r_z}{l_{z,x} \times p_x} \quad (17)$$

και στον κόμβο edge:

$$u_{z,x,y} = \frac{r_z}{l_{z,x,y} \times p_y} \quad (18)$$

## Πειραματική Υλοποίηση

Προκειμένου να αξιολογήσουμε την απόδοση του SGRM, επιστρατεύουμε μια σειρά από εφαρμογές τεχνητής νοημοσύνης, τις οποίες οργανώνουμε χρησιμοποιώντας την πλατφόρμα edgbench [5], ένα σύστημα εκτέλεσης, παρακολούθησης και συντονισμού εργασιών, κατασκευασμένο ειδικά για τις ανάγκες της εργασίας.

### Εφαρμογές

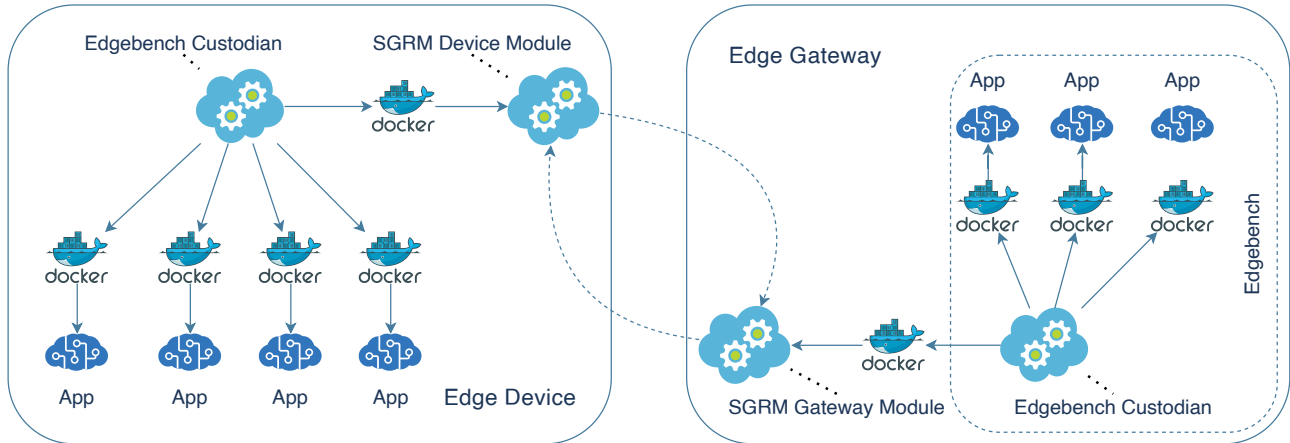
Οι εφαρμογές που επιστρατεύσαμε για τις ανάγκες της αξιολόγησης του αλγορίθμου είναι εν συντομία:

- Deepspeech [6, 7], μια εφαρμογή απομαγνητοφώνησης σε πραγματικό χρόνο,
- Facenet [8, 9], μια εφαρμογή αναγνώρισης προσώπων,
- Lanenet [10, 11], μια εφαρμογή real-time αναγνώρισης λωρίδων κυκλοφορίας,
- Retain [12, 13], μια εφαρμογή ανάλυσης ιατρικών φακέλων και πρόβλεψης καρδιακής ανεπάρκειας. [14].

Οι εφαρμογές Deepspeech, Facenet και Lanenet κάνουν χρήση της πλατφόρμας Tensorflow [15] της Google, ενώ η εφαρμογή Retain χρησιμοποιεί την βιβλιοθήκη μηχανικής μάθησης Theano [14].

## Edgebench

Προκειμένου να επιτύχουμε τον συντονισμό και την παρακολούθηση των παραπάνω εφαρμογών, σχεδιάζουμε την πλατφόρμα Edgebench, ένα application workbench γραμμένο σε γλώσσες προγραμματισμού Python και Shell. Παρουσιάζουμε την οργάνωση της πλατφόρμας στην εικόνα 4.



Εικόνα 4: Πλατφόρμα edgebench

Κάνοντας εκτενή χρήση **docker containers**, η πλατφόρμα **edgebench** προσφέρεται για γρήγορη και εύκολη εγκατάσταση σε πληθώρα συσκευών και αρχιτεκτονικών. Βασικό συστατικό της πλατφόρμας είναι η υπηρεσία *custodian*, η οποία εξασφαλίζει την ορθή εκτέλεση, παρακολούθηση και συντονισμό των εφαρμογών, αλλά και καταγραφή και αξιολόγηση των αποτελεσμάτων. Η επικοινωνία της υπηρεσίας με τις εφαρμογές που εκτελούνται στο εκάστοτε σύστημα επιτυγχάνεται με την χρήση πινάκων **pandas** ενώ τα αλγοριθμικά **modules** επικοινωνούν μεταξύ τους χρησιμοποιώντας το δημοφιλές πρωτόκολλο επικοινωνίας **MQTT**.

## Αλγόριθμοι Αναφοράς

Προκειμένου να αξιολογήσουμε τα αποτελέσματα της εκτέλεσης του αλγορίθμου **SGRM**, κρίνεται αναγκαίος ο σχεδιασμός ορισμένων αλγορίθμων αναφοράς, τους οποίους θα εκτελέσουμε και συγκρίνουμε με τα αποτελέσματα του αλγορίθμου μας. Αυτοί οι αλγόριθμοι είναι οι εξής:

- **Offload None:** Αφελής αλγόριθμος, εξαναγκάζει όλες τις συσκευές να εκτελέσουν τις εργασίες τους τοπικά.
- **Offload All:** Αφελής αλγόριθμος, εξαναγκάζει όλες τις συσκευές να εκφορτώσουν τις εργασίες τους στον πλησιέστερο διαθέσιμο κόμβο.
- **Oracle:** Αλγόριθμος εξαντλητικής αναζήτησης, δοκιμάζει όλα τα ενδεχόμενα εκτέλεσης και εκφόρτωσης εργασιών και επιστρέφει το βέλτιστο δυνατό αποτέλεσμα.

Εδώ σημειώνουμε πως ο αλγόριθμος Oracle, όντας αλγόριθμος εξαντλητικής αναζήτησης, παρουσιάζει εκθετική χρονική πολυπλοκότητα συναρτήσει του αριθμού εργασιών και κόμβων εκφόρτωσης. Γι' αυτόν τον λόγο, θα χρησιμοποιήσουμε τον εν λόγω αλγόριθμο μόνο στο σενάρια 'ριπών' εργασιών που παρουσιάζουμε παρακάτω.

## Συγκριτική Μελέτη

Έχοντας υλοποιήσει τον αλγόριθμο SGRM και το πειραματικό workbench που περιγράψαμε παραπάνω, υλοποιούμε ένα πραγματικό δίκτυο edge, αποτελούμενο από μια ευρεία γκάμα ετερογενών συσκευών, με ποικιλία δυνατοτήτων και περιορισμών. Ενδεικτικά, το δίκτυό μας περιλαμβάνει τρεις συσκευές Raspberry Pi 4 Model B αρχιτεκτονικής ARMv7 32-bit, δύο συσκευές Jetson αρχιτεκτονικής ARMv8 64-bit (Jetson TX1 και Jetson Nano), καθώς και δύο ισχυρά εικονικά συστήματα (Virtual Machines) αρχιτεκτονικής AMD64 64-bit. Περισσότερες πληροφορίες για τις συσκευές μπορούν να βρεθούν στο κεφάλαιο 6.1 της εργασίας.

### 'Ριπές' Εργασιών

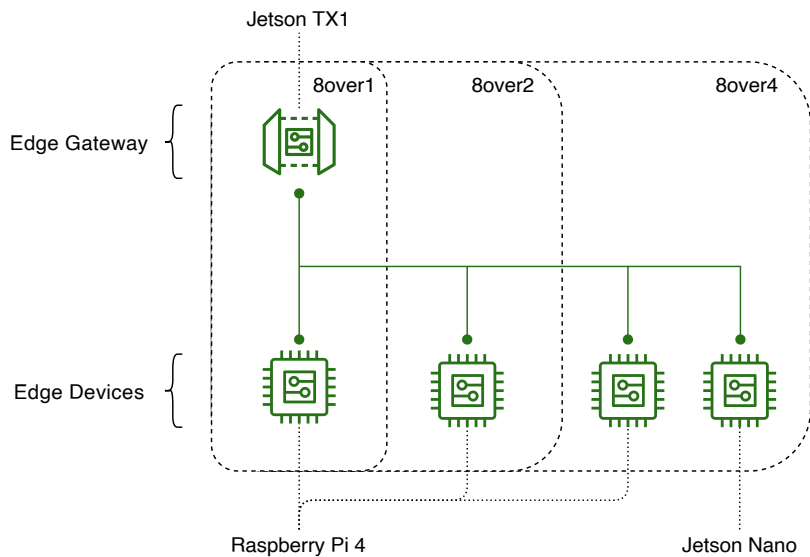
Αρχικά, υποβάλλουμε τους αλγόριθμους σε μια σειρά από σύντομες ακολουθίες εργασιών, τις οποίες ονομάζουμε συμβατικά 'ριπές'. Πιο συγκεκριμένα, καταστρώνουμε τα σενάρια εργασιών **8over1**, **8over2** και **8over4**, όπου ο πρώτος αριθμός υποδηλώνει τον αριθμό των εργασιών του σεναρίου και ο δεύτερος αριθμός τον αριθμό των συσκευών edge που συμμετέχουν στο σενάριο. Οι παράμετροι των σεναρίων παρουσιάζονται πιο αναλυτικά στον πίνακα 3.

Σενάριο	Εργασίες	Συσκευές	Κόμβοι	Χρονική Κατανομή
8over1	8	1	1	 Κανονική
8over2	8	2	1	
8over4	8	4	1	

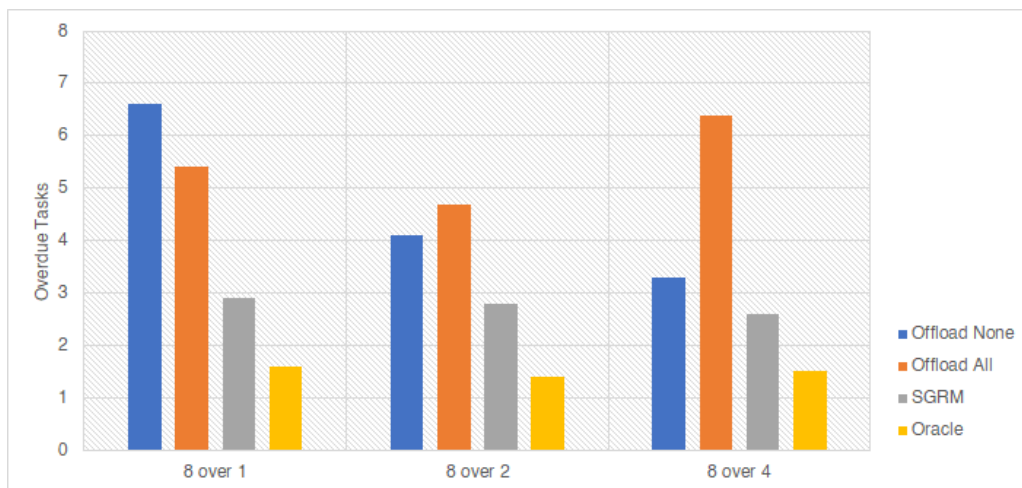
Πίνακας 3: 'Ριπές' Εργασιών: Παράμετροι

Και στα τρία σενάρια συμμετέχει μόνο ένας κόμβος edge, μιας και η συμμετοχή περισσότερων αποδεικνύεται περιττή για έναν τόσο μικρό όγκο εργασιών. Στην εικόνα 5 παρουσιάζουμε την αρχιτεκτονική του δικτύου edge που κατασκευάσαμε για τα προκείμενα σενάρια, ενώ στην εικόνα 6 παρουσιάζουμε τα αποτελέσματα της εκτέλεσης των αλγορίθμων.

Σημειώνουμε πως ο μικρός αριθμός εργασιών και η συμμετοχή ενός μόνο κόμβου στο σύστημα έχει ως αποτέλεσμα την ύπαρξη  $2^8$  διαφορετικών ενδεχομένων εκφόρτωσης, γεγονός που καθιστά δυνατή την εκτέλεση του αλγορίθμου εξαντλητικής αναζήτησης Oracle σε εύλογο χρονικό διάστημα.



Εικόνα 5: 'Ριπές' Εργασιών: Σύστημα



Εικόνα 6: 'Ριπές' Εργασιών: Αποτελέσματα

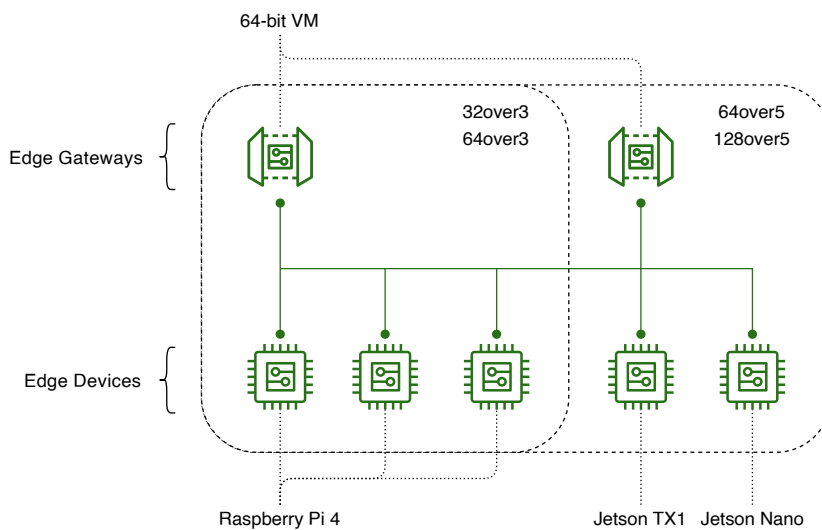
Όπως φαίνεται και στο διάγραμμα, ο αλγόριθμος SGRM καταφέρνει να διαχειριστεί και να εκφορτώσει σύντομες ριπές απαιτητικών εργασιών αποτελεσματικά, επισιχιάζοντας τους 'αφελείς' αλγορίθμους Offload None & All και προσωμοιώνοντας σε μεγάλο βαθμό την συμπεριφορά του αλγορίθμου Oracle.

## Παρατεταμένες Ακολουθίες Εργασιών

Στην συνέχεια, σχεδιάζουμε μια σειρά από παρατεταμένες ακολουθίες εργασιών, προκειμένου να μελετήσουμε την συμπεριφορά του αλγορίθμου SGRM όταν καλείται να διαχειριστεί μεγάλο πλήθος εργασιών σε ευρύ χρονικό διάστημα. Παρουσιάζουμε τα εν λόγω σενάρια στον πίνακα 4, την αρχιτεκτονική του δικτύου στο σχήμα 7 και τα αποτελέσματα της αξιολόγησης στο σχήμα 8, ενώ σημειώνουμε πως εδώ αποφύγαμε την χρήση του αλγορίθμου Oracle, μιας και το αυξημένο πλήθος εργασιών καθιστά την χρήση του απαγορευτική.

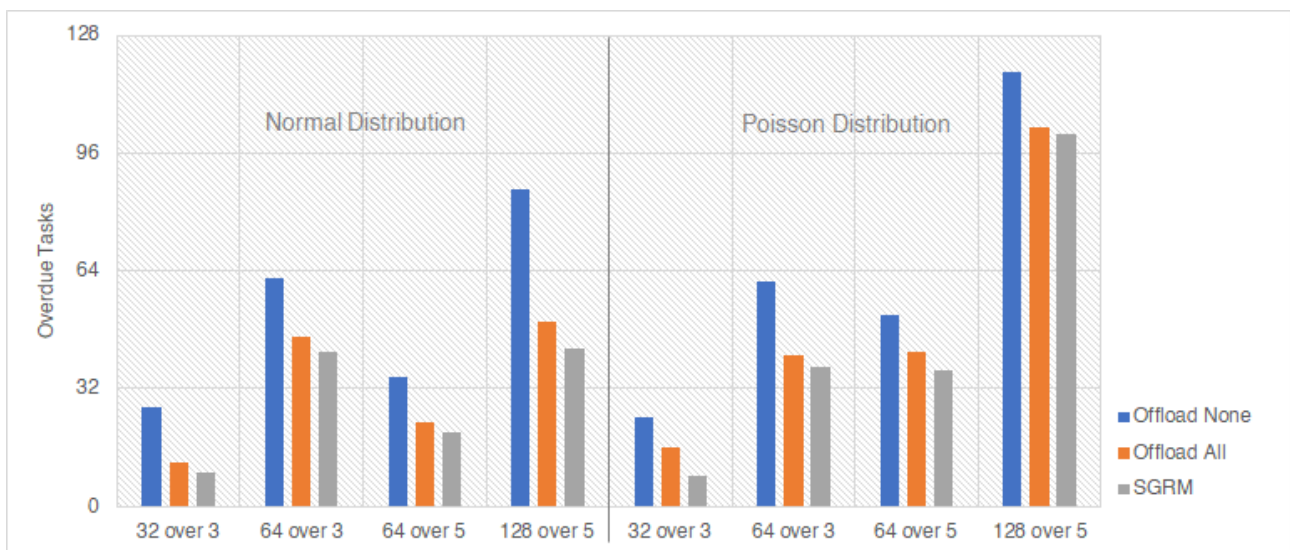
Σενάριο	Εφαρμογές	Συσκευές	Κόμβοι	Χρονική Κατανομή
32over3 <sub>g</sub>	32	3	1	 Κανονική
64over3 <sub>g</sub>	64	3	1	
64over5 <sub>g</sub>	64	5	2	
128over5 <sub>g</sub>	128	5	2	
32over3 <sub>p</sub>	32	3	1	 Poisson
64over3 <sub>p</sub>	64	3	1	
64over5 <sub>p</sub>	64	5	2	
128over5 <sub>p</sub>	128	5	2	

Πίνακας 4: Παρατεταμένες Ακολουθίες Εργασιών: Παράμετροι



Εικόνα 7: Παρατεταμένες Ακολουθίες Εργασιών: Σύστημα

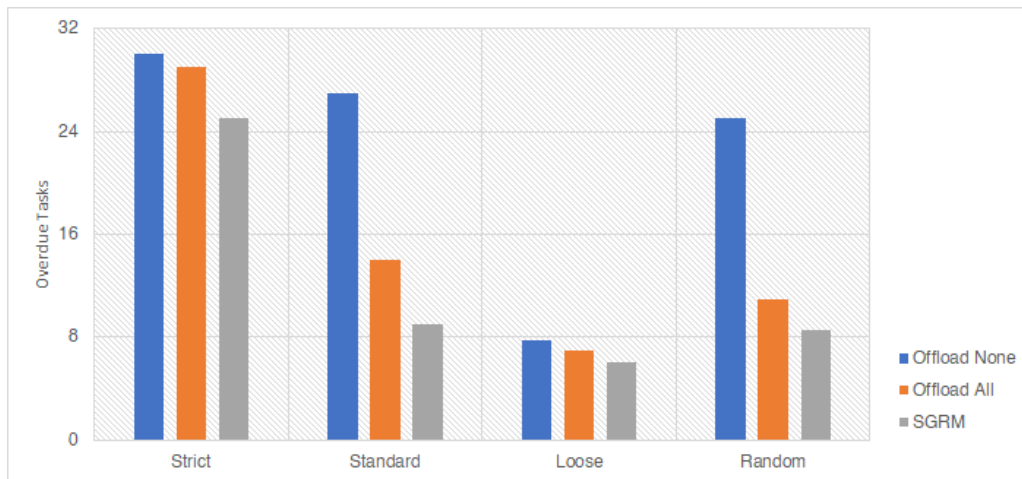
Όπως βλέπουμε στο διάγραμμα, ο αλγόριθμός καταφέρνει να διαχειριστεί επιτυχώς παρατεταμένες ακολουθίες εργασιών για μεγάλα χρονικά διαστήματα και να παράξει καλύτερα αποτελέσματα από ότι οι αφελείς συγκριτικοί αλγόριθμοι Offload None και Offload All.



Εικόνα 8: Παρατεταμένες Ακολουθίες Εργασιών: Αποτελέσματα

## Προθεσμίες Εργασιών

Στα σενάρια εργασιών των προηγούμενων ενοτήτων, οι προθεσμίες των εργασιών επιλέχτηκαν κατάλληλα, προκειμένου να εξασφαλισθεί η παραγωγή διαφοροποιημένων αποτελεσμάτων μεταξύ των αλγορίθμων που εκτελέστηκαν και να καταστεί δυνατή η ουσιαστική σύγκρισή τους. Στην εικόνα 9 φαίνεται η επίδραση της λανθασμένης επιλογής προθεσμιών στην σύγκριση των αλγορίθμων κατά την εκτέλεση του παρατεταμένου σεναρίου εργασιών 32over3, καθώς και η σημαντικότητα επιλογής ορθών προθεσμιών.



Εικόνα 9: Επίδραση Προθεσμιών στα Αποτελέσματα των Πειραμάτων

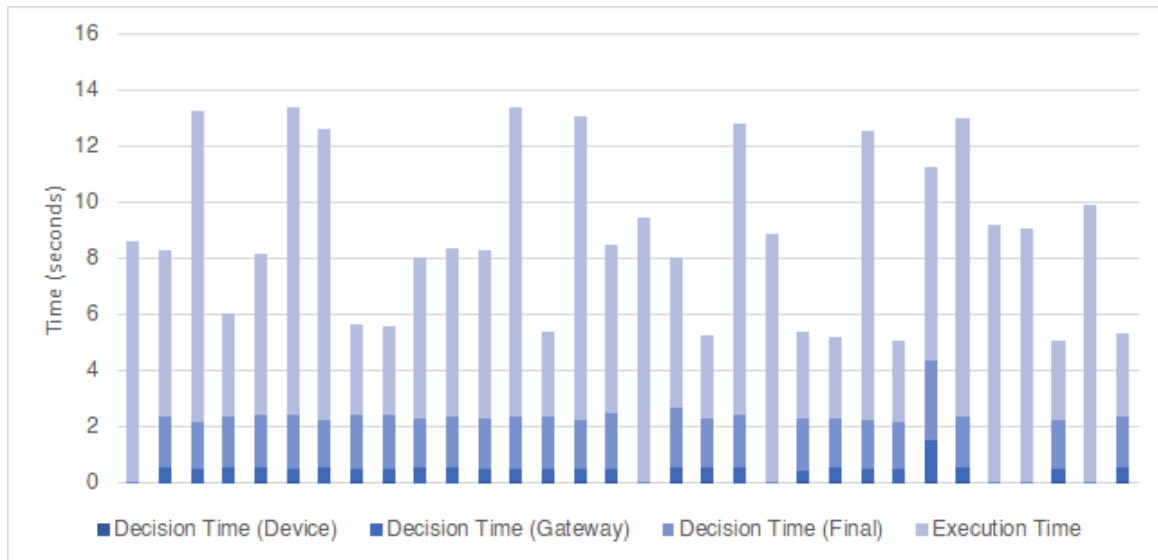
## Περισσότερα Αποτελέσματα

Εξίσου σημαντική για την αξιολόγηση του αλγορίθμου μας είναι η μελέτη του χρονικού και υπολογιστικού κόστους που προσθέτει στο σύστημά μας.

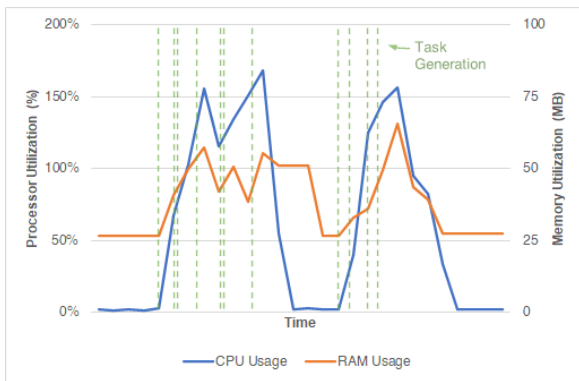
Όπως φαίνεται και στο σχήμα 10, ο αλγόριθμος SGRM προσθέτει 2 δευτερόλεπτα κατά μέσο όρο στον συνολικό χρόνο εκτέλεσης των εργασιών ενός συστήματος, εκτός των περιπτώσεων όπου η συσκευή επιλέγει απευθείας να εκτελέσει την εκάστοτε εργασία τοπικά.

Στα σχήματα 11α και 11β απεικονίζεται το υπολογιστικό κόστος του αλγορίθμου, όταν αυτός εκτελείται σε μια συσκευή και σε έναν κόμβο edge αντίστοιχα. Όπως βλέπουμε, η κατανάλωση μνήμης του αλγορίθμου δεν ξεπερνά τα 100MB ανά πάσα στιγμή, ενώ η κατανάλωση επεξεργαστικής ισχύς είναι λιγότερο αμελητέα, με τον αλγόριθμο να καταλαμβάνει μέχρι και 2 πυρήνες του επεξεργαστή της συσκευής edge.

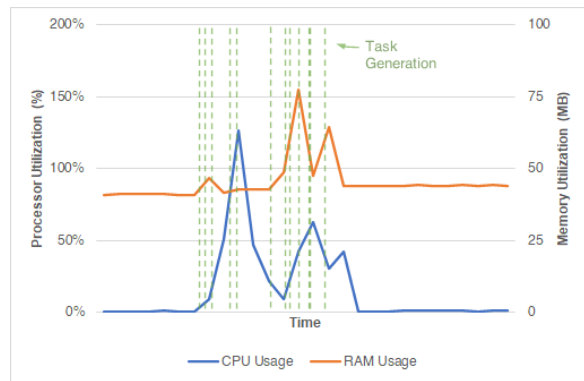
Σχετικά με την κλιμακωσιμότητα του αλγορίθμου, επαναλαμβάνουμε ένα από τα παραπάνω πειράματα αυξάνοντας τον αριθμό των κόμβων edge που συμμετέχουν στο σύστημα και παρατηρούμε την επίδραση αυτής της αύξησης στην συμπεριφορά του αλγορίθμου. Όπως βλέπουμε στα σχήματα 12α και 12β, ο χρόνος απόφασης



Εικόνα 10: Χρονικό Κόστος Αλγορίθμου



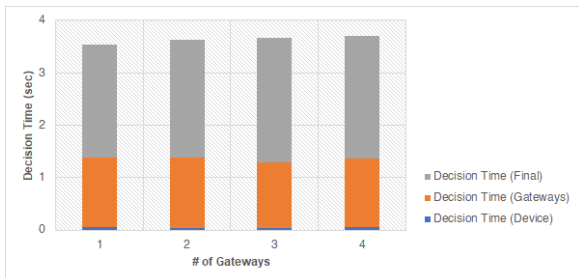
(a) Μονάδα Συσκευής



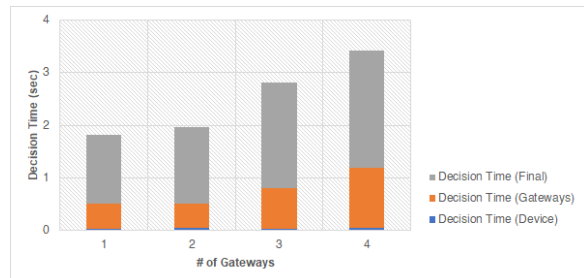
(b) Μονάδα Κόμβου

Εικόνα 11: Υπολογιστικό Κόστος Αλγορίθμου

είναι ίσος με τον χρόνο απόκρισης του αργότερου κόμβου edge και δεν επηρεάζεται άμεσα από τον αριθμό των συνδεδεμένων κόμβων.



(a) Προσθήκη κόμβων σε αύξουσα ισχύ

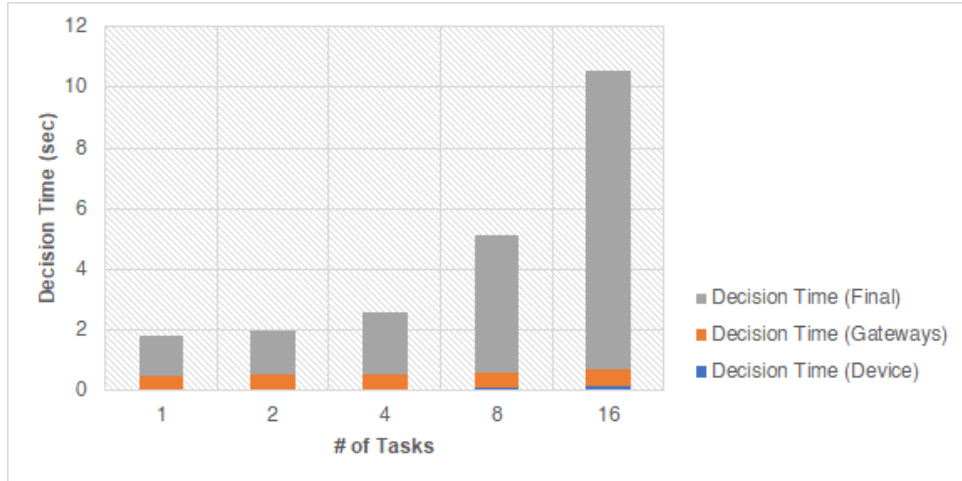


(b) Προσθήκη κόμβων σε φθίνουσα ισχύ

Εικόνα 12: Επίδραση σύνδεσης περισσότερων κόμβων

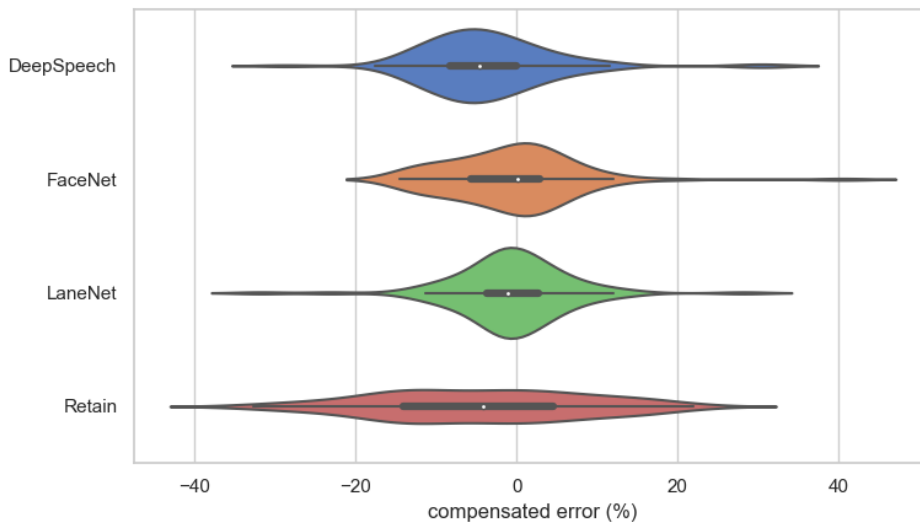
Επίσης, στην εικόνα 13 απεικονίζεται η συμπεριφορά του αλγορίθμου κατά την άφιξη πολλαπλών εργασιών ταυτόχρονα. Παρά την φαινομενικά ραγδαία χρονική αύξηση που φαίνεται στο σχήμα, οι ικανοποιητικές επιδόσεις του αλγορίθμου

στην διαχείριση ‘ριπών’ εργασιών που πραγματοποιήσαμε παραπάνω υποδεικνύει πως αυτή η συμφόριση συμβαίνει μόνο κατά την ταυτόχρονη άφιξη εργασιών και αποφεύγεται εφόσον υπάρχει μια μικρή έστω χρονική διαφορά μεταξύ των χρόνων αύξησης των εργασιών.



Εικόνα 13: Επίδραση ταυτόχρονης άφιξης περισσότερων εργασιών

Τέλος, στο σχήμα 14 παρουσιάζουμε την αποτελεσματικότητα του μηχανισμού πρόβλεψης χρόνου εκτέλεσης εργασιών κατά την εκτέλεση των σεναρίων εργασιών που περιγράψαμε παραπάνω. Όπως βλέπουμε, το ποσοστό λάθους του μηχανισμού μας κυμαίνεται κατά κύριο λόγο μεταξύ  $-20\%$  και  $+20\%$ , ποσοστό εξαιρετικά ικανοποιητικό για τις ανάγκες της εργασίας, δεδομένης της εγγενούς τυχαιότητας που παρουσιάζουν οι εφαρμογές τεχνητής νοημοσύνης που χρησιμοποιήθηκαν.



Εικόνα 14: Ποσοστό Λάθους Μηχανισμού Πρόβλεψης



## Τελική Αποτίμηση

Έχοντας υποβάλλει τον αλγόριθμο SGRM σε εκτενή συγκριτική μελέτη, βρισκόμαστε στην θέση να εξάγουμε σημαντικά συμπεράσματα για τον αλγόριθμό μας, την επίδοσή του και τα χαρακτηριστικά του. Συγκεκριμένα, ο αλγόριθμος SGRM παρουσιάζει τα παρακάτω χαρακτηριστικά:

- **Βελτιστότητα:** Ο αλγόριθμος SGRM επισκιάζει τους αφελείς αλγορίθμους σε κάθε σενάριο εργασιών που τον υποβάλλαμε, ενώ οι αποφάσεις εκφόρτωσης που λαμβάνει προσεγγίζουν αυτές του βέλτιστου αλγορίθμου Oracle.
- **Κατανεμημένη Λειτουργία:** Ο αλγόριθμος είναι πλήρως αποκεντροποιημένος, και κάθε συσκευή που συμμετέχει στο σύστημα επικοινωνεί με τους κόμβους και παίρνει αποφάσεις αυτόνομα.
- **Κλιμακωσιμότητα:** Η κατανεμημένη φύση του αλγορίθμου και η ανεξαρτησία του υπολογιστικού και χρονικού κόστους του από το πλήθος των συμμετέχοντων συσκευών και κόμβων τον καθιστούν άκρως κλιμακώσιμο.
- **Χρονικό και Υπολογιστικό Κόστος:** Ο αλγόριθμος προσθέτει μια σχετικά μικρή χρονική καθυστέρηση στην εκτέλεση των εφαρμογών, η οποία μπορεί να μειωθεί περαιτέρω με βελτιστοποίηση του κώδικα και αναβάθμιση των δικτυακών υποδομών του συστήματος edge.
- **Προσαρμοστικότητα:** Οι συσκευές έχουν την δυνατότητα να συμμετάσχουν στον αλγόριθμο και να αναλάβουν χρέη κόμβου κατά το δοκούν.



# Chapter 1

## Introduction

### 1.1 Internet of Things and Cloud Computing

Fueled by the continual advances in networking technologies and the ubiquitous flourishing of artificial intelligence applications, the Internet of Things (or IoT) has enjoyed an unprecedented surge in popularity in recent years. A strict definition for the term is difficult to pin down, but, in short, “Internet of Things” describes an assemblage of computing devices that possess the ability to monitor and interact with the environment and transmit data through the internet. These devices - or “things” - come in a variety of forms, ranging from vehicles and mobile devices to “smart” home appliances, industrial machinery and even biochip transponders embedded into farm animals [16]. The interconnection of these devices through and with the internet, along with the ever-increasing processing capabilities of mobile devices and microcomputers and the recent introduction of the 5G network standard have opened up numerous research opportunities and led industry leaders to rapidly embrace the IoT architecture. [1]

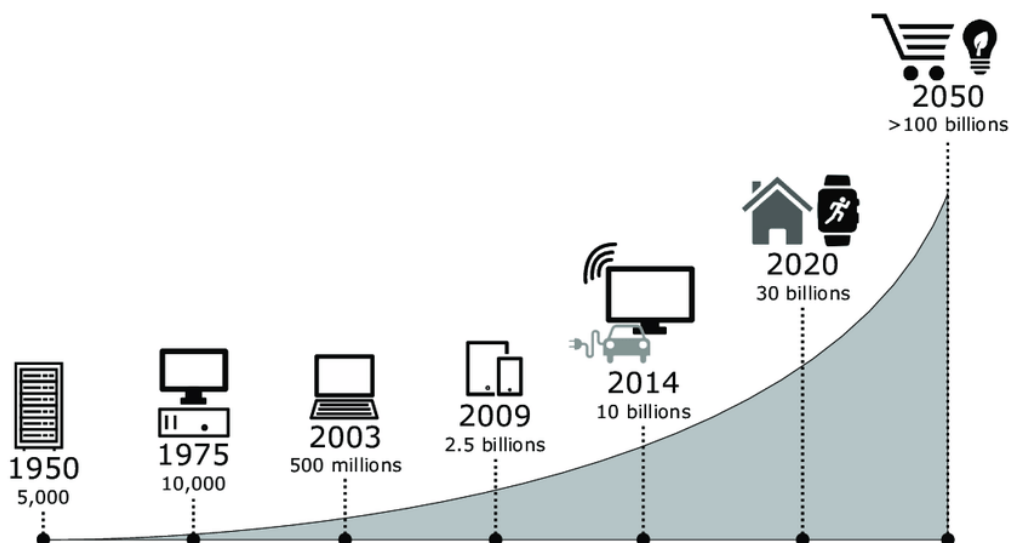


Figure 1.1: Expected Adoption Growth of IoT Devices [1]

Notwithstanding the steadily increasing processing capabilities of IoT devices, the data workloads produced by their sensors often prove too taxing to be processed in place. Hence, the cloud computing paradigm was assumed: powerful servers equipped with hefty multi-core CPUs and substantial amounts of memory, persistently connected to the internet and tasked with processing workloads generated and forwarded by the IoT devices. Stemming from the principles of time-sharing, cloud computing rose as a solution to the ever-growing need of companies for computing power with low upfront costs, and proved to be a good fit for a lot of use-cases of the IoT architecture.

Despite its proven usefulness in numerous implementations, cloud computing falls short in several applications that would be otherwise served greatly by the IoT paradigm. Health and utility providers require user and data anonymity that is difficult to ensure whilst transferring customer information to data centers, and the need for constant connection to the internet creates new vectors of attack for safety-critical devices. At the same time, the expanding employment of artificial intelligence and machine learning applications on IoT devices has introduced new challenges that are not being catered to sufficiently by cloud computing solutions. For example, voice command recognition or health monitoring applications require real time or low latency data processing, while road monitoring auto-drive applications such as the ones employed on Tesla vehicles require several layers of redundancy, as well as the capacity to work even when not directly connected to the internet. Requirements like these eventually gave rise to the “Edge Computing” paradigm.

## 1.2 Edge Computing

Edge Computing is a network architecture aiming to distribute processing workloads between IoT devices (labeled “edge devices”), local mid-powered processing stations (labeled “edge nodes” or “edge gateways”) and, optionally, distant high-end cloud servers. An overview of the architecture is presented in figure 1.2.

Coming back to the IoT paradigm, the IoT devices produce substantial data workloads via their sensors interfacing with the environment, and require data processing that often proves too arduous to be performed locally within sensible time constraints. At the same time, some or all of this data processing needs to be fast to near-instantaneous or the particularities of the situation dictate that it is to be performed on a local network (and not transferred to cloud servers through the internet). Through analysis, these data workloads are identified, prioritized, and forwarded to more capable edge nodes connected to the local network. Though usually not as powerful as off-site cloud servers, these edge

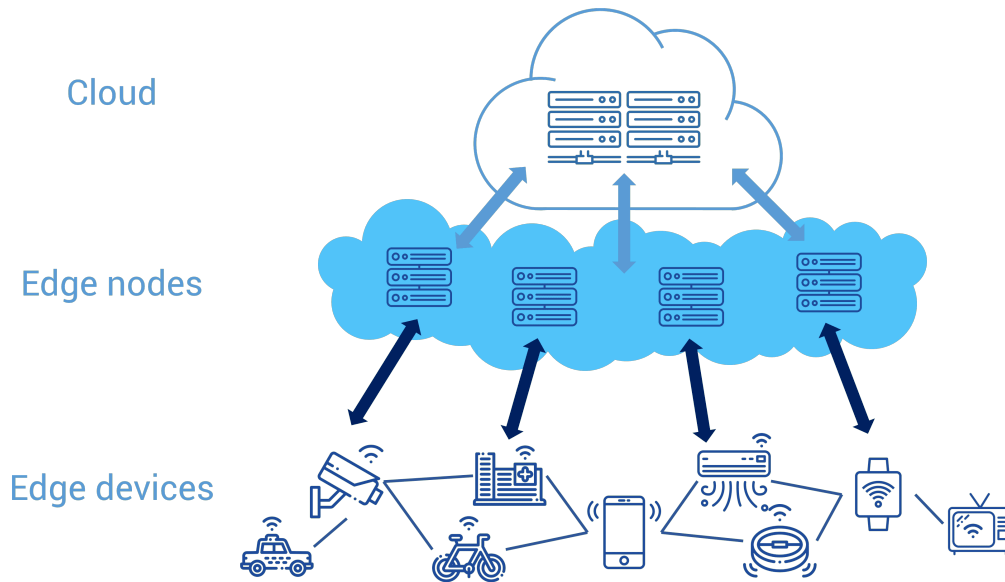


Figure 1.2: Edge Computing Architecture [2]

nodes can perform the more “sensitive” data processing and significantly cut down or even nullify the data needing to be transferred to a cloud server [1].

As a result, the edge computing architecture presents several apparent advantages over other solutions, including the capacity for larger workload processing and real time task execution. The confinement of data transfers over the local network ensures prime transfer speeds and provides a higher level of data security, while the lower dependence on cloud servers and a stable internet connection allows for mobility and geographical independence. Another noteworthy feature is the utilization of low-powered edge nodes over power-hungry data centers, leading to a smaller carbon footprint.

Along with these considerable advantages, edge computing comes with a slew of challenges and potential research opportunities, with resource management being the main focus of this thesis. Beyond that, the heterogeneity and interoperability of potential edge devices and nodes needs to be taken into account, and high levels of redundancy and reliability need to be ensured in proposed edge computing solutions and algorithms. Finally, even though the data locality provided by the edge computing architecture reduces the potential attack vectors on the system, the anonymity and security of data need to be anticipated for, in contrast to cloud computing solutions where the service provider usually provides for these needs.

### 1.3 Resource Management in Edge Computing

One of the centerpieces of current research of edge computing architectures and solutions is the management of the limited resources of edge devices and edge nodes (or gateways, as we will often call them in this work). Every device taking

part in an edge computing system is limited by certain constraints, ranging from limited processing speeds and a finite memory capacity to a low power or energy availability. Also, the network over which an edge computing system operates presents its own limitations in the form of a limited bandwidth or dependence on network reach.

If the aforementioned constraints are modelled as resources, the need to analyze and estimate them becomes an essential first step in the quest to address the resource management problem. After auditing the available resources comes the workload allocation between the edge devices and the edge nodes, as well as the coordination between those devices, in order to achieve the optimal levels of resource utilization and, in turn, the highest quality of service for the applications being executed on the edge.

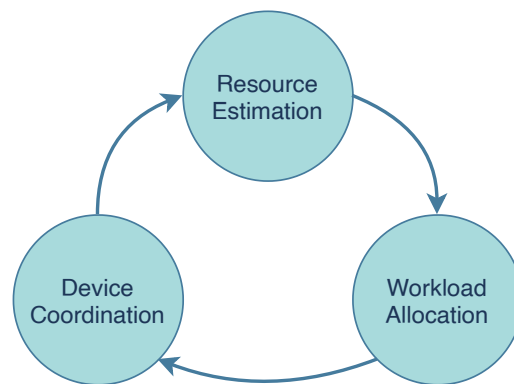


Figure 1.3: Resource Management Cycle

## 1.4 Approaches to Resource Management

While the resource estimation and device coordination challenges can usually be addressed by developing fitting techniques, the workload allocation challenge, once tackled, quickly proves to be an NP-hard problem. Given a certain amount of devices and tasks and the problem of allocating said tasks to said devices optimally, the obvious solution of validating every possible allocation combination and accepting the optimal one, while deterministically producing the optimal workload allocation strategy, is prohibitive to any time-sensitive application due to its immense computational complexity.

As a result, researchers focusing on the resource management challenge have utilized models, theories and paradigms from multiple scientific fields and developed various different approaches to the problem. The competition between devices and applications over scarce resources paves the way for a game theoretic approach, where the resource management problem is modelled as a game and the devices are modelled as the players of said game, competing with each other over the workloads and resources. Once the problem is properly modelled

as a game, it is possible to employ tools and theorems on offer by the field of mathematical and algorithmic game theory in order to find a satisfactory methodology to solving it.

## 1.5 Thesis Overview

The presented work employs a well-established edge architecture of a local network consisting of edge devices and gateways. Tasks are continually generated on the edge devices, and they, in turn, are granted the ability to offload said tasks to the gateways. In our setup, both devices and gateways have resource constraints in the form of limited processing power and memory, while the network connecting them offers a limited bandwidth.

Consequently, an efficient system of coordinating between the devices, estimating the available resources and deciding on which tasks should be offloaded and where to becomes essential. While many different approaches have been developed to tackle this particular challenge, this thesis offers a novel approach with the following contributions:

- We present a model of an edge computing setup, where edge devices can choose to offload generated tasks to edge gateways. We take into account the limited resources and constraints of our system, and model the resource management problem as a deadline miss minimization problem.
- We describe, design and implement SGRM, a distributed game-theoretic resource management algorithm that models the resource management problem as a Stackelberg game where devices act as leaders and decide on which tasks to offload, and gateways act as followers and compete over the tasks in a second-price sealed-bid auction.
- We evaluate our algorithm and compare it to common baselines utilizing edgebench, a workbench of heterogeneous artificial intelligence applications developed specifically for the needs of this thesis.

The remainder of this thesis is organized as follows:

- In chapter 2, we summarize existing work related to our research.
- In chapter 3, we discuss game theoretic models and their usefulness in tackling the resource management challenge.
- In chapter 4, we formulate the resource management problem and put forward our solution, the SGRM algorithm.

- In chapter 5, we provide information related to the technical implementation of our algorithm, and present edgebench, an application workbench especially designed for the needs of this thesis.
- In chapter 6, we assess SGRM, describe the evaluation methodology and analyze the results of our experiments.
- Finally, in chapter 7, we draw conclusions from our work and provide ideas for future research.



# Chapter 2

## Related Work

Over the last years, a significant amount of research has been carried out in order to address the resource management challenges present in edge and cloud computing systems, and an array of interdisciplinary approaches have been employed for that purpose; many with noteworthy success. In this chapter, we offer a short survey of such proposals, focusing on those that employ elements of game & market theory.

### 2.1 Game & Market Theory Based Approaches

In [17], authors design a game-theoretic top-down & bottom-up task allocation algorithm for edge computing systems, while authors in [18] offer a cooperative-competitive approach. Liu et al. in [19] combine a multi-item auction and a congestion game in order to optimize a data offloading decision mechanism in mobile cloud computing environments. Messous et al. in [20] and [21] formulate a computation offloading problem in UAV edge networks, and design two discrete game theoretic algorithms to tackle it with significant success.

A lot of research has also gone into developing market and economic theory based techniques for the resource management challenge. Katsaragakis et al. in [22] offers “DMRM”, a distributed market-based approach for resource management in edge computing systems. Melissaris et al. in [23] put forward “Agora”, a resource management algorithm based on principles of economic theory for many-core systems. Lastly, [24] models the task allocation problem in a software development process as a resource management problem, and offers a Vickrey auction solution to address it, not unlike the one utilized in the present thesis.

### 2.2 Stackelberg Game Approaches

Researchers have had notable success with employing the Stackelberg game (SG) model to a variety of resource management challenges. Authors in [25]

employ a multiple-leader, single-follower Stackelberg game based approach to tackle an anti-jamming problem, while Yang et al. in [26] model the resource allocation problem in energy-hungry data centers after a single-leader, multiple-follower SG. In [27], authors introduce blockchain mining to the edge by constructing collaborative mining networks (CMNs) consisting of non-mining devices and the edge cloud, and formulating the interactions between the edge cloud operators and CMNs after a Stackelberg game, in order to estimate resource prices and demands. [28] follows along the same lines, putting forward a three-stage SG for edge devices participating in the mobile blockchain.

Chen et al. in [29] tackle a resource management problem akin to the one addressed by the present thesis utilizing a reverse Stackelberg model, whereas authors in [30] combine an SG approach with a many-to-many matching game to design a framework for resource allocation in three-tier edge networks. Facing a similar allocation challenge, [31] models the interactions between cloud and edge servers as an SG. Finally, [32] offers a multi-leader, multi-follower SG for the purposes of allocating end users to mobile networks.

# Chapter 3

## Game Theory and Models

### 3.1 Game Theory Fundamentals

Game theory, the branch of applied mathematics that studies the models of conflict and cooperation between rational decision-takers [33], has been widely utilized to tackle challenges present computing systems and networks with significant success. Authors in [34] and [35] provide surveys of economic models and components of pricing theory applied in resource management problems in cloud networks and IoT systems respectively, while authors in [36] offer a detailed presentation of applications of game theory in the field of mobile edge computing (MEC).

A **game** is defined as any situation in which one or more agents can act to maximize their utility through anticipating the responses to his actions by other agents [37]. The agents (or decision-makers), are called **players**, and the decisions they make are called **actions**. In games with multiple decision-making steps, also called extensive-form games, a collection of actions of a player is called this player's **strategy**. The possible results of a game are called **outcomes**, and they are usually quantified into **payoff** or **utility values** [38].

### 3.2 Game Theoretic Models

This diploma thesis focuses on classical games, defined as isolated encounters, devoid of the behavioural regularities found in evolutionary games [39]. Classical games are distinguished further into cooperative and non-cooperative games, depending on the ability of players to form coalitions with each other in pursuit of a common goal [40].

This thesis utilizes elements of non-cooperative (NC) game theory (namely Stackelberg games (SG)), as well as auction theory. These elements are elaborated upon further in the following sections.

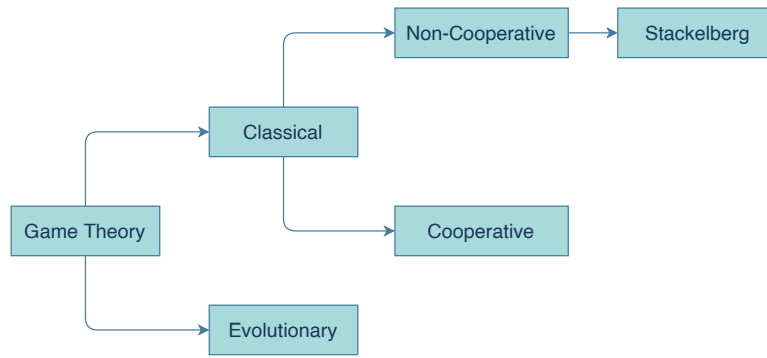


Figure 3.1: Taxonomy of Games

### 3.2.1 Non-Cooperative Games

Non-cooperative games constitute the fundamental type of game studied by the field of game theory. In these games, the players are unable to coordinate and cooperate with each other, but are forced to act independently. [41] puts forward the following features as descriptive of non-cooperative game theory:

- Rules are complete.
- The ultimate decision units are the players.
- Commitments are not available.

A principal concept of non-cooperative game theory is the Nash Equilibrium (NE), defined as a set of strategies (called a strategy profile) such that each player maximizes their payoff, provided that all other players remain fixed in their strategies [40].

Despite its merits, the parallel nature of generic non-cooperative games does not lend itself well to the challenges being addressed by this thesis. For that reason, the sequential Stackelberg game is introduced.

### 3.2.2 Stackelberg Games

A Stackelberg game is a special type of the non-cooperative game, where players are segregated into **leaders** and **followers** [3]. A Stackelberg game is played in cycles, and each cycle is defined as a series of actions:

1. The leaders choose their strategies,
2. The leaders announce their strategies,
3. The followers are informed of the leaders' strategies and, in turn, choose their own strategies,
4. The followers announce their strategies.

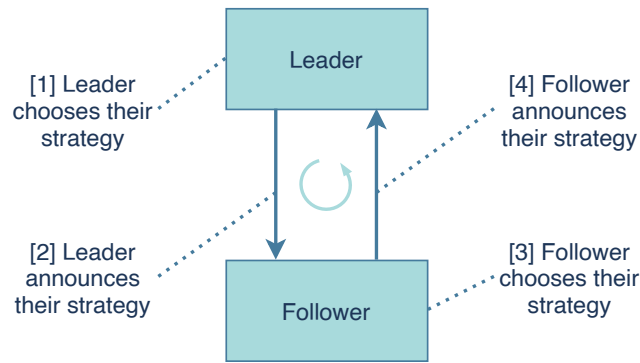


Figure 3.2: Stackelberg Game Cycle

Stackelberg games are classified based on the number of followers and leaders participating. Figure 3.2 provides a visual model of a cycle of a single-leader, single-follower Stackelberg game, while the algorithm in this thesis models the task offloading problem as a single-leader, multiple-follower game.

Similarly to the Nash equilibrium, a set of strategies that provide the optimal outcomes for both leaders and followers participating in a Stackelberg game is defined as the Stackelberg equilibrium of the game [3]. Authors in [36] suggest that a Stackelberg game model constitutes a non-cooperative game model, enhanced with the novel aspects of action observation and stage repetition.

### 3.2.3 Sealed-Bid Auctions

Sealed-Bid auctions provide a very straightforward way of assigning and auctioning value to competing agents. In these auctions, buyers submit sealed bids simultaneously, with no prior knowledge regarding the bidding strategy of the competition. Afterwards, the bids are opened and announced, and the highest bidder is awarded the good or service in question [42].

While first-price sealed-bid auctions are more common, in this thesis we utilize Vickrey (or second-price, sealed-bid) auctions, where the winner of the auction pays the price bid by the second highest bidder. Here, bidding truthfully is proven to be the optimal strategy for all players [43]. In chapter 4.2.1, we take advantage of this property to prove the existence of a Stackelberg equilibrium for our game.

### 3.2.4 Other Approaches

Besides the elements described above, game theory offers a multitude of models and theorems that may prove useful in addressing the diversity of challenges introduced by the edge computing paradigm. Some possibilities would be:

- organizing the devices into coalitions and employing **cooperative game theory**,

- modeling the prolonged interactions of edge devices after **evolutionary games**,
- reinterpreting and analyzing the system in question using **economic theory**.

# Chapter 4

## SGRM: Stackelberg Game-Based Resource Management

This research addresses a common resource management challenge present in edge computing networks. In this chapter, we model the system's architecture, objectives and constraints. Afterwards, we formulate the optimization problem and put forward our proposed solution: the **SGRM** algorithm.

### 4.1 System & Problem Formulation

#### 4.1.1 System Formulation

The edge computing architecture posited in this work is constituted by a number of edge devices interconnected via a local network. The devices are distinguished into two discrete groups:

- **Edge Devices:** IoT devices, ranging from sensors and systems-on-chips to fully-fledged microcomputers. These devices are equipped with processing capabilities but are tasked with carrying out tasks that exceed their limited available resources.
- **Edge Gateways:** Edge computing devices, equipped with their own set of limited resources and tasked with optimally utilizing these resources in order to aid the execution of tasks generated on the edge devices. These gateways can be IoT devices or any other type of device equipped with processing, memory and networking capabilities, such as edge servers and workstations.

Every edge device is assigned an identifying value  $x \in \{1, \dots, X\}$ , and is described by a tuple  $D_x = \{C_x, M_x, N_x, t_x\}$ . In the same manner, every edge gateway is assigned an identifying value  $y \in \{1, \dots, Y\}$ , and is described by a tuple  $G_y = \{C_y, M_y, N_y, t_y\}$ . Variables  $C_x$ ,  $M_x$  and  $N_x$  denote the available

Denotation	Description	Values
$x$	Device ID	$1, \dots, X$
$y$	Gateway ID	$1, \dots, Y$
$C_x$	Available CPU	$0 - 100\%$
$M_x$	Available RAM	$0 - 100\%$
$N_x$	Available Bandwidth	$0 - 100\%$
$t_x$	Current Tasks	$\mathbb{N}$
$z$	Task ID	$1, \dots, Z$
$w_z$	Task Type	$\mathbb{N}$
$\Omega_z$	Task Deadline	$\mathbb{R}_+^*$
$r_z$	Task Reward	$\mathbb{R}_+^*$
$c_z^i$	Maximum CPU Usage	$0 - 100\%$
$m_z^i$	Maximum RAM Usage	$0 - 100\%$
$n_z^i$	Maximum Bandwidth Usage	$0 - 100\%$
$x_z$	Generation Device	$1, \dots, X$
$y_z$	Execution Device	$0, \dots, Y$

Table 4.1: Key System Model Parameters

processing, memory and network resources of the device respectively, while  $t_x$  indicates the number of tasks currently being executed on the device.

Following along the same lines, every task is assigned an identifying value  $z \in \{1, \dots, Z\}$ , and is described by a tuple  $T_z = \{w_z, \Omega_z, r_z, c_z^i, m_z^i, n_z^i, x_z, y_z\}$ .  $c_z^i$ ,  $m_z^i$  and  $n_z^i$  signify the amount of processing power, memory and bandwidth required for the task to be executed on device (or gateway)  $i$ , and are calculated via the task evaluation module described in chapter 4.3.3.  $\Omega_z$  signifies the deadline of the task in seconds, while  $x_z$  and  $y_z$  denote the device wherein the task was generated and the gateway whereto the task was offloaded, respectively. For tasks that were executed locally and not offloaded, the variable  $y_z$  is set to zero. Lastly,  $w_z$  indicates the category (or “type”) of the task produced by the task evaluation module, and  $r_z$  signifies the base reward offered by the operator of the algorithm to the device that carries out the task, as explained in section 4.1.3.

We summarize the denotations of our model in table 4.1, and offer a visualisation of the system architecture in figure 4.1. We also note that the variables  $x$  and  $y$  indicate an edge device and gateway respectively, while the variable  $i$  that we use in some of the following sections indicates a device that can be either of the two.

Having properly defined the system architecture utilized in this research, we move on to formulate the problem addressed in our work.



### 4.1.2 Problem Formulation

Given a set of  $X$  edge devices and  $Y$  edge gateways, a set of  $Z$  tasks is generated over a set period of time on the devices of the edge computing network. We define the objective of our system as the minimization of the number of tasks that overage their deadline:

$$\begin{aligned} \text{minimize} \quad & \sum_{z=1}^Z \delta_z & (4.1) \\ \text{where} \quad & \delta_z = \begin{cases} 1 & \text{if task } z \text{ misses its deadline} \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

We also define the constraints of the problem:

$$\text{Device Constraints :} \quad \forall x, z \quad \text{s.t.} \quad (x_z, y_z) = (x, 0) : \quad \sum_z c_z^x < C_x \quad (4.2)$$

$$\forall x, z \quad \text{s.t.} \quad (x_z, y_z) = (x, 0) : \quad \sum_z m_z^x < M_x \quad (4.3)$$

$$\forall x, z \quad \text{s.t.} \quad (x_z, y_z) = (x, 0) : \quad \sum_z n_z^x < N_x \quad (4.4)$$

$$\text{Gateway Constraints :} \quad \forall y, z \quad \text{s.t.} \quad y_z = y : \quad \sum_z c_z^y < C_y \quad (4.5)$$

$$\forall y, z \quad \text{s.t.} \quad y_z = y : \quad \sum_z m_z^y < M_y \quad (4.6)$$

$$\forall y, z \quad \text{s.t.} \quad y_z = y : \quad \sum_z n_z^y < N_y \quad (4.7)$$

It is important to note that modern CPUs do not offer quantifiable “processing units”, as the constraints imply, but rather provide all their available processing power, which is then split between all tasks currently being executed on the device. Therefore, the processing constraints described by equations 4.2 and 4.5 are always enforced by the central processing unit of the device, and only present an indirect constraint, in the sense of increasing the execution time of a task, pushing it closer to the task deadline. The bandwidth constraints described in equations 4.3 and 4.6 present a similar indirect constraint to the system.

### 4.1.3 Incentive Definition

While completely optional, introducing an incentive mechanism for agents participating in the RM algorithm allows for its application in scenarios where the agents do not share the minimization objective of the system as formulated in chapter 4.1.2, but work strictly for personal gain. [4] presents a survey of mechanisms that could be adapted to be used as incentives for participating in

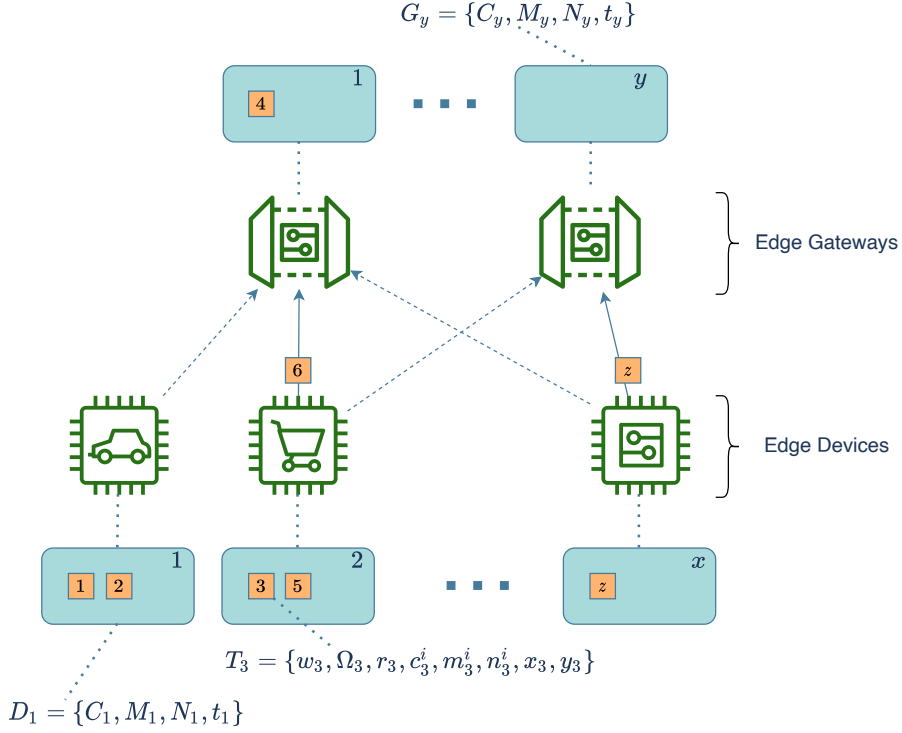


Figure 4.1: Target system architecture

the algorithm, of which we suggest a pricing mechanism [22] or a reputation system [44] as optimal for the nuances of SGRM.

## 4.2 Theoretical Fundamentals

Having strictly formulated the utilized system and the problem in question, we proceed to designate the **SGRM** algorithm.

The fundamental idea behind SGRM is to model the interactions between an edge device and the edge gateways offering to receive and perform the device's computation tasks as a **single-leader, multiple-follower Stackelberg game**. In this game, the edge device takes up the role of the **leader**, and the edge gateways are the **followers**, reacting to the leader's decisions. A single Stackelberg game cycle is defined as a lifetime of a task, and is played as follows:

1. A task is generated on the edge device.
2. The device (leader) evaluates the newly generated task, calculates the utility of executing the task locally, and, if it decides so, advertises the task for offloading.
3. The edge gateways (followers) evaluate the newly advertised task, calculate the utility of receiving the task, and announce the value to the edge device.
4. The device conducts an auction to designate the recipient of the task, using the estimated utility values of itself and the gateways as bids.

For the final step of our algorithm, we employ a **Vickrey auction**, also called a second-price, sealed-bid auction. Here, the different utilities are entered in an auction, and the winner of this auction receives the task and pays the second highest bid to the edge network operator (or receives it, in the case of negative bids). When the task is complete, the winner receives the initial reward value associated with that particular task. Both the game and the auction make use of a digital currency that can be quantified into one of the incentives suggested in chapter 4.1.3.

### 4.2.1 Stackelberg Equilibrium

In order to prove the existence of a Stackelberg Equilibrium (SE) in our game, we will generalize the well known property of truthful bidding dominating other strategies in Vickrey auctions [45] for positive and negative bids. As explained previously, when a player is allocated a task, they pay the value equivalent to the second highest bid (or receive it, when the bid is negative), and once the task is complete, they receive the reward for the task by the edge network operator.

Let  $u_i \in \mathbb{R}_{\neq 0}$  be player  $i$ 's utility for a given task, and let  $b_i \in \mathbb{R}_{\neq 0}$  be that player's bid. The player's payoff for this task is:

$$p_i = \begin{cases} u_i - \max_{j \neq i} b_j & \text{if } b_i > \max_{j \neq i} b_j \\ 0 & \text{otherwise} \end{cases}$$

Obviously, the player's payoff is positive if  $u_i > \max_{j \neq i} b_j$ . We will examine the strategies of overbidding and underbidding.

Assume that  $b_i > u_i$  (overbidding). Then:

- If  $\max_{j \neq i} b_j < u_i < b_i$ , the player wins the auction, but overbidding does not increase the payoff.
- If  $\max_{j \neq i} b_j > b_i > u_i$ , the player loses the auction, and their payoff is zero.
- If  $u_i < \max_{j \neq i} b_j < b_i$ , the player wins the auction, but the payoff is negative. Hence, overbidding decreases the payoff.

Thus, we proved that the strategy of bidding truthfully dominates the strategy of overbidding.

Assume that  $b_i < u_i$  (underbidding). Then:

- If  $\max_{j \neq i} b_j > u_i > b_i$ , the player loses the auction, and the payoff is zero.

- If  $\max_{j \neq i} b_j < b_i < u_i$ , the player wins the auction, but underbidding does not increase the payoff.
- If  $b_i < \max_{j \neq i} b_j < u_i$ , the player loses the auction, and the payoff is zero.

Thus, we proved that the strategy of bidding truthfully dominates the strategy of underbidding.

Therefore, we prove that the strategy of bidding truthfully dominates the strategies of overbidding and underbidding in every case. Thus, any player that participates in the Stackelberg game maximizes their payoff when bidding truthfully.

### 4.3 Modules

To achieve the goals set forth, SGRM is comprised by a number of discrete modules:

- An **edge device module**, containing the task auctioning and offloading mechanisms,
- An **edge gateway module**, executing a task bidding mechanism,
- A **task evaluation module**, executed on all devices.

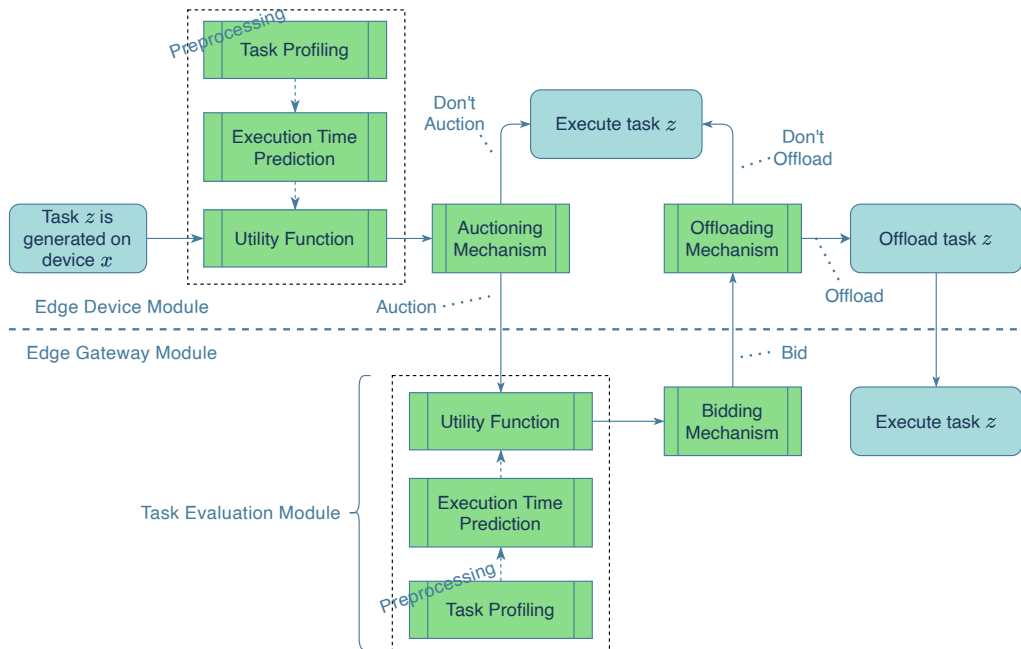


Figure 4.2: SGRM Organization

Figure 4.2 describes the organization of these modules, and in the following subsections we analyze their operations.

### 4.3.1 Edge Device Module

The device module is comprised of a task auctioning and a task offloading mechanism, with both of them being executed separately for each task generated on the device.

---

**Algorithm 1:** SGRM Algorithm (device module)

---

```
Input: timeline // task timeline
Output: count // overdue task counter
begin
  count = 0
  for task in timeline open thread and do
    utility = evaluate(task)
    if utility > threshold then // Don't bother auctioning
      | flag = executeLocally(task) // when utility adequately high
    else
      | auction(task)
      | bidTable = receiveBid(task) // receive bids
      | winner = max(bidTable)
      | flag = offload(task, winner) // true if task overdue
    if flag then
      | count += 1
    end
  return count
end
```

---

Given a newly generated task  $z$ , the device first evaluates it, as described in chapter 4.3.3, and assigns an appropriate utility value to it. Based on this evaluation, the device decides on auctioning (and potentially offloading) the task, or skipping the process and executing it locally. If the device decides to auction the task, the auctioning mechanism advertises the task to the edge gateways, who in turn bid as described in the following section. Once all bids are collected, the task offloading mechanism determines the auction winner and proceeds to transfer the task workload to them.

Algorithm 1 summarizes the functionality of the device module. We define *threshold* as the utility value over which the task should not be considered for offloading. Through experimental analysis we determine that a margin value of 1 produces adequate results in our use-case.

### 4.3.2 Edge Gateway Module

The gateway module is comprised by a single bidding mechanism, invoked once for each task auctioned by the edge devices.

Given an advertised task  $z$ , the gateway follows a similar procedure as the one described in chapter 4.3.1, evaluating the task and assigning a utility value

to it. Next, the task bidding mechanism forwards the estimated utility value as a bid for the auction conducted by the respective device module. Finally, if the gateway wins the auction, it receives the offloaded task and executes it locally. We summarize the functionality of the gateway module in algorithm 2.

---

**Algorithm 2:** SGRM Algorithm (gateway module)

---

```

Input: timeline // task timeline
Output: count // overdue task counter
begin
  count = 0
  for auctioned task open thread and do
    utility = evaluate(task) // Evaluate task
    bid(task, utility) // and place bid
    winner = receiveResult(task)
    if winner then
      flag = executeLocally(task) // true if auction won
      if flag then
        count += 1
    end
  return count
end

```

---

### 4.3.3 Task Evaluation Module

Both edge devices and gateways are equipped with a task evaluation module, consisting of a task profiling mechanism, an execution time prediction mechanism and a utility function. This module allows a device to appraise a task, estimate its execution time and quantify the benefit it will receive for receiving and executing it.

#### Task Profiling Mechanism

Before a device is able to participate meaningfully to the edge system governed by our algorithm, a pre-processing step of profiling the different types of tasks present in the system needs to be carried out on the device, and the effects of multiple tasks being executed in parallel needs to be analyzed and quantified.

To that end, the task profiling mechanism executes multiple combinations of tasks on the target device and records their execution times. Once enough readings have been collected, an analysis of each type of task is performed, as follows:

1. **Chunk Gauging:** Execute the task by itself, and define the work carried out by the device in 1 second as a *chunk*. E.g., a 12 second task consists

of 12 chunks:

$$chunks_{w,i} = wct_{w,i}(1) \quad (4.8)$$

2. **WCET Estimation:** For each task combination tested by the task profiling mechanism, estimate the worst case execution time, following a naive formula:

$$wct_{w,i}(t_i) = chunks_{w,i} \times t_i \quad (4.9)$$

Each time, calculate the error between the estimated value and the actual value.

3. **Error compensation:** Generate a logarithmic trendline of the estimation error percentage as a function of the current number of tasks being executed on the device.

$$compf_{w,i}(t_i) = \alpha_{w,i} \times \ln t_i + \beta_{w,i} \quad (4.10)$$

As a result of the aforementioned process, a tuple  $(chunks_{w,i}, \alpha_{w,i}, \beta_{w,i})$  is generated for every task type  $w$  and device  $i$  combination, which is then utilized by the execution time prediction mechanism. In regards to the amount of different task combinations required to be run to produce sufficient results, figure 4.3 displays the prediction error improvement as more combinations of tasks are executed. In this case, we conclude that even with an average of 25 executed combinations, the time prediction mechanism manages to produce satisfactory results.

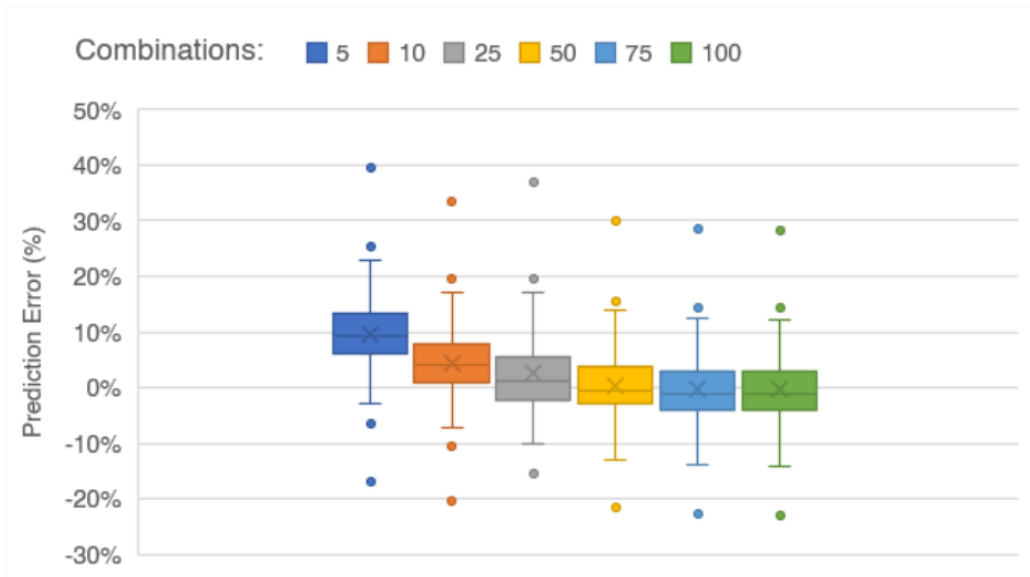


Figure 4.3: Time Prediction Improvement

## Execution Time Prediction Mechanism

Armed with the resulting tuples of the task profiling mechanism, the time prediction mechanism predicts the worst case execution time of a task as a function of the number of tasks currently running on the device, using the equation:

$$wcet_{w,i}(t_i) = chunks_{w,i} \times t_i - compf_{w,i}(t_i) \quad (4.11)$$

$$\text{where } compf_{w,i}(t_i) = \alpha_{w,i} \times \ln t_i + \beta_{w,i} \quad (4.12)$$

We evaluate the accuracy of this prediction mechanism in chapter 6.3.

## Utility Function

Central to the decision making of the algorithm is the utility function, which quantifies the value that a device stands to gain by undertaking a particular task, and doubles as a bid in the sealed-bid auction orchestrated by the corresponding edge device.

First, two penalty functions are estimated. The first penalty function penalizes the final utility according to the estimated execution time of the task, similarly to [18] and [17]. It becomes negative when the task is estimated to miss its deadline:

$$l_{z,x} = \frac{\Omega_z}{\Omega_z - wcet_{w_z,x}(t_x) - wt_{z,x}} \quad (4.13)$$

Table 4.2 offers an elaboration on the notation used in the penalty function above.

A slightly altered version of this penalty function is estimated on the gateway devices, taking into account the extra time required to receive the task:

$$l_{z,x,y} = \frac{\Omega_z}{\Omega_z - wcet_{w_z,y}(t_y) - tt_{z,x,y} - wt_{z,y}} \quad (4.14)$$

The second penalty function penalizes the device according to the number of tasks currently being executed locally:

$$p_x = \begin{cases} c^{t_x} & \text{if } l_{z,x} < 0 \\ \frac{1}{c^{t_x}} & \text{if } l_{z,x} > 0 \end{cases} \quad (4.15)$$

Similarly, when run on a gateway device the second penalty function is estimated according to the equation:



Denotation	Description
$\Omega_z$	Deadline of task $z$
$wcet_{w,i}$	Worst Case Execution Time of task type $w$ on device $i$
$tt_{z,x,y}$	Transfer time of task $z$ from device $x$ to gateway $y$
$wt_{z,i}$	Waiting time for task $z$ to begin execution on device $i$

Table 4.2: Time Variable Definitions

$$p_y = \begin{cases} c^{t_y} & \text{if } l_{z,x,y} < 0 \\ \frac{1}{c^{t_y}} & \text{if } l_{z,x,y} > 0 \end{cases} \quad (4.16)$$

where  $c > 1$  constitutes a weight factor.

Finally, the utility function is estimated on the device:

$$u_{z,x} = \frac{r_z}{l_{z,x} \times p_x} \quad (4.17)$$

and on the gateway:

$$u_{z,x,y} = \frac{r_z}{l_{z,x,y} \times p_y} \quad (4.18)$$



# Chapter 5

## Technical Implementation

Having formally defined SGRM in the previous chapter, here we describe the details of the technical implementation of our algorithm in length. First, we designate the artificial intelligence applications employed for the sake of providing our algorithm with realistic workloads. Then, we describe edgebench, an application workbench designed for managing, coordinating and monitoring the execution of these applications. Finally, we present the resource management algorithms utilized as baselines to compare SGRM against.

### 5.1 Applications

In order to accurately simulate the execution of SGRM under realistic workloads we employ a number of artificial intelligence applications. These applications present real-time, heterogeneous workloads with disparate but significant processing and memory requirements. In the following sections, we showcase these applications, describe their functions and outline their processing and memory requirements, as observed when run on a testing device (here, a Raspberry Pi 4 Model B, as described in chapter 6.1.1).

#### 5.1.1 Deepspeech

Deepspeech [6, 7] is an open-source, real-time text-to-speech (TTS) engine, utilizing a pretrained machine learning model [6] and Google’s Tensorflow [15] platform. As shown in figure 5.1, the application receives as input a wave audio format (WAV) file, preferably short in duration, and outputs the transcribed text.

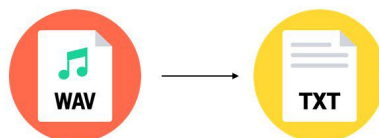


Figure 5.1: Deepspeech function

As figure 5.2 shows, a single deepspeech task requires a miniscule amount of RAM and an average, albeit consistent, percentage of processing power throughout its lifetime of 12 seconds when run isolated on the testing device.

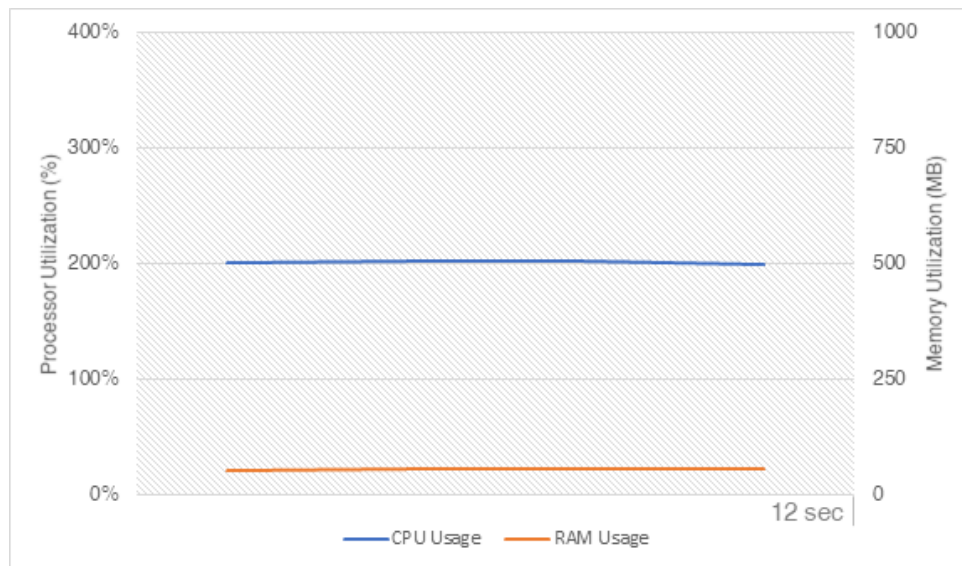


Figure 5.2: Deepspeech Resource Utilization

### 5.1.2 Facenet

Facenet [8, 9] is a face recognition and classification algorithm, implementing the machine learning techniques described in [8] on the Tensorflow platform [15]. In its regular operating mode, the algorithm receives a collection of face images in JPG format and classifies them in discrete clusters, as shown in figure 5.3.

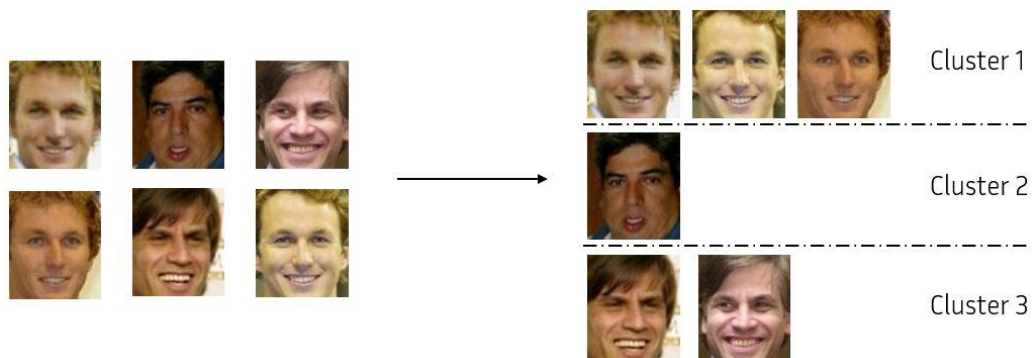


Figure 5.3: Facenet function

For the purposes of this thesis, the algorithm is utilized as a real-time face recognizer: after being provided with a database of “accepted” faces, it receives an image containing a single face, and is tasked with discerning whether the face in question coincides with one of the accepted faces.

Unlike deepspeech, facenet presents a more considerable workload in terms of memory usage, occupying upwards of 500 MB of RAM on the testing device (figure 5.4). In contrast, its processing requirements are significantly lower, with a single CPU core bearing the brunt of the processing throughout the 6.6 seconds of execution.

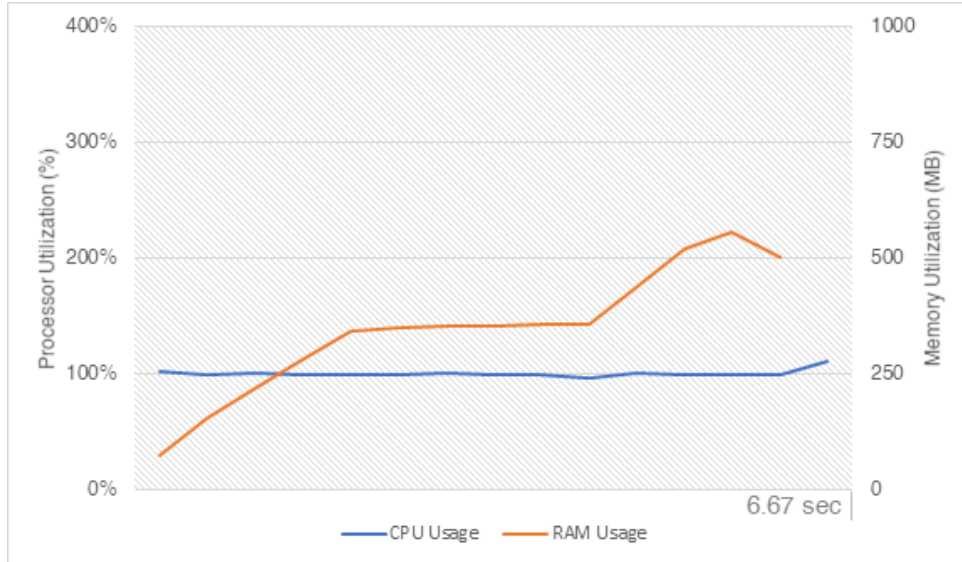


Figure 5.4: Facenet Resource Utilization

### 5.1.3 Lanenet

Lanenet [10, 11] is a Tensorflow implementation [15] of a real-time lane detection algorithm based on a Deep Neural Network [10]. As shown in figure 5.5, the application receives a JPG formatted image of a road as input; then, it promptly detects and outputs the discrete lanes of traffic in the image in question.

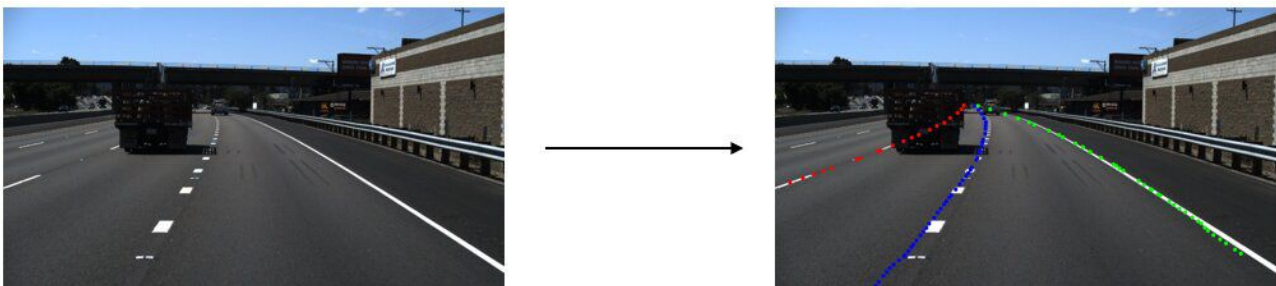


Figure 5.5: Lanenet function

As shown in figure 5.6, a lanenet task presents a short but taxing workload for our testing device. More precisely, a single lanenet task requires less than 2 seconds to execute, provided it is run isolated from other applications, but it occupies between 1 and 3 CPU cores, as well as 200 MB of RAM to that end. Special note is also taken of the variance of the processing and memory usages,

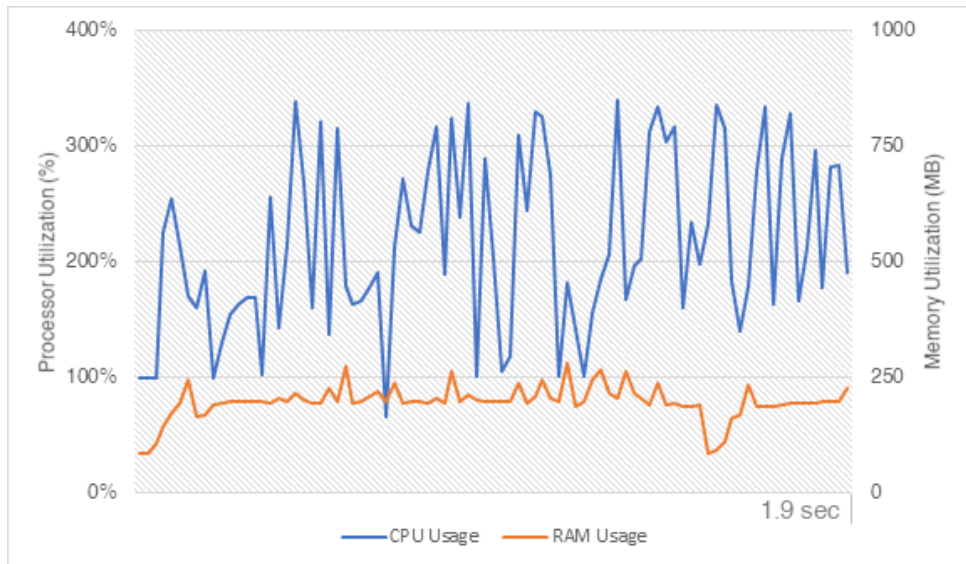


Figure 5.6: Lanenet Resource Utilization

in contrast to the other applications that maintain a mostly constant resource utilization.

### 5.1.4 Retain

RETAIN [12, 13] utilizes the Theano ML library [14] to offer a heart failure prediction model for healthcare use-cases. The application implements the algorithm described in [12], and outputs predictions once provided with a patient's health record in CSV format, as shown in figure 5.7.

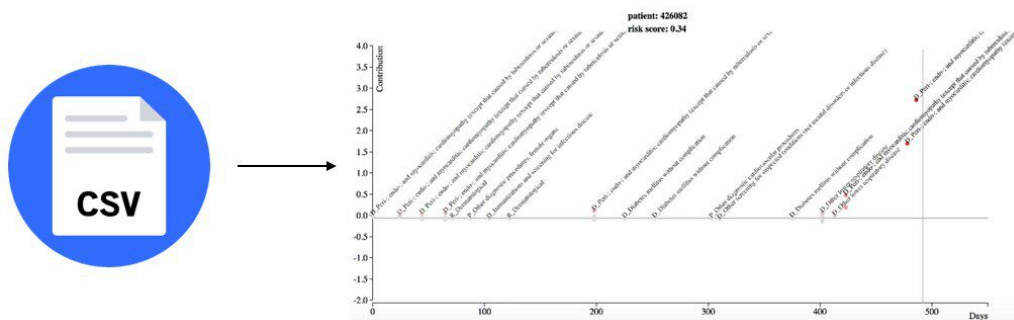


Figure 5.7: Retain function

Retain constitutes the lightest of the workloads presented in this section in terms of resource requirements, requiring only 1 processing core and less than 60 MB of memory for the entire duration of its execution, which took 11 seconds in total on our testing device (figure 5.8).

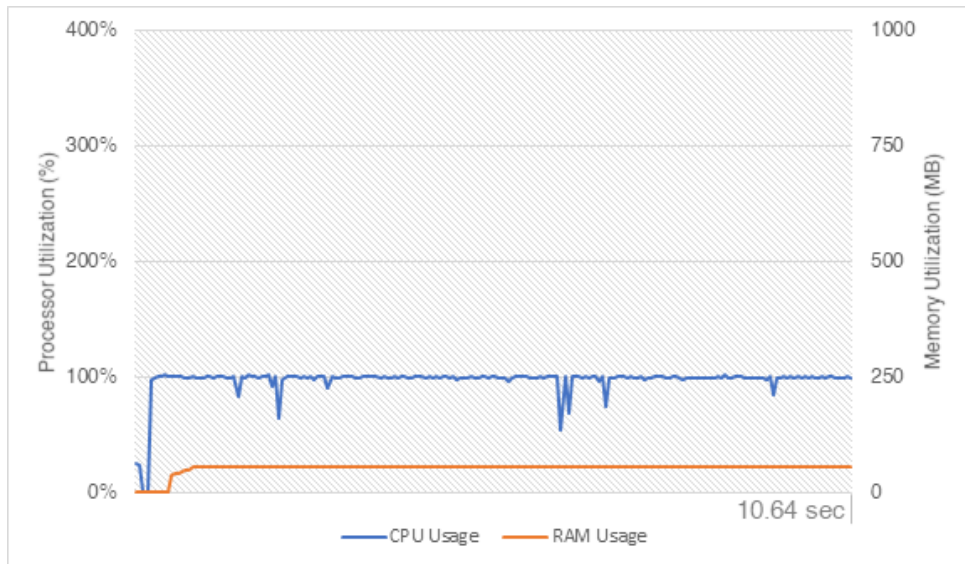


Figure 5.8: Retain Resource Utilization

## 5.2 Application Workbench

Having collected and properly configured the applications described in the previous section, the need to administer, monitor and coordinate them arises. To that end, we design **edgebench** [5], an all-in-one application workbench written in Shell and Python 3. A short overview of the workbench is offered below.

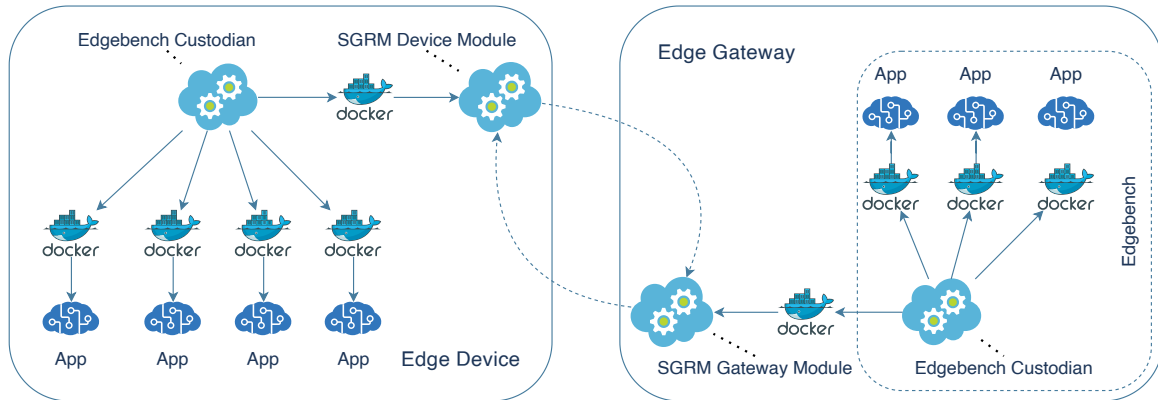


Figure 5.9: Edgebench organization

The workbench addresses the following challenges:

- **Deployment** of the application workbench on a variety of architectures and devices is made easy and fast through the dockerization [46] of the applications into discrete containers. This removes the need for prior configuration or research into the nuances of the device (aside, of course, from the need to install docker), while also allowing the quick addition or removal of any other application onto the workbench.
- **Coordination** of the workbench and the algorithm modules with the applications is achieved through a *custodian service*, designed specifically

for this purpose. This service, programmed in Python, initiates the applications, provides them with input, monitors their execution and keeps comprehensive logs of a myriad of details, essential both for the orderly operation of the workbench and the evaluation of the algorithm, as shown in chapter 6.

- **Communication** between the custodian service and the algorithmic modules being executed on the device is attained through pandas matrices, which hold extensive information on the state of the device and the applications being executed. On the other hand, communication between algorithmic modules running on different devices is realized through application of the popular MQTT communication protocol [47].

## 5.3 Algorithmic Baselines

In order to effectively and meaningfully evaluate SGRM, we design and implement a number of baseline algorithms, against which we will compare our algorithm. By assigning identical workloads to SGRM and the baselines, we can extract important information on the efficiency of our algorithm, as shown in chapter 6.

### 5.3.1 Offload None & Offload All

An obvious first candidate for a comparison baseline is an algorithm that does not allow the offloading of any tasks from the edge devices to the connected gateways, but forces the devices to execute all tasks locally. Thus, we are able to gauge the effectiveness of our algorithm compared to it being entirely absent from the edge computing system. We present this algorithm, coined “Offload None”, in algorithm 3.

---

**Algorithm 3:** Offload None Algorithm

---

```

Input: timeline                                     // task timeline
Output: count                                     // overdue task counter
begin
  count = 0
  for task in timeline do
    flag = executeLocally(task)                       // True if task overdue
    if flag then
      count += 1
    end
  return count
end

```

---



Crossing over to the other side of absolute decision making, we design the “Offload All” algorithm, which instructs the devices to immediately offload any workloads generated on them to a valid gateway. While representing another obviously suboptimal offloading decision, this algorithm provides another valuable metric for SGRM to compare against. “Offload All” is presented in algorithm 4.

---

**Algorithm 4:** Offload All Algorithm

---

```

Input: timeline                                // task timeline
Output: count                                  // overdue task counter
begin
  count = 0
  for task in timeline do
    flag = offload(task)                          // True if task overdue
    if flag then
      count += 1
    end
  return count
end

```

---

### 5.3.2 Oracle

Finally, it is essential to specify how well SGRM emulates the behaviour of an *optimal* (as defined in chapter 4.1.2) offloading algorithm. To that end, we put forward the *Oracle* prediction algorithm, a brute-force, exhaustive-search algorithm that possesses the details of the workload a priori and infers the optimal task offloading scenario by carrying out and experimentally evaluating every possibility. It’s important to mention that the Oracle algorithm grows exponentially in time as the number of tasks increases. Hence, we only employ this algorithm in the “Burst” workload scenario for a maximum of 8 generated tasks, as presented in chapter 6.2.1.

We present the algorithmic design in algorithm 5.

---

**Algorithm 5:** Oracle Algorithm

---

```
Output: bestCount // overdue task counter
Input: timeline // task timeline
begin
  bestCount = 0
  combinations = iterate(timeline)
  for combination in combinations do
    for task, gateway in combination do
      if gateway then // nonzero if task is destined
        | flag = offload(task, gateway) // for offloading
      else
        | flag = executeLocally(task)
      if flag then
        | count += 1
      end
    if count < bestCount then
      | bestCount = count
    end
  return bestCount
end
```

---

# Chapter 6

## Experimental Evaluation

In this chapter, we perform an experimental assessment of SGRM in order to evaluate its performance and illustrate the validity of our resource management approach. First, we present the experimental setup we have prepared and showcase the participating devices. Afterwards, we carry out an extensive comparative study, submitting SGRM and the baseline algorithms to an extensive array of workload scenarios. Finally, we organize and present the results of our study, and utilize them to arrive to a final verdict for our algorithm.

### 6.1 Experimental Setup

In order to provide a realistic framework on which to execute and assess the algorithms, we put together a real edge network, consisting of a diverse array of devices, showcased briefly below.

#### 6.1.1 Raspberry Pi 4 Model B

The Raspberry Pi 4 Model B is the latest rendition of the popular Raspberry Pi microcomputer series, put together by the charitable Raspberry Pi foundation. This Single-Board Computer (SBC) offers a significant performance step-up from the previous versions, while maintaining a small size factor. The technical specifications of the Pis employed in this thesis are <sup>1</sup>:

- **Processor:** Quad-core Cortex-A72 32-bit SoC @ 1.5GHz
- **Memory:** 4GB LPDDR4 RAM
- **Graphics Processor:** Broadcom VideoCore VI
- **Networking:** 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE, Gigabit Ethernet

---

<sup>1</sup>Detailed specifications can be found at <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications>

- **Operating System:** Raspbian Stretch Lite
- **Architecture:** ARMv7

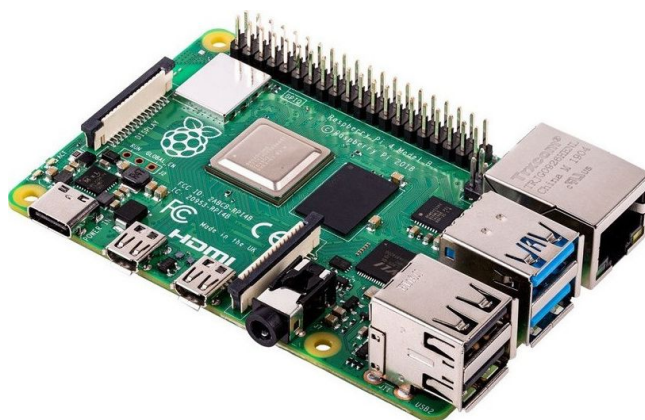


Figure 6.1: Raspberry Pi 4 Model B

We note that the processing unit of this device, albeit branded as ARMv8, in actuality implements an upgraded version of the ARMv7 instruction set.

### 6.1.2 Nvidia Jetson TX1

The Jetson TX1 board is a fully-fledged development platform, equipped with Nvidia’s powerful Tegra X1 System on a Chip (SoC) and designed with visual computing applications in mind. The Tegra X1 SoC offers four ARMv8 processing cores and a Maxwell-Based graphics processing unit, and is notably the basis for the popular Nintendo Switch console. The device’s technical specifications are <sup>2</sup>:

- **Processor:** Quad-core Cortex-A57 64-bit SoC @ 1.9 GHz
- **Memory:** 4GB LPDDR4 RAM
- **Graphics Processor:** Maxwell 256-core GPU
- **Networking:** Gigabit Ethernet
- **Operating System:** Ubuntu Linux
- **Architecture:** ARMv8

---

<sup>2</sup>Detailed specifications can be found at <https://developer.nvidia.com/embedded/jetson-tx1-developer-kit>

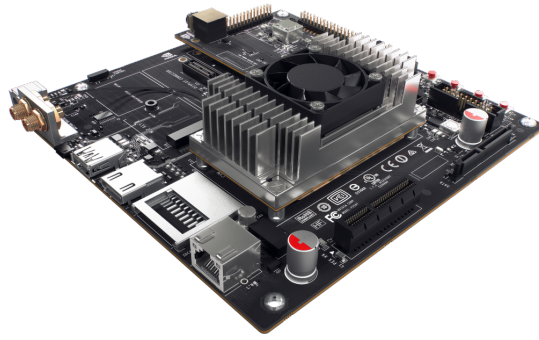


Figure 6.2: Jetson TX1 Developer Kit

### 6.1.3 Nvidia Jetson Nano

The Jetson Nano board is another SBC offering by Nvidia, a powerful embedded device optimized for running neural networks. The Nano board is equipped with a slightly downgraded version of the Tegra X1 SoC present in the Jetson TX1, and the exact technical specifications are <sup>3</sup>:

- **Processor:** Quad-core Cortex-A57 64-bit SoC @ 1.43 GHz
- **Memory:** 4GB LPDDR4 RAM
- **Graphics Processor:** Maxwell 128-core GPU
- **Networking:** Gigabit Ethernet
- **Operating System:** Ubuntu Linux
- **Architecture:** ARMv8



Figure 6.3: Jetson Nano Developer Kit

---

<sup>3</sup>Detailed specifications can be found at <https://developer.nvidia.com/embedded/jetson-nano-developer-kit>

### 6.1.4 64-bit Virtual Machine

Besides the microcomputers described above, we also utilize two powerful 64-bit virtual machines as edge gateways in the “Sustained Workload” scenarios presented in the following section. The VM’s specifications are:

- **Processor:** Octa-core Intel Xeon 64-bit @ 2 GHz
- **Memory:** 4GB DDR4 RAM
- **Networking:** Gigabit Ethernet
- **Operating System:** Ubuntu Linux
- **Architecture:** AMD64

## 6.2 Comparative Study

After configuring the application workbench in chapters 5.1 - 5.2, implementing the algorithms described in chapters 4.3 and 5.3 in the Python 3 programming language, and setting up an edge network with the devices showcased in chapter 6.1, we devise workload scenarios and conduct a comparative study in order to properly and meaningfully assess the performance and characteristics of SGRM.

### 6.2.1 Burst Workloads

First, we submit the algorithms to a variety of “burst” workloads, consisting of a small number of tasks generated in short succession on the edge devices. Specifically, we design 3 separate workload scenarios, coined **8over1**, **8over2** and **8over4**, with the prefixed and suffixed numbers indicating the number of tasks and devices participating in the scenario, respectively. An overview of the scenario parameters and architecture is provided in table 6.1 and figure 6.4, respectively.


Scenario	Tasks	Devices	Gateways	Distribution	Mean	Std. Dev.
8over1	8	1	1	 Gaussian	10	5
8over2	8	2	1		10	5
8over4	8	4	1		10	5

Table 6.1: Burst Scenarios: Parameters

In all three scenarios, we utilize the Jetson TX1 device as the sole gateway of the edge system, since connecting more gateways or using one of the virtual machines as a gateway proves to be excessive for such a limited number of tasks. Furthermore, employing 8 tasks and a single gateway in this scenario equates

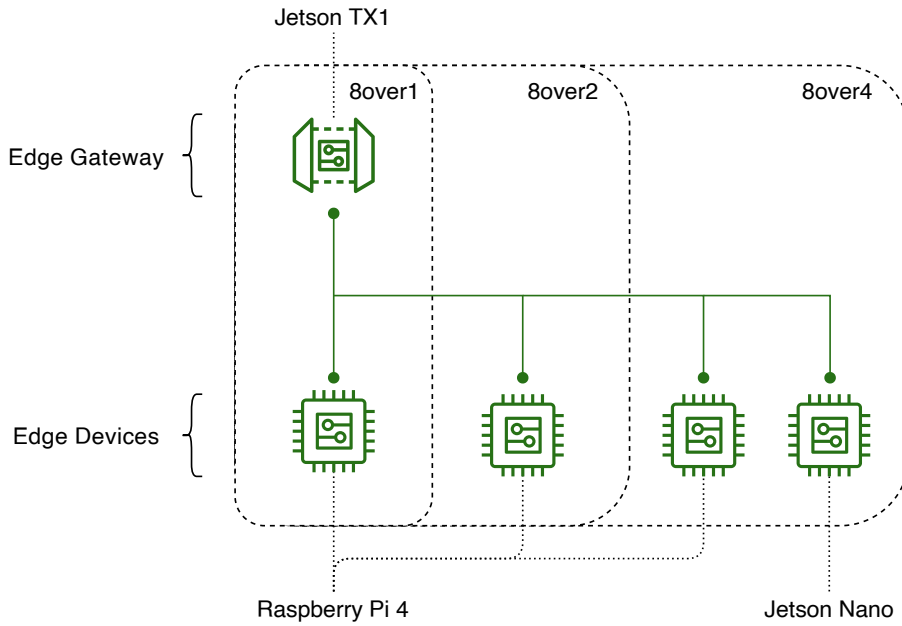


Figure 6.4: Burst Scenario System

to  $2^8$  offloading possibilities, allowing for the execution of the Oracle algorithm in a realistic time frame. We present the results of this evaluation in figure 6.5.

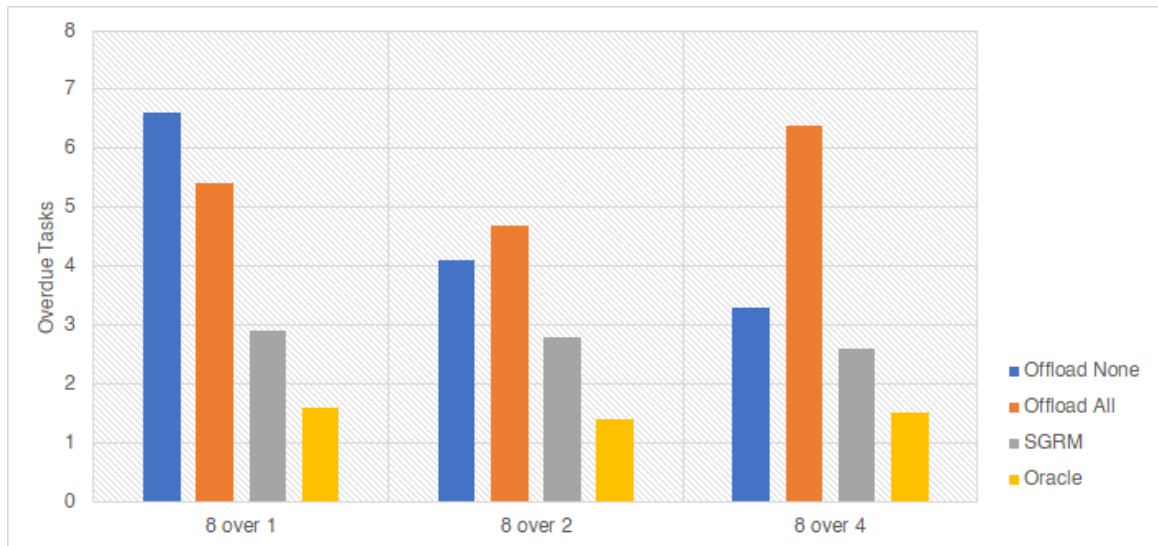


Figure 6.5: Burst Scenario Results

Evidently, when tasked with managing and allocating a quick burst of resource-heavy tasks, the SGRM algorithm produces satisfactory results, outshining both naive offloading algorithms, and achieving performances comparable to those of the optimal Oracle algorithm.

### 6.2.2 Sustained Workloads

Next, we devise an array of “sustained” workload scenarios, where the system in question is called to decide on offloading a larger amount of tasks, generated

through a wider time frame. We design 4 separate scenarios, labeling them **32over3**, **64over3**, **64over5** and **128over5** and randomly generate the task creation times according to the Gaussian and Poisson distributions; thus, we create 8 distinct workload scenarios to test our algorithm with:



Scenario	Tasks	Devices	Gateways	Distribution	Mean	Std. Dev.
32over3 <sub>g</sub>	32	3	1	 Gaussian	30	15
64over3 <sub>g</sub>	64	3	1		30	15
64over5 <sub>g</sub>	64	5	2		60	30
128over5 <sub>g</sub>	128	5	2		60	30
32over3 <sub>p</sub>	32	3	1	 Poisson	30	-
64over3 <sub>p</sub>	64	3	1		30	-
64over5 <sub>p</sub>	64	5	2		60	-
128over5 <sub>p</sub>	128	5	2		60	-

Table 6.2: Sustained Scenarios: Parameters

In these workload scenarios, the increased number of tasks prohibits a comparison with the Oracle algorithm or any other exhaustive search task allocation algorithm. Indicatively, even the smallest scale scenario of 32 tasks generated over 3 devices with a single available gateway leads to  $2^{32}$  distinct offloading possibilities, translating to a total evaluation time of thousands of years.

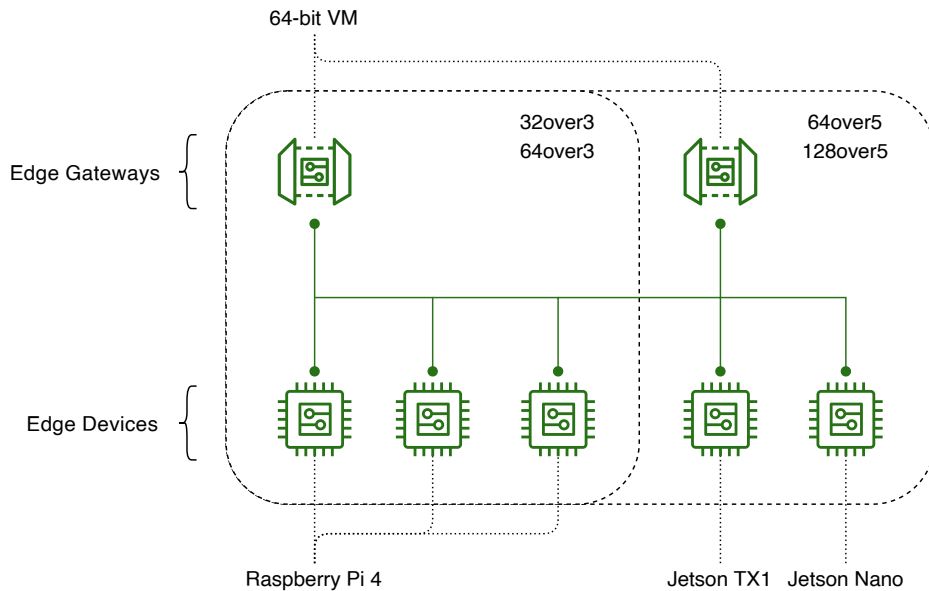


Figure 6.6: Sustained Scenario System

We provide an overview of the device configuration in figure 6.6, and the results of this evaluation in figure 6.7. As the chart shows, SGRM exhibits the ability to effectively handle and allocate a large number of tasks over a prolonged period of time, surpassing both naive baseline algorithms in all tested scenarios.



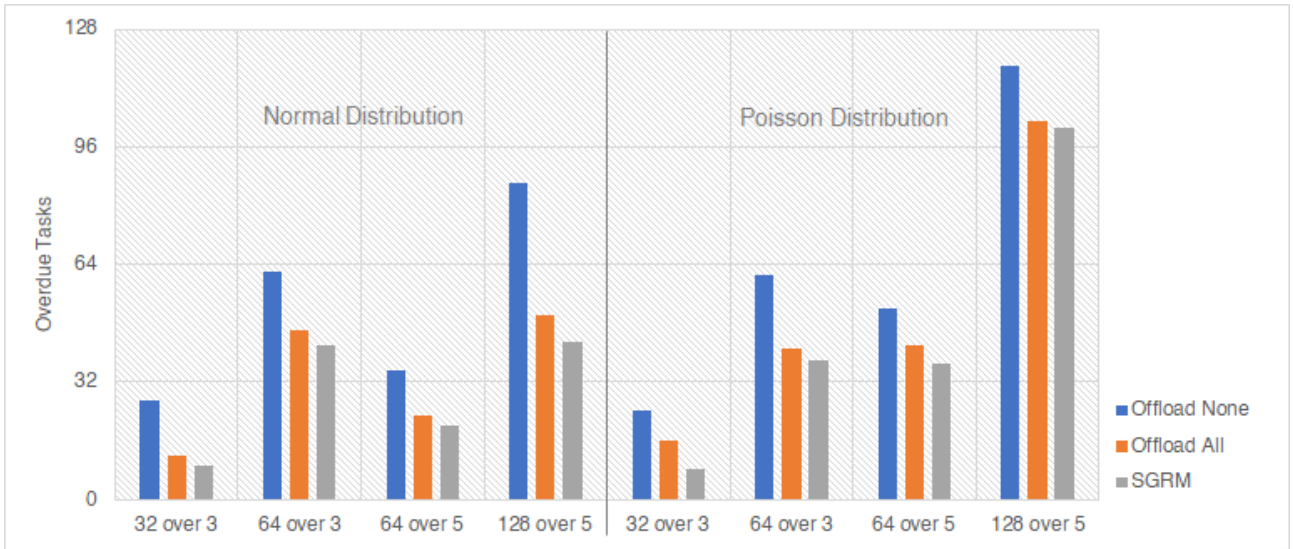


Figure 6.7: Sustained Scenario Results

## 6.3 Implementation Specifics

Having evaluated the performance of the SGRM algorithm in terms of achieving the primary system objective of minimizing the number of tasks overaging their deadlines, in this section we quantify and assess important features of the algorithm.

### 6.3.1 Deadline Selection

In the workload scenarios described in the previous sections, each task type was assigned a static deadline, as shown in table 6.3 (“Standard” label).

Task Type	Strict	Standard	Loose
Deepspeech	16 sec	20 sec	24 sec
Facenet	10 sec	13 sec	16 sec
Lanenet	2 sec	3 sec	4 sec
Retain	16 sec	20 sec	24 sec

Table 6.3: Task Deadlines

These deadlines were found after a trial-and-error process to provide meaningfully differentiated results between the algorithms used. For comparison, figure 6.8 displays how setting a too strict or too loose deadline fuzzes the results between the competing algorithms. We also note that selecting a deadline for each task randomly from the  $[strict, loose]$  time interval produces similar results to our static deadline selection.

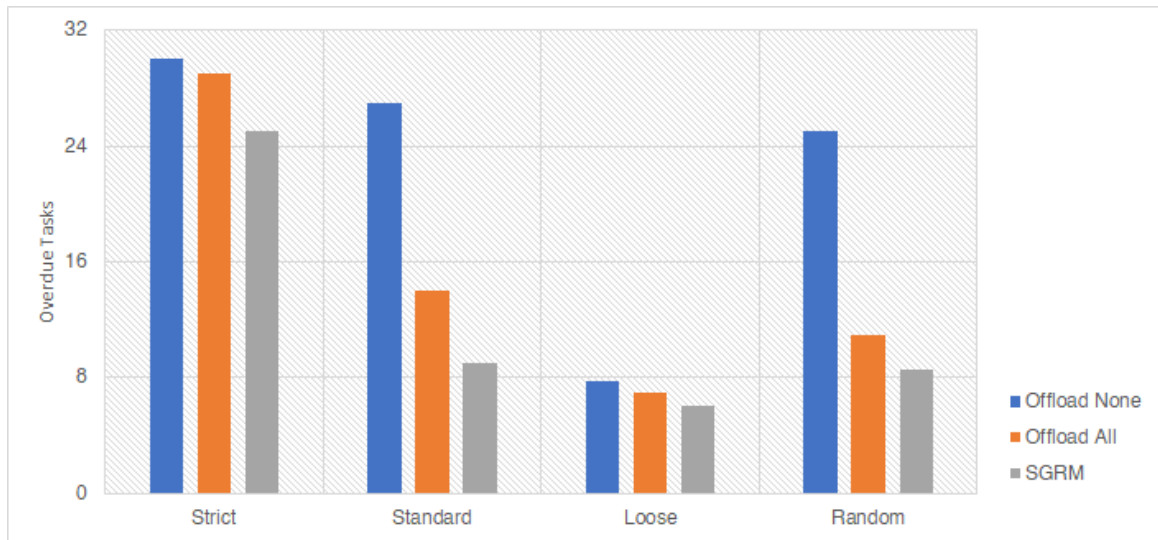


Figure 6.8: Deadline Choice Effect Demonstration

### 6.3.2 Overhead

A crucial characteristic of the algorithm to be taken into account is the overhead, both temporal and computational, that it introduces to a system executing it.

Figure 6.9 displays the time overhead introduced by SGRM during the execution of the 32over3 scenario. As pictured, the algorithm adds an average of 2 extra seconds of decision time to the total lifetime of a task. Albeit considerable, this time overhead is mostly attributed to the network latency present in the edge network. This suspicion is further enforced by the fact that the decision time for tasks that were not auctioned - and hence, no network messages were exchanged - is near-instantaneous.

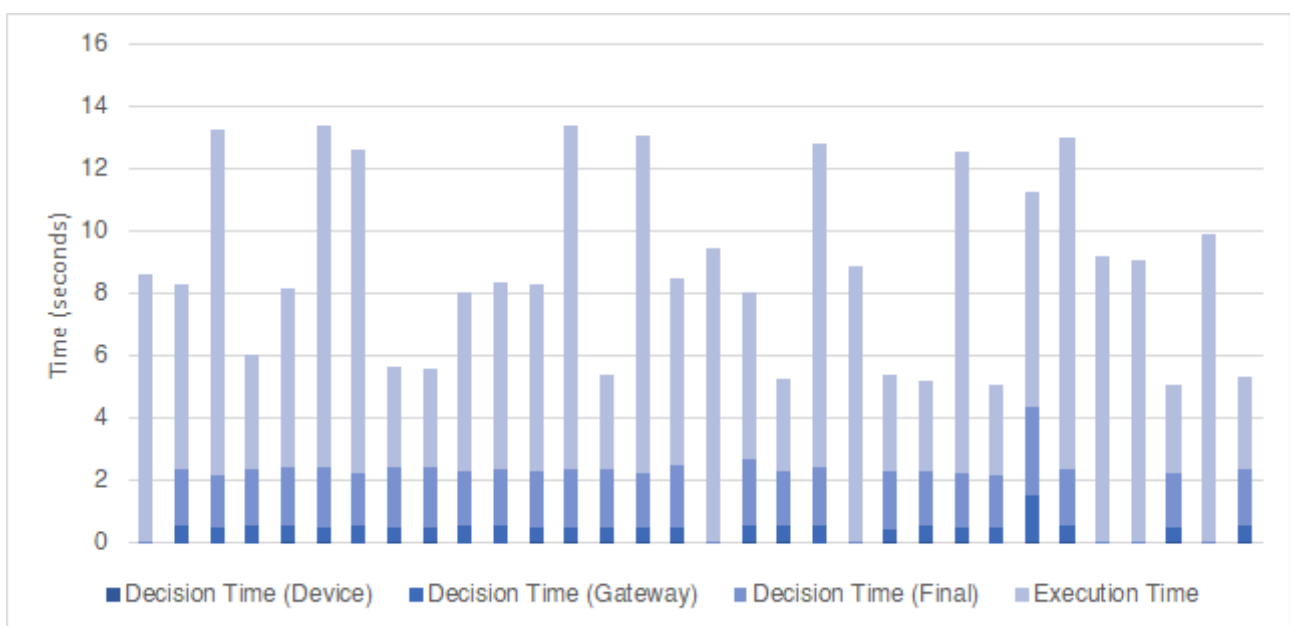


Figure 6.9: Time Overhead of SGRM

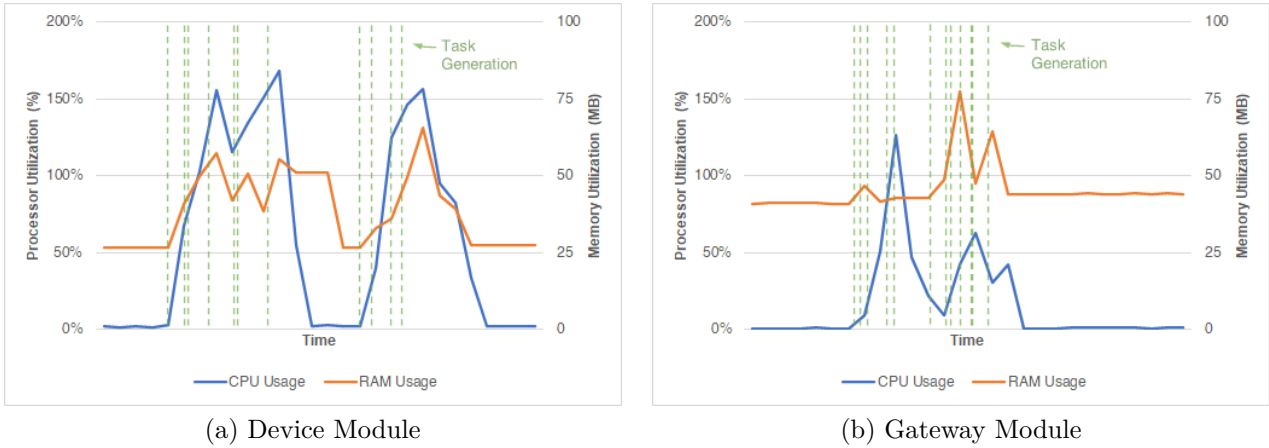


Figure 6.10: Resource Overhead of SGRM

Another important metric is the computational overhead presented by the algorithm. Figures 6.10a and 6.10b present the processor and memory utilization of the algorithm when run on an edge device and gateway respectively, with each green line indicating the generation and auction of a new task. Evidently, SGRM introduces a negligible memory overhead, peaking at a measly 75MB for both gateways and devices. On the other hand, the algorithm manages to utilize a considerable percentage of the device’s processor, taking upwards of two processing cores at times. However, given that no additional slowdown is observed in the execution of the tasks, we can safely assume that this high percentage is due to the optimality of modern CPUs’ in taking advantage of their full processing power when asked to do so.

### 6.3.3 Scalability

Evaluating the scalability of our algorithm is another essential step to assessing the usefulness of it. To that end, we conduct two additional experiments, where we steadily increase the number of participating gateways and tasks being generated simultaneously, and observe the effect of the increase to the decision making time of SGRM.

As shown in figure 6.11, increasing the number of tasks simultaneously generated on the device creates a bottleneck and leads to a steep increase to the decision making times of the device. Despite that observation, the satisfactory performance of our algorithm in the “burst” workload scenarios leads us to believe that staggering the generation of the tasks even by a few milliseconds is enough to bypass this bottleneck.

Regarding the connection between the number of gateways and the decision making time of a device executing the SGRM algorithm, figures 6.12a and 6.12b indicate that the decision time is equal to the response time of the slowest gateway, and does not worsen due to more gateways participating in the edge

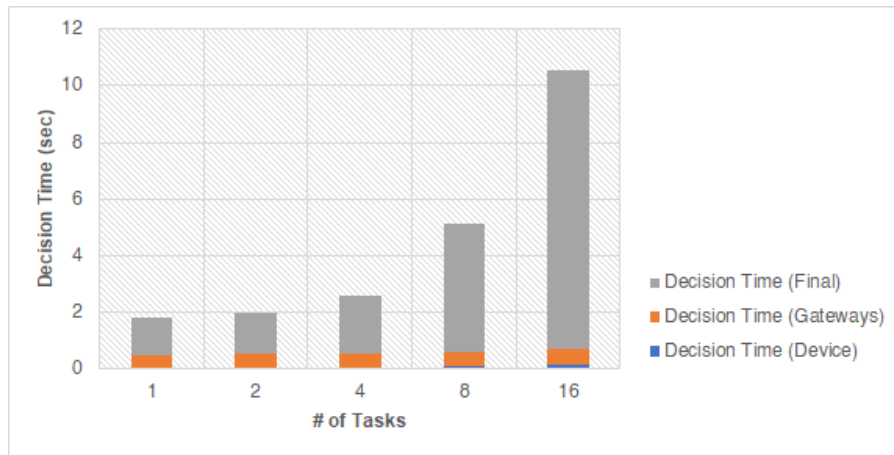
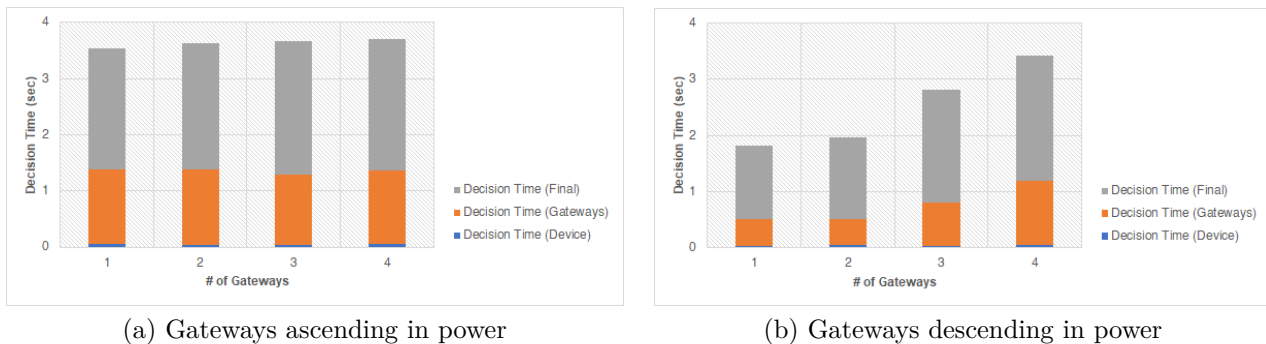


Figure 6.11: Effect of Generating Tasks Simultaneously

computing network. Furthermore, extreme slowdowns can be entirely avoided by introducing a timeout to the auctions.



(a) Gateways ascending in power

(b) Gateways descending in power

Figure 6.12: Effect of Additional Gateway Participants to Decision Times

### 6.3.4 Prediction Mechanism Evaluation

Lastly, figure 6.13 shows the accuracy of the execution time prediction module throughout the different workload scenarios. The module produces satisfactory results within acceptable error margins, and the few outliers are attributed to the innate random nature of the real-life applications employed (in contrast to synthetic or simulated workloads).

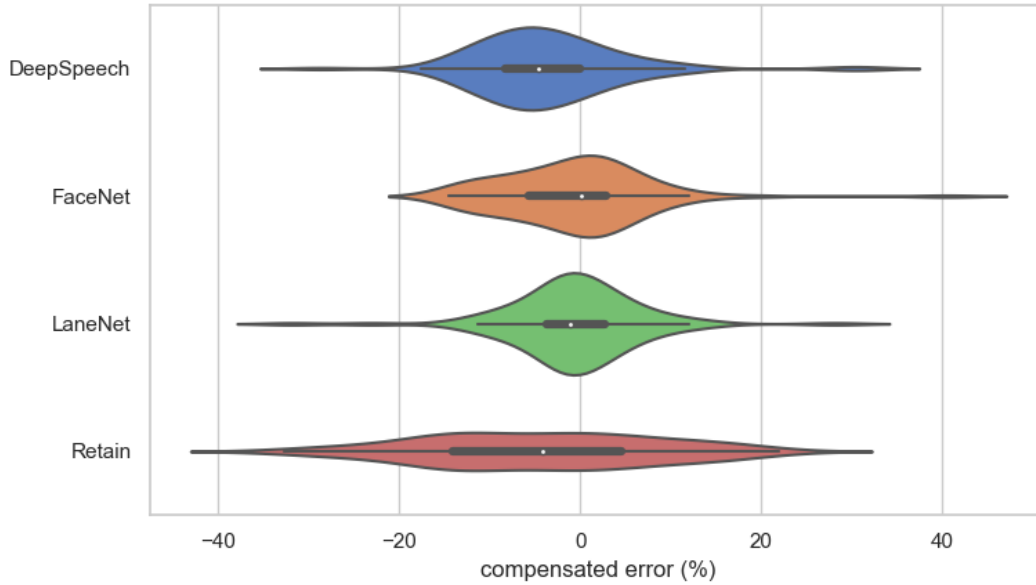


Figure 6.13: Prediction Module Error Rate

## 6.4 Aggregate Evaluation

Having submitted SGRM to a variety of different workload scenarios and compared it to a number of baseline resource management algorithms, as described in the previous sections, we are able to confidently state that the algorithm offers the following advantages:

- **Optimality:** SGRM outperforms the inflexible decision-making of the naive offloading algorithms, while also managing to produce results comparable to those of the optimal decision-making scenario.
- **Distributed nature:** Every device participating in the algorithm communicates with the available gateways autonomously.
- **Scalability:** SGRM scales extremely well due to its distributed nature. The addition of more devices or gateways does not produce extra overheads, and the only potential restricting factor is a limited network bandwidth.
- **Overhead:** SGRM produces a small time overhead, which can be mitigated further by optimizing the algorithmic code or upgrading the network connection between the devices and gateways.
- **Adaptability:** Devices participate in the network dynamically, entering and leaving at will. Also, the gateway and device roles are not mutually exclusive, and a device can choose to act as a gateway at its own accord.



# Chapter 7

## Conclusions

### 7.1 Thesis Summary

Resource Management in edge computing and IoT systems presents a demanding problem, with numerous intricacies to be taken into account and challenges to be overcome. In this thesis, we designated one such problem and addressed it by proposing SGRM: a novel task offloading algorithm for real edge computing systems under processing, memory, bandwidth and deadline constraints.

Our proposal, based on principles of game and auction theory, constitutes an effective distributed solution for allocating resources and offloading tasks on the “edge” of a network, and offers high degrees of adaptability and scalability to any system that utilizes it. At the same time, the algorithm introduces a small (albeit non-negligible) temporal and computational overhead to the system, which we quantify and offer solutions for further minimization.

The extensive experimental evaluation showcases the validity of our approach and exhibits the competence of our algorithm in tackling a diverse mix of workload scenarios. Through comparison with algorithmic baselines designed specifically for this purpose, the SGRM algorithm proves to be a valuable asset for an edge computing network operator that wishes to allocate a set of tasks optimally between devices and minimize the dependence on cloud servers or similar off-site solutions.

### 7.2 Future Work

While the concept of edge computing is not in any way new, with edge implementations dating as far back as the 90’s, its recent surge in popularity has rekindled the interest of researchers, and the widespread adoption of the edge architecture by industry leaders has uncovered numerous opportunities for study and research.

On the crucial matter of resource management in edge computing systems, our own implementation tackles a specific rendition of the task allocation prob-

lem by formulating it as a deadline miss minimization problem with processing, memory and bandwidth constraints and utilizing a game theoretic approach to solve it. The heterogeneity of edge computing networks presents numerous possible variations of the task allocation problem in terms of differentiated system objectives and constraints to be studied. Furthermore, new approaches to the resource management challenge can be developed, utilizing elements from mathematical theories and interdisciplinary fields of science (including market, game and auction theory).

Lastly, the employment of an edge computing solution instead of cloud or similar solutions reintroduces challenges that need to be addressed and accounted for. Such examples include the safeguarding of data privacy and security, as well as the certitude of reliability and redundancy.



# Bibliography

- [1] M. Capra, R. Peloso, G. Masera, M. Ruo Roch, and M. Martina, “Edge computing: A survey on the hardware requirements in the internet of things world,” *Future Internet*, vol. 11, p. 100, 04 2019.
- [2] Alibaba Cloud. What Is Edge Computing? [Online]. Available: <https://www.alibabacloud.com/knowledge/what-is-edge-computing>
- [3] Z. H. et al, *Game theory in wireless and communication networks : theory, models, and applications*, 1st ed. Cambridge University Press, 2012.
- [4] N. Samian, Z. A. Zukarnain, W. K. Seah, A. Abdullah, and Z. M. Hanapi, “Cooperation stimulation mechanisms for wireless multihop networks: A survey,” *Journal of Network and Computer Applications*, vol. 54, pp. 88–106, 2015. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1084804515000855>
- [5] A. Karteris. (2021) Edgebench: A workbench of heterogeneous AI applications. [Online]. Available: <https://github.com/UphillD/edgebench>
- [6] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, “Deep speech: Scaling up end-to-end speech recognition,” 2014.
- [7] mozilla. Project DeepSpeech. [Online]. Available: <https://github.com/mozilla/deepspeech>
- [8] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015. [Online]. Available: <http://dx.doi.org/10.1109/CVPR.2015.7298682>
- [9] davidsandberg. Face Recognition using Tensorflow. [Online]. Available: <https://github.com/davidsandberg/facenet>
- [10] D. Neven, B. D. Brabandere, S. Georgoulis, M. Proesmans, and L. V. Gool, “Towards end-to-end lane detection: an instance segmentation approach,” 2018.

- [11] MaybeShewill-CV. LaneNet Lane Detection. [Online]. Available: <https://github.com/MaybeShewill-CV/lanenet-lane-detection>
- [12] E. Choi, M. T. Bahadori, J. Sun, J. Kulas, A. Schuetz, and W. Stewart, “Retain: An interpretable predictive model for healthcare using reverse time attention mechanism,” in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: <https://proceedings.neurips.cc/paper/2016/file/231141b34c82aa95e48810a9d1b33a79-Paper.pdf>
- [13] mp2893. RETAIN. [Online]. Available: <https://github.com/mp2893/retain>
- [14] Theano Development Team, “Theano: A Python framework for fast computation of mathematical expressions,” *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: <http://arxiv.org/abs/1605.02688>
- [15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from [tensorflow.org](https://www.tensorflow.org/). [Online]. Available: <https://www.tensorflow.org/>
- [16] A. S. Gillis. (2020) What is IoT (Internet of Things) and How Does it Work? [Online]. Available: <https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT>
- [17] D. Y. Zhang and D. Wang, “An integrated top-down and bottom-up task allocation approach in social sensing based edge computing systems,” in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, April 2019, pp. 766–774.
- [18] D. Zhang, Y. Ma, C. Zheng, Y. Zhang, X. S. Hu, and D. Wang, “Cooperative-competitive task allocation in edge computing for delay-sensitive social sensing,” in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, Oct 2018, pp. 243–259.
- [19] D. Liu, L. Khoukhi, and A. Hafid, “Decentralized data offloading for mobile cloud computing based on game theory,” in *2017 Second International*

*Conference on Fog and Mobile Edge Computing (FMEC)*, May 2017, pp. 20–24.

- [20] M.-A. Messous, S.-M. Senouci, H. Sedjelmaci, and S. Cherkaoui, “A game theory based efficient computation offloading in an uav network,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 4964–4974, May 2019.
- [21] M.-A. Messous, H. Sedjelmaci, N. Haouari, and S.-M. Senouci, “Computation offloading game for an uav network in mobile edge computing,” pp. 1–6, 05 2017.
- [22] M. Katsaragakis, D. Masouros, V. Tsoutsouras, F. Samie, L. Bauer, J. Henkel, and D. Soudris, “DMRM: Distributed Market-Based Resource Management of Edge Computing Systems,” in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2019, pp. 1391–1396.
- [23] T. Melissaris, I. Anagnostopoulos, D. Soudris, and D. Reisis, “Agora: Agent and market-based resource management for many-core systems,” in *2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Dec 2016, pp. 400–403.
- [24] M. Yilmaz and R. V. O’Connor, “A market based approach for resolving resource constrained task allocation problems in a software development process,” in *Systems, Software and Services Process Improvement*, D. Winkler, R. V. O’Connor, and R. Messnarz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 25–36.
- [25] Y. Zhang, Y. Xu, Y. Xu, Y. Yang, Y. Luo, Q. Wu, and X. Liu, “A multi-leader one-follower stackelberg game approach for cooperative anti-jamming: No pains, no gains,” *IEEE Communications Letters*, vol. 22, no. 8, pp. 1680–1683, Aug 2018.
- [26] B. Yang, Z. Li, S. Chen, T. Wang, and K. Li, “Stackelberg game approach for energy-aware resource allocation in data centers,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 12, pp. 3646–3658, Dec 2016.
- [27] S. Guo, Y. Dai, S. Guo, X. Qiu, and F. Qi, “Blockchain meets edge computing: Stackelberg game and double auction based task offloading for mobile blockchain,” *IEEE Transactions on Vehicular Technology*, vol. 69, no. 5, pp. 5549–5561, May 2020.
- [28] Y. Fan, Z. Jin, G. Shen, D. Hu, L. Shi, and X. Yuan, “Three-stage stackelberg game based edge computing resource management for mobile

- blockchain,” *Peer-to-Peer Networking and Applications*, vol. 14, pp. 1–15, 05 2021.
- [29] Y. Chen, Z. Li, B. Yang, K. Nai, and K. Li, “A stackelberg game approach to multiple resources allocation and pricing in mobile edge computing,” *Future Generation Computer Systems*, vol. 108, pp. 273–287, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167739X19311653>
- [30] H. Zhang, Y. Xiao, S. Bu, D. Niyato, F. R. Yu, and Z. Han, “Computing resource allocation in three-tier iot fog networks: A joint optimization approach combining stackelberg game and matching,” *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1204–1215, Oct 2017.
- [31] Y. Liu, C. Xu, Y. Zhan, Z. Liu, J. Guan, and H. Zhang, “Incentive mechanism for computation offloading using edge computing: A stackelberg game approach,” *Computer Networks*, vol. 129, pp. 399–409, 2017, special Issue on 5G Wireless Networks for IoT and Body Sensors. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1389128617301068>
- [32] H. Zhang, M. Bennis, L. A. DaSilva, and Z. Han, “Multi-leader multi-follower stackelberg game among wi-fi, small cell and macrocell networks,” in *2014 IEEE Global Communications Conference*, Dec 2014, pp. 4520–4524.
- [33] R. B. Myerson, *Game Theory: Analysis of Conflict*. Harvard University Press, 1991.
- [34] N. C. Luong, P. Wang, D. Niyato, Y. Wen, and Z. Han, “Resource management in cloud networking using economic analysis and pricing models: A survey,” *IEEE Communications Surveys & Tutorials*, vol. 19, pp. 954–1001, 2017.
- [35] N. C. Luong, D. T. Hoang, P. Wang, D. Niyato, D. I. Kim, and Z. Han, “Data collection and wireless communication in internet of things (iot) using economic analysis and pricing models: A survey,” *IEEE Communications Surveys Tutorials*, vol. 18, no. 4, pp. 2546–2590, Fourthquarter 2016.
- [36] J. Moura and D. Hutchison, “Game theory for multi-access edge computing: Survey, use cases, and future trends,” *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 260–288, Firstquarter 2019.

- [37] D. Ross, “Game Theory,” in *The Stanford Encyclopedia of Philosophy*, Winter 2019 ed., E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2019.
- [38] T. Fujiwara-Greve, *Non-Cooperative Game Theory*, 1st ed., ser. Monographs in Mathematical Economics 1. Springer Japan, 2015.
- [39] H. Gintis, “Classical versus evolutionary game theory,” *Journal of Consciousness Studies*, vol. 7, pp. 300–304, 01 2000.
- [40] J. Nash, “Non-cooperative games,” *Annals of Mathematics*, vol. 54, no. 2, pp. 286–295, 1951.
- [41] K. Ritzberger, *Foundations of Non-Cooperative Game Theory*. Oxford University Press, USA, 2002.
- [42] V. Krishna, *Auction theory*, 1st ed. Academic Press, 2002.
- [43] L. Ausubel and P. Milgrom, “The lovely but lonely vickrey auction,” *Comb. Auct.*, vol. 17, 01 2006.
- [44] R. Trestian, O. Ormond, and G.-M. Muntean, “Reputation-based network selection mechanism using game theory,” *Physical Communication*, vol. 4, pp. 156–171, 09 2011.
- [45] Wikipedia. Vickrey Auction. [Online]. Available: [https://en.wikipedia.org/wiki/Vickrey\\_auction#Proof\\_of\\_dominance\\_of\\_truthful\\_bidding](https://en.wikipedia.org/wiki/Vickrey_auction#Proof_of_dominance_of_truthful_bidding)
- [46] D. Merkel, “Docker: lightweight linux containers for consistent development and deployment,” *Linux journal*, vol. 2014, no. 239, p. 2, 2014.
- [47] “MQTT Version 3.1.1, Edited by Andrew Banks and Rahul Gupta.” Oct. 29, 2014, OASIS Standard. [Online]. Available: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>