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

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Distributed Market-Based Resource Management of Edge Computing Systems and IoT Architectures

Εμμανουήλ Κατσαραγάκης

Επιβλέπων : Δημήτριος Ι. Σούντρης
Αναπληρωτής Καθηγητής ΕΜΠ

Αθήνα
Οκτώβριος 2018

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

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Distributed Market-Based Resource Management of Edge Computing Systems and IoT Architectures

Εμμανουήλ Κατσαραγάκης
Α.Μ. : 03113059

Επιβλέπων : Δημήτριος Ι. Σούντρης
Αναπληρωτής Καθηγητής ΕΜΠ

Τριμελής Επιτροπή Εξέτασης

(Υπογραφή)	(Υπογραφή)	(Υπογραφή)
.....	.....	.....
Δημήτριος Σούντρης Αναπληρωτής Καθηγητής ΕΜΠ	Κιαμάλ Πεκμεστζή Καθηγητής ΕΜΠ	Κωνσταντίνος Σιώζιος Επίκουρος Καθηγητής ΑΠΘ

Ημερομηνία Εξέτασης:
3 Οκτωβρίου 2018

Περίληψη

Η ραγδαία ανάπτυξη του Internet of Things(IoT) έχει επιφέρει σημαντική αύξηση των συσκευών που είναι συνδεδεμένες στο διαδίκτυο. Ο αριθμός αυτός εξακολουθεί να αυξάνεται με ταχύτατους ρυθμούς. Συνέπεια αυτού είναι η παραγωγή τεράστιου όγκου πληροφορίων, ο οποίες, παραδοσιακά, μεταφέρονται και επεξεργάζονται στο Cloud. Ωστόσο, οι υποδομές του Cloud είναι γεωγραφικά κεντρικοποιημένες και απομακρυσμένες από τους χρήστες. Έτσι, εφαρμογές που απαιτούν επεξεργασία σε πραγματικό χρόνο, με χαμηλό latency αδυνατούν να εκτελεστούν επιτυχώς, λόγω μεγάλου χρόνου μεταφοράς των δεδομένων, συμφόρησης του δικτύου και υποβάθμισης της ποιότητας. Επιπλέον, σε εφαρμογές που απαιτείται ιδιωτικότητα των δεδομένων, αυτή δε μπορεί να διασφαλιστεί απόλυτα σε απομακρυσμένες υποδομές. Αυτοί οι λόγοι οδήγησαν στη δημιουργία του Edge Computing, όπου οι απαιτούμενοι υπολογισμοί γίνονται κοντά στις IoT συσκευές. Έτσι οι χρήστες ανεξαρτητοποιούνται από τις υποδομές του Cloud και, ταυτόχρονα, ενισχύεται η προστασία των προσωπικών τους δεδομένων.

Η παρούσα διπλωματική παρουσιάζει μία αρχιτεκτονική συσκευών IoT και Gateways, στην οποία οι εφαρμογές των συσκευών μπορούν να σταλούν προς εκτέλεση στο Gateway. Οι πόροι των συσκευών είναι περιορισμένοι, ενω οι πόροι του Gateway είναι διαμοιραζόμενοι και μπορούν να χρησιμοποιηθούν από όλες τις συσκευές. Επομένως, απαιτείται ένας αποδοτικός μηχανισμός διαχείρισης πόρων, προκειμένου να διαμοιραζόνται αποδοτικά οι πόροι του Gateway και να εκπληρώνονται με επιτυχία οι λειτουργικές απαιτήσεις των IoT συσκευών.

Στα πλαίσια της έρευνας αυτής δημιουργήθηκε το *DMRM*: ένα πλήρως κατανεμημένο σύστημα διαχείρισης πόρων σε ένα δίκτυο συσκευών IoT. Η κεντρική ιδέα του αλγορίθμου είναι βασισμένη σε θεμελιώδη μοντέλα της οικονομικής θεωρίας. Πιο συγκεκριμένα, βασίζεται στο μοντέλο προσφοράς και ζήτησης(supply and demand model), στο consumer perceived value pricing. και στο smart data pricing(SDP).

Αρχικά, γίνεται μία εισαγωγή στο IoT , στο Cloud, στο Fog και στο Edge Computing, ενώ αναλύεται και η συσχέτιση της επιστήμης των υπολογιστών με την οικονομική θεωρία. Στη συνέχεια παρουσιάζεται σχετική δουλειά από άλλες μελέτες, προσεγγίσεις αντίστοιχων προβλημάτων με οικονομικά μοντέλα, ενώ γίνεται ανάλυση και σύγκριση των διαφόρων οικονομικών μοντέλων. Έπειτα, αναλυέται η λύση του προβλήματος διαχείρισης πόρων με εφαρμογή της εξαντλητικής μεθόδου, την εφαρμογή μεθόδου Oracle και με εφαρμογή της μεθόδου Simulated Annealing. Γίνεται εκτενής ανάλυση του *DMRM* όλων των μηχανισμών του αλγορίθμου που υλοποιήθηκε και στη συνέχεια παρουσιάζεται μια μελέτη και αξιολόγηση του αλγορίθμου αυτού, καθώς και σύγκρισή του με άλλες λύσεις. Ο αλγόριθμός αυτός εφαρμόζεται σε πλατφόρμες Raspberry pi 3 Model B, Intel Galileo 1, Tegra X1. Τέλος, συνοψίζονται τα αποτελέσματα και γίνονται προτάσεις για μελλοντική έρευνα.

Λέξεις Κλειδιά— IoT, Cloud, Fog, Cloud, Edge, Οικονομική θεωρία, Προσφορά και ζήτηση, Consumer perceived value pricing, Smart Data Pricing, Πραγματικός χρόνος επεξεργασίας, Διαχείριση πόρων , DMRM

Abstract

The rapid growth of IoT has exploded the number of devices connected to the Internet, a number which keeps increasing in high pace. This has led to an enormous amount of collected data and information, traditionally offloaded to cloud computing infrastructure.

However, cloud servers and datacenters are geographically centralized, situated far from the end devices and as a consequence, real-time and latency-sensitive computation services often endure large round-trip delay, network congestion and service quality degradation. Moreover, in data-sensitive domains such as Healthcare, privacy and confidentiality concerns have been raised owned to the storage of data to third-party infrastructure. These reasons (lower latency and higher privacy) have driven the Edge computing paradigm, where the required computation is pushed to the Edge of the IoT network in order to alleviate the dependency on cloud infrastructure and enhance the privacy of identifiable personal data.

The presented work regards the well-established Edge computing architecture of IoT nodes and Gateways, where a portion of the tasks of the IoT nodes are/can be offloaded to the IoT Gateway. In this setup, resources including available CPU, memory and communication bandwidth on both IoT nodes and Gateways are limited and a portion of them is shared. Thus, an efficient resource management mechanism is required to dynamically allocate the shared Gateway resources and to designate the operating configuration of IoT nodes.

In this diploma thesis, *DMRM* is presented: a fully distributed market-based resource management system on IoT architectures. The basic idea of the algorithm is based on fundamental economic and pricing models. More specifically, Supply and Demand model, Consumer Perceived Value Pricing and Smart Data Pricing are applied in this study.

In the beginning, there exists an introduction on IoT, Cloud, Fog and Edge Computing and the connection between computer science and economic theory. Afterwards, there is related work and similar research presented and basic economic models are analyzed and compared among them. Additionally, there is presented a brute-force, an Oracle prediction and a Simulated Annealing solution to the resource management problem. Next, there is an extended analysis of *DMRM* and its mechanisms and, after that, it is being studied, evaluated and compared with other solutions. The algorithm is applies on Raspberry pi 3 Model B, Intel Galileo 1 and Tegra X1. Finally, the conclusions of this diploma thesis are summarized and ideas for future research are proposed.

Keywords— IoT, Cloud, Fog, Cloud, Edge, Economic Theory, Supply and demand, Consumer perceived value pricing, Smart Data Pricing, Real-time processing, Resource management, DMRM

Ευχαριστίες

Καταρχάς, θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου κ. Δημήτριο Σούντρη, ο οποίος μου έδειξε από τη πρώτη στιγμή εμπιστοσύνη για αυτή τη διπλωματική εργασία. Επιπλέον, θέλω να εκφράσω τις ιδιαίτερες ευχαριστίες μου στους υποψήφιους διδάκτορες και φίλους Δημοσθένη Μασούρο και Βασίλειο Τσούτσουρα για την αμέριστη βοήθεια και συμπαράστασή τους καθ'όλη τη διάρκεια της διπλωματικής μου. Τέλος, θέλω να ευχαριστήσω τους γονείς μου Ιωσήφ και Αθηνά, την αδερφή μου Άντα και τους φίλους μου Άρτεμις, Παναγιώτη και Αχιλλέα για τη στήριξή τους όλα αυτά τα χρόνια.

Acknowledgments

First of all, I would like to warmly thank my supervisor, Prof. Dimitrios Soudris, who trusted me from the beginning for this diploma thesis. Also, I would like to express my thanks to the PhD candidates and friends Dimosthenis Masouros and Vasileios Tsoutsouras for their help throughout my diploma thesis. Last but not least, I would like to thank my parents Iosif and Athina, my sister Ada and my friends Artemis, Panagiotis and Achilleas for their support all these years.

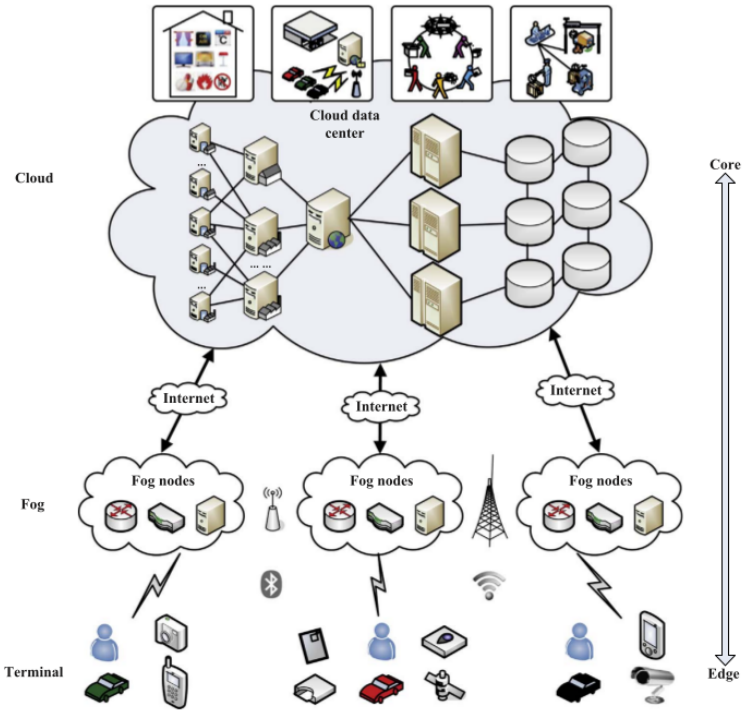
Contents

List of Figures

Εκτεταμένη Περίληψη

Εισαγωγή

Η ραγδαία ανάπτυξη του Internet of Things(IoT) έχει επιφέρει σημαντική αύξηση των συσκευών που είναι συνδεδεμένες στο διαδίκτυο. Ο αριθμός αυτός εξακολουθεί να αυξάνεται με ταχύτατους ρυθμούς. Συνέπεια αυτού είναι η παραγωγή τεράστιου όγκου πληροφορίων, ο οποίες, παραδοσιακά, μεταφέρονται και επεξεργάζονται στο Cloud. Ωστόσο, οι υποδομές του Cloud είναι γεωγραφικά κεντρικοποιημένες και απομακρυσμένες από τους χρήστες. Έτσι, εφαρμογές που απαιτούν επεξεργασία σε πραγματικό χρόνο, με χαμηλό latency αδυνατούν να εκτελεστούν επιτυχώς, λόγω μεγάλου χρόνου μεταφοράς των δεδομένων, συμφόρησης του δικτύου και υποβάθμισης της ποιότητας. Επιπλέον, σε εφαρμογές που απαιτείται ιδιωτικότητα των δεδομένων, αυτή δε μπορεί να διασφαλιστεί απόλυτα σε απομακρυσμένες υποδομές. Αυτοί οι λόγοι οδήγησαν στη δημιουργία του Edge και του Fog Computing, όπου οι απαιτούμενοι υπολογισμοί γίνονται κοντά στις IoT συσκευές. Το σχήμα της αρχιτεκτονικής αυτής παρουσιάζεται στην εικόνα 1. Έτσι οι χρήστες ανεξαρτητοποιούνται από τις υποδομές του Cloud και, ταυτόχρονα, ενισχύεται η προστασία των προσωπικών τους δεδομένων.



Εικόνα 1: Αρχιτεκτονική Fog και Edge Computing [12]

Η παρούσα διπλωματική παρουσιάζει ένα δίκτυο συσκευών IoT και Gateways, στο οποίο οι εφαρμογές των συσκευών μπορούν να σταλούν προς εκτέλεση στο Gateway. Οι πόροι των

συσκευών είναι περιορισμένοι, ενω οι πόροι του Gateway είναι διαμοιραζόμενοι και μπορούν να χρησιμοποιηθούν από όλες τις συσκευές. Επομένως, απαιτείται ένας αποδοτικός μηχανισμός διαχείρισης πόρων, προκειμένου να διαμοιραζόνται αποδοτικά οι πόροι του Gateway και να εκπληρώνονται με επιτυχία οι λειτουργικές απαιτήσεις των IoT συσκευών.

Στα πλαίσια της έρευνας αυτής δημιουργήθηκε το *DMRM*: ένα πλήρως κατανεμημένο σύστημα διαχείρισης πόρων σε ένα δίκτυο συσκευών IoT. Η κεντρική ιδέα του αλγορίθμου είναι βασισμένη σε θεμελιώδη μοντέλα της οικονομικής θεωρίας. Πιο συγκεκριμένα, βασίζεται στο μοντέλο προσφοράς και ζήτησης(supply and demand model), στο consumer perceived value pricing. και στο smart data pricing(SDP).

Διατύπωση του Προβλήματος

Η αρχιτεκτονική του δικτύου συσκευών που εξετάζεται σε αυτή τη διπλωματική εργασία αποτελείται κατά βάση από δύο ειδών συσκευές:

1. Edge Nodes(Gateways) : πρόκειται για οντότητες του Edge Computing, οι οποίες συνεισφέρουν στην ανάπτυξη υπηρεσιών Edge και στη παροχή υπολογιστικών, αποθηκευτικών και δικτυακών πόρων στις συσκευές IoT.

2. Συσκευές IoT : είναι κάθε είδους φυσική συσκευή, οχήματα, οικιακές συσκευές και άλλα αντικείμενα, στα οποία υπάρχουν ενσωματωμένα αισθητήρες, λογισμικό, ενεργοποιητές και σύνδεση στο διαδίκτυο, που τους επιτρέπει να επικοινωνούν και να ανταλλάσουν δεδομένα είτε μεταξύ τους, είτε με το υπόλοιπο διαδίκτυο. Αυτές οι συσκευές έχουν περιορισμένους πόρους σε ό,τι αφορά την cpu, τη μνήμη και τις δικτυακές τους ικανότητες.

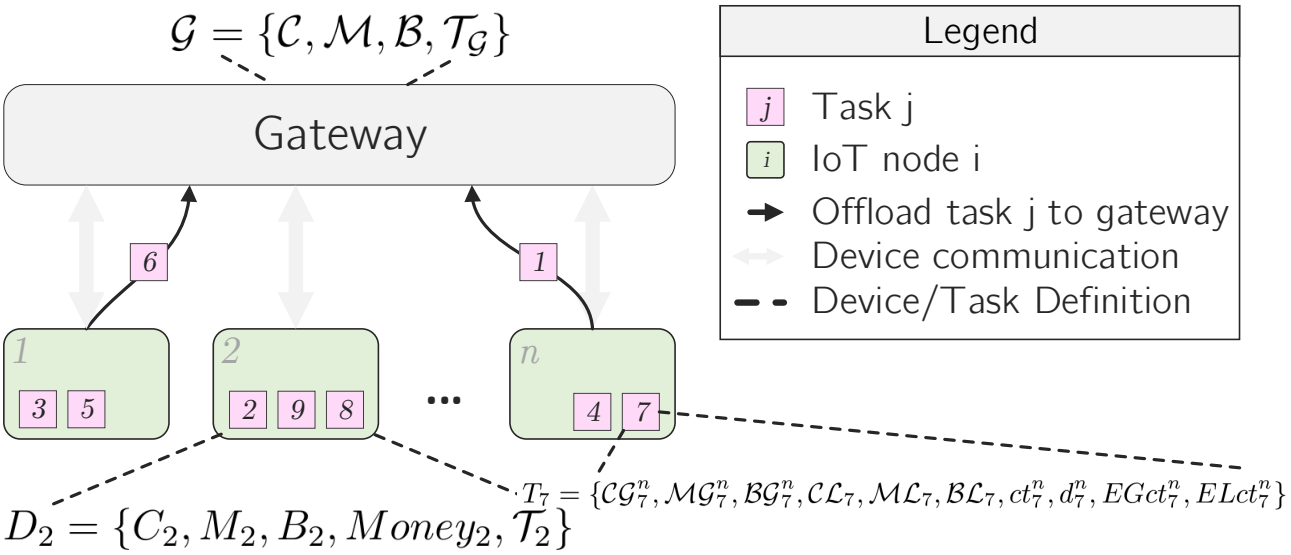
Από τη μία πλευρά, οι συσκευές IoT βρίσκονται κοντά στους καθημερινούς χρήστες και πιο συγκεκριμένα, στο terminal layer της αρχιτεκτονικής που φαίνεται στην εικόνα 1. Από την άλλη πλευρά ο Edge Node βρίσκεται στη διεπαφή του Fog layer και του terminal layer. Επιπροσθέτως, οι συσκευές IoT είναι συνδεδεμένες, είτε ενσύρματα, είτα ασύρματα, με το Gateway κόμβο και μπορούν να επικοινωνούν και να ανταλλάσουν δεδομένα με εκείνον.

Η αρχιτεκτονική του συστήματος που θα μελετήσουμε στα πλαίσια της παρούσας διπλωματικής αποτελείται από ένα σύνολο n IoT συσκευών, μίας συσκευής Gateway και ενός συνόλου εργασιών ανά συσκευή, οι οποίες μπορούν να εκτελεστούν είτε στο Gateway, είτε τοπικά στην εκάστοτε IoT συσκευή, όπως φαίνεται και στο σχήμα της εικόνας 2. Οι συσκευς IoT είναι περιορισμένες σε ό,τι αφορά τους υπολογιστικούς τους πόρους(CPU), τη συνολική χωρητικότητα της μνήμης του και, φυσικά, τις δυνατότητες επικοινωνίας τους με το δίκτυο(bandwidth). Θεωρώντας δεδομένη τη σύνδεση των IoT συσκευών στο δίκτυο, είτε ενσύρματα είτε ασύσματα, κάθε μία από αυτές έχει τη δυνατότητα να επικοινωνεί και να ανταλλάσει δεδομένα με τη συσκευσή του Gateway.

Κάθε IoT συσκευή που βρίσκεται στο εν λόγω δίκτυο ορίζεται μονοσήμαντα από μία τιμή $i \in \{1, ..., n\}$ και χαρακτηρίζεται από την πλειάδα $D_i = \{C_i, M_i, B_i, Money_i, \mathcal{T}_i\}$.

Οι μεταβλητές C_i, M_i, B_i καθορίζουν τη CPU, τη μνήμη και το bandwidth της συσκσυής i αντίστοιχα. Η μεταβλητή $Money_i$ προσδιορίζει τα συνολικά χρήματα που κατέχει η αντίστοιχη συσκευή, ενώ η μεταβλητή $\mathcal{T}_i$ καθορίζει το σύνολο των εργασιών(tasks) τ_j^i της συσκευής i. Για κάθε task απαιτούνται δύο βοηθητικές μεταβλητές οι οποίες προσδιορίζουν την εκτέλεσή του. Πιο συγκεκριμένα, για το task j της συσκευής i, η μεταβλητή o_j^i τίθεται στο 1 αν το συγκεκριμένο tasks εκτελείται στο Gateway, διαφορετικά τίθεται στο 0. Παρομοίως, η μεταβλητή l_j^i τίθεται στο 1 αν το task εκτελείται τοπικά στην IoT συσκευή, αλλιώς τίθεται η τιμή του στο 0.

Στη συσκευή του Gateway, οι IoT συσκευές μπορούν να αποκτήσουν πόρους, προκειμένου να εκτελέσουν τις εργασίες τους. Παρόμοια με τις IoT συσκσυές, ο Edge Node καθορίζεται από μία πλειάδα μεταβλητών $\mathcal{G} = \{\mathcal{C}, \mathcal{M}, \mathcal{B}, \mathcal{T}_\mathcal{G}\}$.



Εικόνα 2: Αρχιτεκτονική του υπό εξέταση συστήματος

Οι μεταβλητές $\mathcal{C}, \mathcal{M}, \mathcal{B}$ προσδιρίζουν τη CPU, τη μνήμη και το bandwidth του Gateway. Το $\mathcal{T}_\mathcal{G}$ είναι το σύνολο των tasks που γίνονται offload στο Gateway κάθε στιγμή. Κάθε task της συσκευής i ορίζεται μονοσήμαντα από το δικό του μοναδικό id j και προσδιορίζεται από την πλειάδα:

$$T_j = \{\mathcal{CG}_j^i, \mathcal{MG}_j^i, \mathcal{BG}_j^i, \mathcal{CL}_j, \mathcal{ML}_j, \mathcal{BL}_j, ct_j^i, d_j^i, EGct_j^i, ELct_j^i\} \quad (1)$$

όπου οι μεταβλητές $\mathcal{CG}_j^i, \mathcal{MG}_j^i, \mathcal{BG}_j^i$ καθορίζουν τους απαιτούμενους πόρους του task j αν αυτό εκτελεστεί στο Gateway και, παρομοίως, οι συσκσυές $\mathcal{CL}_j, \mathcal{ML}_j, \mathcal{BL}_j$ τους αντίστοιχους πόρους που απαιτούνται για να εκτελεστεί το αντίστοιχο task τοπικά. Επιπλέον, η ματαβλητή ct_j^i ορίζει τη χρονική στιγμή στην οποία κάθε task ολοκλήρωσε την εκτέλεσή του, ενώ το d_j^i είναι το χρονικό deadline, μέχρι το οποίο πρέπει να έχει περατωθεί το task. Τέλος, οι μεταβλητές $EGct_j^i$, $ELct_j^i$ δείχνουν τους εκτιμώμενους χρόνους εκτέλεσης του εκάστοτε task της συσκευής j στον Edge Node και στην IoT συσκευή αντίστοιχα.

Οι βασικότερες μεταβλητές του συστήματος που αναλύεται συνοψίζονται στον Πίνακα 1.

Ο βασικός στόχος βελτιστοποίησης του συστήματος όπως εκφράζεται στην εξίσωση 2, είναι να ελαχιστοποιήσουμε το πλήθος των tasks των IoT συσκευών που υπερέβησαν το deadline τους. Οι περιορισμοί των συνολικών πόρων σε ότι αφορά τη CPU, τη μνήμη και το bandwidth στις IoT συσκευές και στο Gateway παρουσιάζονται στις εξισώσεις 4 εως 9.

Η ειδική απαίτηση της μοναδικής εκτέλεσης ενός task(*Unique Execution*) στην εξίσωση 10 σημαίνει ότι κάθε task δε μπορεί να εκτελεστεί και στην IoT συσκευή του και στο Gateway, ενώ

Denotation	**Description**
τ_j^i	task j of device i
o_j^i	1 if task j of device i is executed (offloaded) to Gateway, 0 otherwise
l_j^i	1 if task j of device i is executed locally, 0 otherwise
$\mathcal{C}$, $\mathcal{M}$, $\mathcal{B}$	CPU, Memory and Bandwidth resources of Gateway
C_k, M_k, B_k	CPU, Memory and Bandwidth resources of device k
$\mathcal{CL}_j$, $\mathcal{ML}_j$, $\mathcal{BL}_j$	CPU, Memory and Bandwidth resources required for task j executed locally
$\mathcal{CG}_j^i$, $\mathcal{MG}_j^i$, $\mathcal{BG}_j^i$	CPU, Memory and Bandwidth resources required for task j of device i executed on Gateway
ct_j^i, d_j^i	Completion time and deadline of task j of device i
$EGct_j^i$, $ELct_j^i$	Estimated completion time of task j of device i on Gateway and locally, respectively

Πίνακας 1: Παράμετροι του συστήματος

ταυτόχρονα η αναβολή της εκτέλεσης του task i για μελλοντική εξέταση είναι μία πιθανή επιλογή. Η προτεινόμενη market-based λύση της παρούσας διπλωματικής εργασίες, συνεισφέρει στη λύση του δεδομένου προλήματος βελτιστοποίσης, αποφασίζοντας δυναμικά που θα εκτελεστεί κάθε task.

$$\text{minimize} \quad N_{missed} = \sum_{\forall i}(\sum_{\forall j} miss(\tau_j^i)) \tag{2}$$

$$\text{where} \quad miss(\tau_j^i) = \begin{cases} 0 & if\ ct_j^i \leq d_j^i \\ 1 & otherwise \end{cases} \ \text{subject to:} \tag{3}$$

$$\textit{Gateway} \quad \sum_{\forall i}(\sum_{\forall j} o_j^i \cdot \mathcal{CG}_j^i) \leq \mathcal{C} \qquad \textit{CPU} \tag{4}$$

$$\textit{constraints:} \quad \sum_{\forall i}(\sum_{\forall j} o_j^i \cdot \mathcal{MG}_j^i) \leq \mathcal{M} \qquad \textit{Memory} \tag{5}$$

$$\sum_{\forall i}(\sum_{\forall j} o_j^i \cdot \mathcal{BG}_j^i) \leq \mathcal{B} \qquad \textit{Bandwidth} \tag{6}$$

$$\textit{Device } k \quad \sum_{\forall j} l_j^k \cdot \mathcal{CL}_j \leq C_k \qquad \textit{CPU} \tag{7}$$

$$\textit{constraints:} \quad \sum_{\forall j} l_j^k \cdot \mathcal{ML}_j \leq M_k \qquad \textit{Memory} \tag{8}$$

$$\sum_{\forall j} l_j^k \cdot \mathcal{BL}_j \leq B_k \qquad \textit{Bandwidth} \tag{9}$$

$$\textit{Unique execution} \quad o_j^k + l_j^k \leq 1, \forall j \in \mathcal{T}_k \tag{10}$$

Επιπροσθέτως, αξίζει να αναφερθεί το γεγονός ότι τα tasks μπορούν να κατηγοριοποιηθούν σε τρεις κατηγορίες ανάλογα με τους πόρους που απαιτούν για την εκτέλεσή τους:

- CPU intensive tasks, τα οποία απαιτούν υψηλούς υπολογιστικούς πόρους για να εκτελεστούν επιτυχώς,
- Memory intensive tasks, τα οποία απαιτούν υψηλούς πόρους μνήμης για να εκτελεστούν επιτυχώς και
- Bandwidth intensive tasks, τα οποία απαιτούν υψηλούς πόρους δικτύου για να εκτελεστούν επιτυχώς.

Μέθοδος Brute-Force

Στον κόσμο της επιστήμης των υπολογιστών, η brute-force αναζήτηση, γνωστή και ως εξαντλητική αναζήτηση είναι μία γενική μέθοδος επίλυσης προβλημάτων, σύμφωνα με την οποία εξετάζεται όλος ο χώρος πιθανών λύσεων και, σύμφωνα με τα κριτήρια του εκάστοτε προβλήματος, επιλέγεται η καταλληλότερη λύση. Παρόλο που είναι πολύ απλή στην υλοποίηση και βρίσκει πάντα τη βέλτιστη λύση, το υπολογιστικό και χρονικό της κόστος για την εξέταση όλων των υποψήφιων λύσεων αυξάνει απαγορευτικά καθώς το μέγεθος του προβλήματος μεγαλώνει. Συνεπώς μία τέτοια μέθοδος εφαρμόζεται κυρίως σε προβλήματα, όπου το μέγεθος των λύσεων είναι περιορισμένο, όταν θέλουμε να αποκλείσουμε μία ομάδα λύσεων ή, τέλος, όταν δε μας απασχολεί η ταχύτητα εύρεσης της λύσης.

Εν προκειμένω, στο πρόβλημα της διαχείρισης πόρων της παρούσας διπλωματικής, η εξαντλητική μέθοδος υλοποιήθηκε, προκειμένου για ένα δεδομένο σύνολο συσκευών IoT και ένα δεδομένο σύνολο εφαρμογών να μπορούμε να βρούμε τη βέλτιστη δυνατή λύση ανάμεσα σε όλες τις

Algorithm 1: Brute-force Algorithm

```
Result: Find the minimum number of delayed tasks.
Data: Gateway, DevTuple, Tasks
1  minimum = numberOfTasks + 1;   // in worst case all tasks will be delayed
   Brute-Force(DevTuple,Gateway,Tasks,round):
2    checkForTerminatedTasks(DevTuple,Gateway,Tasks,round);;   // check of
      terminated tasks
3    if all tasks terminated then
4      delayed = countDelayedTasks(); // count the tasks tat exceeded deadline
5      return min(delayed,minimum);
     // find all possible combinations of task scheduling at current round
6    combinations = findAllCombinations(DevTuple,Gateway,Tasks,round);
7    if combinations = NULL & exist undone tasks then
8      minimum = Brute-Force(DevTuple,Gateway,Tasks,round+1);   // recursive
        call
9      return minimum;
10   while combinations!=NULL do
11     insertNextCombination(DevTuple,Gateway,Tasks,combinations);   // on fog or
        IoT device
12     minimum = Brute-Force(DevTuple,Gateway,Tasks,round+1);   // recursive
        call
13   return minimum;
```

δυνατές λύσεις ανά γύρο. Πιο συγκεκριμένα, η λύση αυτή υπολογίζει όλους τους πιθανούς συνδυασμούς, με τους οποίους οι διεργασίες μπορούν να χρονοδρομολογηθούν και να εκτελεστούν και επιστρέφει εκείνη, η οποία ελαχιστοποιεί το πλήθος των εφαρμογών που ξεπέρασαν το χρονικό deadline τους και ταυτόχρονα ελαχιστοποιεί και την συνολική καθυστέρηση που υπάρχει.

Αναλυτικότερα, δεδομένου ενός συνόλου IoT συσκευών *DevTuple*, ενός συνόλου εργασιών *Taks* και του Gateway *Gateway*, θέλουμε να εξετάσουμε όλα τα δυνατά schedulings. Επιπλέον, θεωρούμε ότι ο χρόνος μετράται σε γύρους, R, ξεκινώντας από το 0.Ο αλγόριθμος δουλεύει ως εξής: αρχικά, στο γύρο i αναζητούμε όλους τους δυνατούς συνδυασμούς, με τους οποίους μπορούμε να διαμοιράσουμε τις εφαρμογές που γνωρίζουμε μέχρι εκείνο τον γύρο. Βάζουμε έναν από τους παραπάνω συνδυασμούς προς εκτέλεση και προχωράμε στον επόμενο γύρο επαναλαμβάνοντας την ίδια διαδικασία. Μόλις όλες οι εργασίες ολοκληρωθούν επιστρέφουμε αναδρομικά προς τα πίσω, μέχρι να βάλουμε όλους τους συνδυασμούς. Μετά το πέρας όλων των αναδρομών επιστρέφεται η χρονοδρομολόγηση, η οποία ελαχιστοποιεί το πλήθος των καθυστερημένων εργασιών και, σε δεύτερο χρόνο, που ελαχιστοποιεί τη συνολική καθυστέρηση. Ο αντίστοιχος αλγόριθμος παρουσιάζεται στο 1 σε μορφή ψευδοκώδικα.

Παρόλο που η εξαντλητική μέθοδος επιστρέφει πάντα τη βέλτιστη λύση, το υπολογιστικό και το χρονικό κόστος αυτής καθιστούν την πρακτική εφαρμογή της απαγορευτική μετά από ένα μέγεθος εισόδου. Η προσέγγιση αυτή θα χρησιμοποιηθεί κυρίως για λόγους σύγκρισης των βέλτιστων αποτελεσμάτων.

Μέθοδος Oracle Prediction

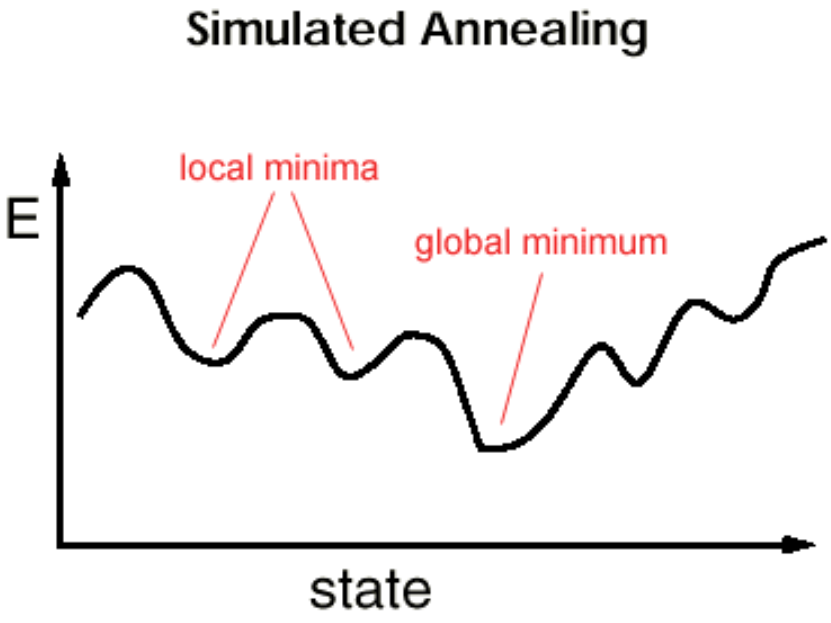
Η μέθοδος αυτή ειναι παρόμοια με τη brute-force λύση. Η βασική διαφορά τους είναι ότι τη προσέγγιση Oracle Prediction γνωρίζουμε εκ των προτέρων όλα τα tasks που θα έρθουν σε όλους τουε γύρους. Συνεπώς η προσέγγιση αυτή μπορεί να οδηγήσει σε ακόμα καλύτερα αποτελέσματα,

καθυστερώντας την εκτέλεση ορισμένων tasks, προκειμένου να κρατήσει πόρους για tasks που θα έρθουν σε επόμενους γύρους με αυστηρότερα deadlines. Παρόλα αυτά και σε αυτή τη προσέγγιση υπάρχουν πολύ υψηλές απαιτήσεις σε ό,τι αφορά τη μνήμη και το χρόνο εκτέλεσης.

Μέθοδος Simulated Annealing

Σύμφωνα με το [6], η μέθοδος Simulated Annealing είναι μία τεχνική βασισμένη στις πιθανότητες, προκειμένου να προσεγγιστεί όσο γίνεται καλύτερα το ολικό βέλτιστο ενός προβλήματος ή μιας συνάρτησης. Πιο συγκεκριμένα, είναι ένας ευριστικός τρόπος να προσεγγίσουμε μία λύση ανάμεσα σε ένα τεράστιο εύρος πιθανών διακριτών λύσεων. Για προβλήματα, στα οποία είναι επιθυμητό να βρούμε μία υποβέλτιστη λύση σε γρήγορο χρόνο, η μέθοδος αυτή συνίσταται.

Ο αλγόριθμος του Simulated Annealing πηγαζει από την επιστήμη της μεταλλειολογίας. Το annealang είναι ουσιαστικά μία μέθοδος κατά την οποία ψύχοντας και θερμαίνοντας ένα υλικό προκειμένου να μεταβάλλουμε τις φυσικές του ιδιότητες, αφού αλλάζουν η εσωτερική δομική του κατάσταση.καθώς το υλικό ψύχεται η νέα του δομή αρχίζει να σταθεροποιείται και, πλέον, το υλικό τείνει να διατηρεί μετέπειτα τις νέες ιδιότητες που απέκτησε. Βασιζόμενοι στα προαναφερθέντα, στη μέθοδο Simulated Annealing θεωρούμε ότι αρχικά η θερμοκρασία του σώματος είναι υψηλή και σταδιακά ψύχεται το υλικό καθώς ο αλγόριθμος προχωράει. Όσο η θερμοκρασία είναι υψηλή, τόσο πιο πιθανό είναι να πλησιάσουμε μία λύση η οποία να γίνει εν τέλει αποδεκτή. Με αυτό τον τρόπο ο αλγόριθμος καταφέρνει να μην παγιδεύεται σε τοπικά βέλτιστες λύσεις. Καθώς η θερμοκρασία μειώνεται, η πιθανότητα να αποδεχτούμε χειρότερες λύσεις, δηλαδή να απομακρυνθούμε από το ολικό βέλτιστο του προβλήματος μειώνονται σημαντικά. Η διαδικασία κατά την οποία το σώμα ψύχεται σταδιακά εξασφαλίζει ουσιαστικά το γεγονός ότι θα προσεγγιστεί η βέλτιστη λύση και θα αποφευχθουν τοπικά βέλτιστες λύσεις.



Εικόνα 3: Simulated Annealing [1]

Παρότι υπάρχουν και άλλες ευριστικές τεχνικές για επίλυση τέτοιου είδους προβλημάτων, όπως είναι η τεχνική hill climbing, υπάρχει μεγάλος κίνδυνος να κολλήσουν σε τοπικά βέλτιστα της λύσης. Αντιθέτως, ο αλγόριθμος simulated annealing αποφεύγει αυτόν τον κίνδυνο και, στη πλειοψηφία των περιπτώσεων, προσεγγίζει την βέλτιστη λύση. Αυτό είναι το βασικό πλεονέκτημα της μεθίδου αυτής έναντι των υπολοίπων.

Για την εκτέλεση του αλγορίθμου, βρίσκουμε μία αρχική κατάσταση s, η οποία είναι ουσιαστικά μία αρχική, τυχαία, λύση του προβλήματος. Στη συνέχεια, μεταβαίνουμε σε μία γειτονική κατάσταση s* και πιθανοτικά αποφασίζουμε αν θα αποδεχτούμε τη νέα κατάσταση ή θα παραμείνουμε στην s. Ειδικότερα, προκειμένου να αποδεχτούμε μία γειτονική κατάσταση ως νέα λύση υπάρχουν δύο βασικά κριτήρια. Το πρωτό κριτήριο είναι αν η νέα κατάσταση είναι καλύτερη της προηγούμενης. Τότε την αποδεχόμαστε. Σε διαφορετική περίπτωση, πρέπει να εξετάσουμε πόσο χειρότερη είναι η νέα λύση και πόσο υψηλή θερμοκρασία έχει το σύστημα. Αυτή η επιλογή γίνεται με βάση την παρακάτω πιθανότητα:

$$exp(\frac{solutionEnergy - neighbourEnergy}{temperature}) \tag{11}$$

Εύκολα καταλαβαίνουμε ότι όσο μικρότερη είναι η διαφορά στην ενέργεια, δηλαδή η διαφορά στη ποιότητα των δύο λύσεων, και όσο μεγαλύτερη είναι η θερμοκρασία, τόσο πιθανότερο είναι να γίνει αποδεκτή η νέα λύση. Όπως προαναφέρθηκε, ο αλγόριθμος έιναι πολύ πιθανό να δεχτεί φαινομενικά χειρότερες λύσεις όταν βρίσκεται σε υψηλές θερμοκρασίες. Ο αλγόριθμος μπορεί να αναλυθεί σε βήματα ως εξής:

- Αρχικοποίηση θερμοκρασίας και αρχικής κατάστασης-λύσης.
- Ξεκινάει η επαναληπτική διαδικασία του αλγορίθμου μέχρι να ικανοποιηθεί η συνθήκη τερματισμού. Συνήθως είναι όταν η θερμοκρασία του συστήματος γίνει μικρότερη από κάποια προκαθορισμένη τιμή ή όταν φτάσουμε σε μία αρκετά ικανοποιητική λύση.
- Σε κάθε γύρο επιλέγουμε μία γειτονική λύση της ήδη υπάρχουσας.
- Αποφασίζουμε αν θα αποδεχτούμε ή όχι τη νέα λύση.
- Μειώνουμε τη θερμοκρασία και συνεχίζουμε την επανάληψη.

Σε ό,τι αφορά τις τιμές της θερμοκρασίας, αυτή μπορεί να καθορίσει σε σημαντικό βαθμό τόσο τη ποιότητα της λύσης, αλλά και τη ταχύτητα με την οποία θα φτάσουμε σε αυτή. Ιδανικά, στις αρχικές καταστάσεις η θερμοκρασία θα πρέπει να είναι τέτοια ώστε ο αλγόριθμος να έχει ευελιξία να κινηθεί σε πληθώρα γειτονικών καταστάσεων.

Επιπροσθέτως, για την υλοποίηση της μεθόδου simulated annealing είναι αναγκαίο να καθοριστούν η συνάρτηση ενέργειας E(), η διαδικασία παραγωγής γειτονικών λύσεων, η συνάρτηση αποδοχής, η αρχική θερμοκρασία T και ο παράγοντας μεταβολής της θερμοκρασίας α. Εν προκειμένω η συνέρτηση ενέργειας που δείχνει τη ποιότητα της λύσης είναι το πλήθος των εργασιών που υπερέβησαν το χρονικό όριό τους, ενώ οι υπόλοιπες παράμετροι καθορίζονται στατικά.

Ο αλγόριθμος 2 παρουσιάζει την εξεταζόμανη μέθοδο σε μορφή ψευδοκώδικα παρακάτω.

Η μέθοδος Simulated Annealing ναι μεν εξασφαλίζει ότι δεν θα κολλήσει ο αλγόριθμος σε τοπικά βέλτιστα, ωστόσο δεν εξασφαλίζει το πόσο κοντά στο ολικό βέλτιστο θα φτάσει. Επιπλέον, από τη φύση του ο αλγόριθμος αποτελείται από τυχαιότητα, γεγονός που δε μας επιτρέπει με ασφάλεια να τον αξιοποιήσουμε σε πραγματικά συστήματα. Όπως είναι προφανές, λόγω του γεγονότος ότι η τεχνική αυτή εξετάζει ένα μικρό μέρος του συνόλου των λύσεων, την καθιστά σαφώς γρηγορότερη σε σχέση με την εξαντλητική αναζήτηση. Ωστόσο, σε ένα πραγματικό σύστημα που απαιτεί άμεση απόκριση και real-time αποφάσεις δεν μπορεί να εφαρμοστεί. Τέλος, αξίζει να σημειωθεί ότι για πολύ μεγάλες εισόδους, το γεγονός ότι οι υπολογισμοί γίνονται κεντρικοποιημένα και όχι κατανεμημένα μεταξύ των διαφόρων συσκευών, δυσχεραίνει ακόμη περισσότερο την πρακτική εφαρμογή του.

Algorithm 2: Simulated-Annealing Algorithm

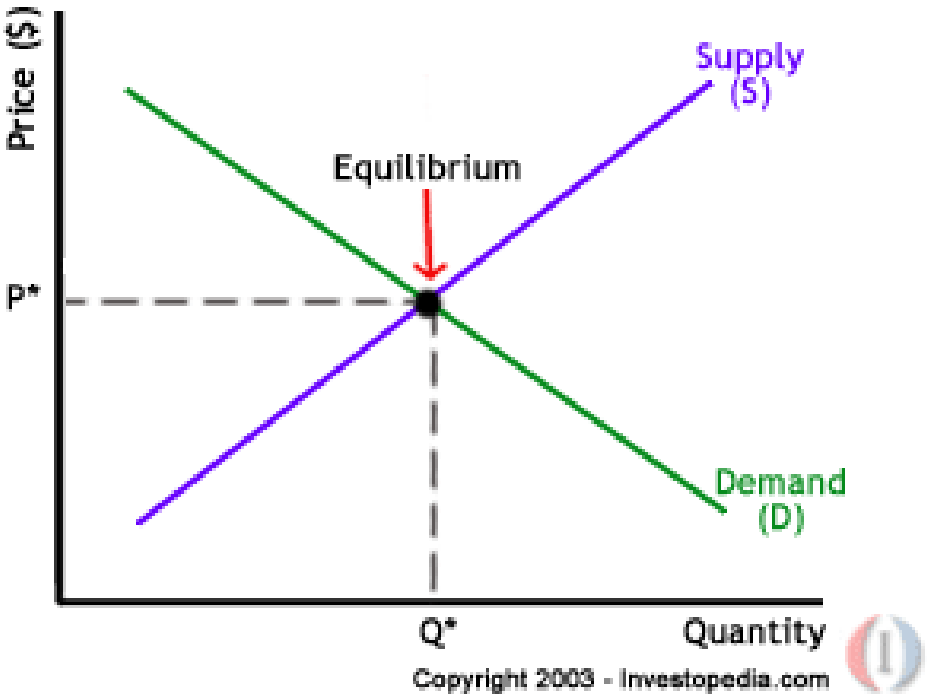
```
Result: Minimuze the number of delayed tasks
Data: Gateway, DevTuple, Tasks
Simulated-Annealing(Gateway, DevTuple, Tasks):
1   initialize(temperature, solution);        // initial temperature and a solution
2   while coolIteration <= maxIterations do
3       coolIteration = coolIteration + 1;
4       tempIteration = 0;
5       while tempIteration <= nrep do
6           tempIteration = tempIteration + 1;
7           newSol = createNewSolution();                  // generate new solution
8           currentEnergy = computeEnergy(newSol);         // energy of new solution
9           d = currentEnergy - previousEnergy;    // compare previous and current
             energy
10          if d < 0 then
11              Accept new solution
12          else
13              Accept new soluiton with probability exp(-d/temperature)
14      T = a * T;                                                        // 0<a<1
```

DMRM: Distributed Market-based Resource Management

Προκειμένου να λύσουμε αποδοτικά και γρήγορα το πρόβλημα της διαχείρισης πόρων της παρούσας διπλωματικής, υλοποιήθηκε μία κατανεμημένη λύση του προβλήματος, η οποία είναι βασισμένη σε θεμελιώδη μοντέλα της οικονομικής θεωρίας. Βασικός σκοπός του αλγορίθμου αυτού είναι να δημιουργήσουμε ένα περιβάλλον αγοράς, στο οποίο οι συσκευές αγοράζουν και πουλάνε πόρους μεταξύ τους, έναντι κάποιου ανταλλάγματος. Πιο συγκεκριμένα οι συσκευές IoT συμπεριφέρονται ως αγοραστές και ζητάνε πόρους προκειμένου να εκτελέσουν τις εργασίες τους. Αντίστοιχα, το Gateway δρα ως πωλητής πόρων και παρέχει τους πόρους του στις συσκευές του δικτύου. Τα πλεονεκτήματα της προσέγγισης αυτής είναι ότι (1) το υπολογιστικό κόστος της λήψης αποφάσεων είναι κατανεμημένο μεταξύ των IoT συσκευών και (2) κάθε κόμβος IoT έχει την ικανότητα να καθορίζει εκείνος τη σπουδαιότητα και τη προτεραιότητα των εργασιών του. Αυτό έχει ως αποτέλεσμα, ο κόμβος του Gateway να έχει γνώση μόνο ενός υποσυνόλου των εργασιών του συστήματος. Έτσι αποφεύγεται συσσωρευμένη λήψη αποφάσεων στο Gateway. Αυτό ενδεχομένως να οδηγήσει σε υποβέλτιστες λύσεις.

Ο αλγόριθμος που υλοποιήθηκε βασίζεται σε 3 βασικά οικονομικά πρότυπα:

- **Μοντέλο Προσφοράς και Ζήτησης(Supply and Demand Model)** : Το μοντέλο προσφοράς και ζήτησης είναι ένα από τα πιο θεμελιώδη μοντέλα της οικονομικής θεωρίας. Η βασική του αρχή είναι ότι σε μία ανταγωνιστική αγορά, η τιμή ενός συγκεκριμένου αγαθού διαφοροποιείται διαρκώς, μέχρι το σημείο όπου η προσφερόμενη ποσότητα του αγαθού αυτού να γίνει ίση με την απαιτούμενη ποσότητα. Το σημείο αυτό ονομάζεται σημείο ισορροπίας της αγοράς(market equilibrium). Η σχέση μεταξύ της προσφοράς και της ζήτησης μιας οντότητας είναι αυτή που καθορίζει τη τελική τιμή του. Από τη στιγμή που θα φτάσουμε σε σημείο ισορροπίας όλοι οι συμμετέχοντες στην αγορά είναι ικανοποιημένοι. Στο σχήμα 4 παρουσιάζεται και γραφικά το σημείο ισορροπίας.



Εικόνα 4: Νόμος προσφοράς και ζήτησης [2]

- **Consumer Perceived Value Pricing** : Πρόκειμένου ο πωλητής ενός αγαθού να αυξήσει τα μακροπρόθεσμα κέρδη του, πολύ συχνά καθορίζει την τιμή του με βάση τις επιθυμίες και τις δυνατότητες των αγοραστών, δηλαδή το ποσό το οποίο οι τελευταίοι ποροτίθενται να πληρώσουν.

- **Smart Data Pricing** : Σύμφωνα με αυτό το οικονομικό μοντέλο, προκειμένου να ρυθμιστεί κατάλληλα η μεγαλή ζήτηση ή η χαμηλή ζήτηση για ένα αγαθό, η τιμή αυτού καθορίζεται με βάση το χρόνο τον οποίο χρησιμοποιείται από έναν αγοραστή, καθώς και επίσης και σε τι βαθμό χρησιμοποιείται από τον αγοραστή. Πιο συγκεκριμένα, όσο αυξάνει ο χρόνος χρήσης του αγαθού και η ποσότητα που χρησιμοποιεί ο αγοραστής, τόσο αυξάνει η τιμή που εκείνος θα κληθεί εν τέλει να πληρώσει. Έτσι από οι πωλητές μπορούν να αποφύγουν φαινόμενα υπερβολικά υψηλής ζήτησης.

Αρχικά, το μοντέλο προσφοράς και ζήτησης χρησιμοποιήθηκε προκειμένου να αποφευχθούν φαινόμενα μεγάλης ανισσοροπίας μεταξύ προσφοράς και ζήτησης των υπολογιστικών πόρων. Επιπλέον, τα Smart Data Pricing και Consumer Perceived Value Pricing συνδυάζονται μεταξύ τους, προκειμένου να καθοριστούν με σαφήνεια οι τιμές πώλησης των πόρων. Οι τελικές αποφάσεις λαμβόνονται στο Gateway, στον οποίο αποστέλλονται όλες οι προσφορές.

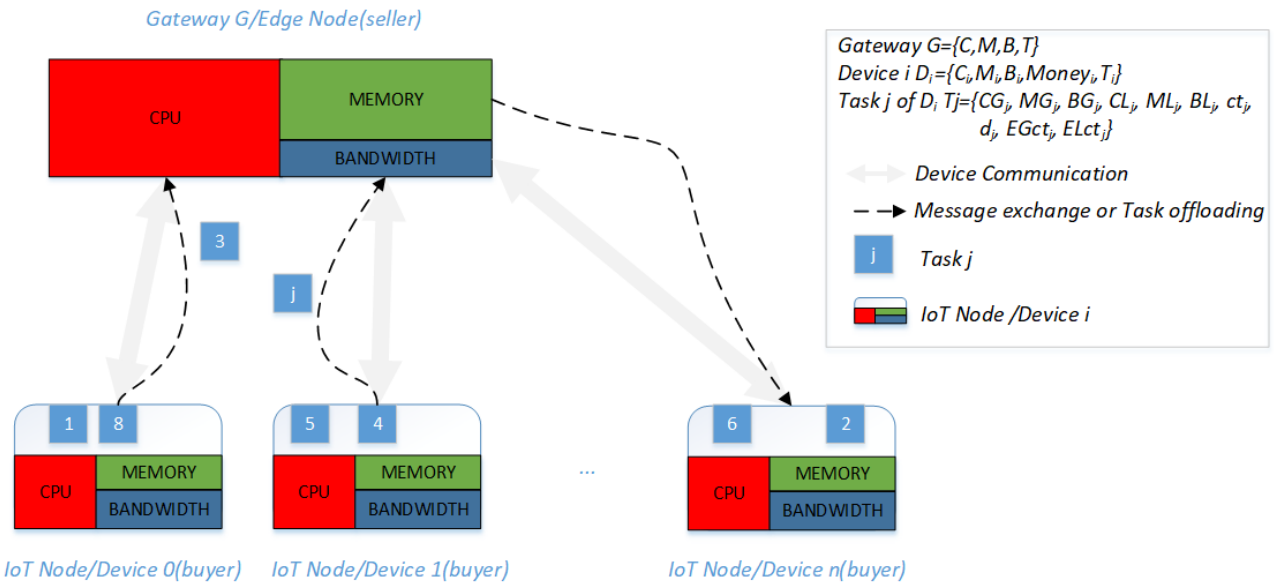
Όπως συμβαίνει σε κάθε αγορά, κάθε οντότητα που συμμετέχει σε αυτή έχει τους δικούς της στόχους. Έτσι και εδώ έχουμε τους εξής στόχους:

- Στόχος IoT συσκευών(αγοραστές) : να ολοκληρωθούν όλες οι εργασίες που επιθυμούν υπό τον περιορισμό των πόρων και του χρόνου.

- Στόχος Fog κόβου(πωλητής) : να μεγιστοποιήσει τα κέρδη του.

- Στόχος Συστήματος : να ελαχιστοποιηθεί το πλήθος των εργασιών που υπερέβησαν το χρονικό deadline τους και να φτάσει η προσφορά και η ζήτηση του συστήματος σε ισορροπία.

Το μοντέλο αγοράς που υλοποιήθηκε παρουσιάζεται στην εικόνα 5.



Εικόνα 5: Αρχιτεκτονική της Αγοράς

Μηχανισμοί του DMRM

Σύμφωνα με τα προαναφερθέντα μοντέλα και τους στόχους που έχουν τεθεί υλοποιήθηκε ένα κατανεμημένο, μη κεντρικοποιημένο σύστημα, το οποίο αποτελείται από τους εξής μηχανισμούς:

1. Money Distribution Mechanism(Μηχανισμός Διαμοιρασμού Χρημάτων)
2. Task Selection Mechanism(Μηχανισμός επιλογής εργασιών)
3. Offloading Mechanism(Μηχανισμός Offloading)
4. Bidding Mechanism(Μηχανισμός αποτίμησης κόστους)

Οι μηχανισμοί *Money Distribution* και *Task Selection* αποτελούν μέρη της λογικής που εκτελείστον στον Edge Node, ενώ οι μηχανισμοί *Offloading* και *Bidding* εκτελούνται σε κάθε IoT κόμβο. Ο συνσδυασμός των μηχανισμών αυτών, σε συνδυασμό με την επιτυχή ανταλλαδή δεδομένων μεταξύ των IoT κόμβων και του Gateway αποτελούν τον DMRM και επιφέρουν τα επιθυμητά αποτελέσματα, δηλαδή τη διαδικασία ορθής λήψης αποφάσεων σε όλες, ανεξαιρέτως, τις συσκευές.

Είναι πολύ σημαντικό να αναφέρουμε το γεγονός ότι η προσέγγιση αυτή έχει ένα σημαντικό πλεονέκτημα. Με τη διαρκή αύξηση και επέκταση του Edge computing αρχίζει να διαφαίνεται ότι θα εφαρμοστεί μία οικονομική πολιτική παρόμοια με αυτή των υπηρεσιών του Cloud. Σύμφωνα με αυτή, οι υπηρεσίες και οι πόροι ενός Gateway θα είναι διαθέσιμες προς πώληση στους χρήστες, ανάλογα με τις ανάγκες και απαιτήσεις του καθενός. Επιπλέον το μοντέλο που παρουσιάζεται θα μπορούσε να προσαρμοστεί σε ένα business μοντέλο, αφού τα θεμέλιά του στηρίζονται σε οικονομικά μοντέλα.

Λήψη αποφάσεων στους IoT κόμβους του DMRM

Η λειτουργία κάθε IoT κόμβου βασίζεται σε δύο εξίσου σημαντικούς μηχανισμούς: το μηχανισμό *Task Offloading* και το μηχανισμό*Bidding*.

Task Offloading Mechanism

Ο μηχανισμός *Task offloading* είναι εκείνος που λαμβάνει την απόφαση για το αν ένα task θα εκτελεστεί τοπικά ή θα σταλεί στο Gateway, έχοντας πάντα ως στόχο να ελαχιστοποιηθεί το πλήθος των tasks που υπερβαίνουν το deadline. Προκειμένου να επιτευχθεί αυτό για κάθε μεμονωμένο task ορίζεται μία μετρική, η οποιά εκτιμάει τη πιθανότητα να υπερβεί το συγκεκριμένο task το deadline του. Η εν λόγω μετρική ονομάζεται **sensitivity (s)** και για το task τ_j^i της συσκευής i ορίζεται ως εξής:

$$s_{\tau_j^i} = \frac{(d_j^i - EGct_j^i - r) + (d_j^i - ELct_j^i - r)}{2} \tag{12}$$

όπου το reίναι η τρέχουσα χρονική στιγμή(σε γύρους) του συστήματος. Όσο χαμηλότερο είναι το sensitivity, τόσο πιθανότερο είναι για το task να υπερβεί το deadline του.

Ανάλογα, λοιπόν με τον υπολογισμό του sensitivity για κάθε task, τα tasks της συσκευής ταξινομούνται σε αύξουσα σειρά σύμφωνα με αυτό. Επομένως, το πρώτο task στη λίστα κάθε συσκευής είναι το πιο ἐπικίνδυνο᾽ να υπερβεί το χρονικό του deadline. Ξεκινώντας από το πρώτο task της λίστας και εξετάζοντας όλα τα tasks, ο IoT κόμβος έχει τρεις διαθέσιμες επιλογές:

- να κάνει offload το task στο Gateway,
- να εκτελέσει τοπικά το task
- να αναβάλει την εκτέλεση του task και να το επενεξετάσει μελλοντικά.

Θεωρούμε, γενικά, ότι τα υπό εξέταση tasks σε κάθε γύρο είναι εκείνα τα οποία δεν έχουν εκτελεστεί ακόμα, ενώ δεν έχουμε καμία γνώση για μελλοντικά tasks. Με άλλα λόγια, οι αποφάσεις σε κάθε γύρο εξαρτώνται αποκλειστικά και μόνο από τη παρελθοντική και παροντική κατάσταση της συσκευής. Σύμφωνα με τα παρακάτω κριτήρια, κάθε συσκευή αποφασίζει αν ένα task θα σταλεί στο Gateway ή όχι.

- Αν η IoT συσκευή δε διαθέτει τους απαιτούμενους πόρους για την εκτέλεση του task, τότε το στέλνει στο Gateway.
- Αν η IoT συσκευή διαθέτει τους απαιτούμενους πόρους για την εκτέλεση του task, αλλά εκτιμάει ότι η εκτέλεσή του στο Gateway θα γίνει γρηγορότερα, τότε το κάνει offload στο Gateway.
- Σε διαφορετική περίπτωση, αν έχει τους πόρους η συσκευή εκτελεί το task, αλλιώς αναβάλει την εκτέλεσή του για το μέλλον.

Για εκείνα τα tasks, τα οποία αποφασίστηκε να σταλούν στο Gateway, ενεργοποιείται ο *Bidding* μηχανισμός, προκειμένου η IoT συσκευή να υποβάλει την αντίστοιχη προσφορά στο Gateway.

Bidding Mechanism

Ο μηχανισμός *Bidding* είναι ένας από τους πιο σημαντικούς παράγοντες για την ορθή και αποδοτική επίδοση της market-based προσέγγισης του προβλήματος. Μέσω αυτής της διαδικασίας καθορίζεται η προσφορά που θα γίνει για κάθε task που είναι πρόθυμη η IoT συσκευή να κάνει. Ο μηχανισμός αυτός είναι σχεδιασμένος με τέτοιο τρόπο, έτσι ώστε να συνυπάρχει αρμονικά με τον *Money distribution* μηχανισμό(ο οποίος παρουσιάζεται παρακάτω), δηλαδή η τιμή πώλησης των πόρων πρέπει να προσαρμόζεται κάθε φορά στις δυνατότητες των συσκευών να πληρώσουν για τους πόρους. Στη πράξη, η τιμή των υπολογιστικών πόρων καθορίζεται με τέτοιο τρόπο, έτσι ώστε η πλειοψηφία των συσκευών να μπορεί να υποστηρίξει την αγορά τους. Με αυτόν τον τρόπο διαβεβαιώνουμε ότι οι τιμές των πόρων και οι προσφορές για αυτούς δε θα έιναι εξαιρετικά

μεγάλες, έτσι ώστε να μπορούν να τις υποστηρίξουν οι συσκευές και, ταυτόχρονα, δε θα είναι ιδιαίτερα χαμηλές, έτσι ώστες να διατηρείται η ανταγωνιστικότητα της αγοράς. Η προσφορά(bid) που θα γίνει για κάθε task υπολογίζεται από την αντίστοιχη IoT συσκευή και η αυτή, μαζί με της απαιτήσεις του task στέλνονται στο Gateway.

Η προσφορά, η οποία γίνεται για κάθε task που γίνεται offload στο Gateway υπολογίζεται σύμφωνα με τους πόρους που απαιτεί το task για να εκτελεστεί στο Gateway, το τρέχον sensitivity του task και, φυσικά, τις οικονομικές δυνατότητες της συσκευής τη τρέχουσα χρονική στιγμή. Πιο συγκεκριμένα, η τιμή της προσφοράς για το task τ_j^i καθορίζεται στην εξίσωση 13.

$$B_{\tau_j^i} = \begin{cases} \mathcal{CG}_j^i + \mathcal{MG}_j^i + \mathcal{BG}_j^i - s_{\tau_j^i} * b, & \text{if } s_{\tau_j^i} \leq 0 \\ \mathcal{CG}_j^i + \mathcal{MG}_j^i + \mathcal{BG}_j^i + \frac{1}{s_{\tau_j^i}} * c, & \text{if } s_{\tau_j^i} > 0 \end{cases} \tag{13}$$

Οι σταθερές b και c ορίζονται στις τιμές 1000 και 800 αντίστοιχα και έχουν καθοριστεί μετά από πολλές προσομοιώσεις του συστήματος. Η εκτιμώμενμη προσφορά, στη συνέχεια, συγκρίνεται με τα εναπομείναντα χρήματα $Money$ της IoT συσκευής. Υπάρχουν δύο πιθανά σενάρια:

1. Αν $B_{\tau_j^i} \leq Money$ τότε το task στέλνεται στο Gateway με προσφορά ίση με την εκτιμώμενη.
2. Διαφορετικά, αν $B_{\tau_j^i} > Money$, η IoT συσκευη δεν έχει τα απαιτούμενα χρήματα για να υποστηρίξει την εκίμηση που έκανε σύμφωνα με την εξίσωση 13. Συνεπώς, το task στέλντεται με προσφορά ίση με τα συνολικά χρήματα $Money$ της συσκευής. Με άλλα λόγια, η συσκευή προσφέρει όλα της τα χρήματα.

Algorithm 3: DMRM functionality on IoT nodes

```
Data: Gateway, IoT device Tuple, Set of Tasks
IoT-Algorithm(Gateway, DevTuple, Tasks, Money):
    volatile curRound /* Round updated at background */
    /* Remaining Tasks */
    RemTasks = checkTasks(Tasks, curRound)
    while length(RemTasks) > 0 do
        /* Invoke Task Offloading Mechanism */
        oTasks = offloading(Tasks, DevTuple, curRound)
        /* Invoke Bidding Mechanism */
        bids = bidding(oTasks, Money)
        /* Send offloading proposition to Gateway */
        sendTasksGateway(oTasks, bids)
        /* Wait for answer */
        answer = waitAnswers(Gateway, offloaded)
        for t in oTasks do
            if answer(t) = "Accept" then
                payGateway(Money) /* Pay Gateway */
            else
                RemoveFromList(t, oTasks)
        RemTasks = checkTasks(Tasks, curRound)
```

Ο αλγόριθμος 3 συνοψίζει τις λειτουργίες του DMRM σε κάθε IoT συσκευη, η οποία κατέχει ένα σύνολο χρημάτων και ένα σύνολο από *Tasks* που πρέπει να εκτελεστούν υπό κάποιους περιορισμούς. Παρουσιάζει την αλληλεπίδραση μεταξύ των μηχανισμών *Task Offloading* και *Bidding* καθώς επίσης και τον τρόπο με τον οποίο οι προσφορές για τα tasks και οι απαντήσεις στέλνονται από και προς το Gateway.

Λήψη αποφάσεων στο Gateway του DMRM

Αντίστοιχα με τις IoT συσκευές, η λειτουργία του Gateway βασίζεται σε δύο εξίσου σημαντικούς μηχανισμούς: το μηχανισμό *Money Distribution* και το μηχανισμό *Task Selection*. Ο πρώτος μηχανισμός είναι υπεύθυνος για τον ορθό διαμοιρασμό των χρημάτων στις ενεργές συσκευές του συστήματος, ενώ ο δεύτερος είναι εκείνος που αποφασίζει τι θα συμβεί με τα ληφθέντα tasks του Gateway.

Money Distribution Mechanism

Ο *Money Distribution* μηχανισμός διασφαλίζει ότι οι IoT συσκευές θα έχουν συνεχώς χρήματα διαμοιράζοντας τακτικά χρήματα σε αυτές με έναν δίκαιο τρόπο, διατηρώντας έτσι την ανταγωνιστικότητα της αγοράς του συστήματος σε υψηλά επίπεδα. Η διαδικασία διαμοιρασμού χρημάτων σε κάθε συσκευή καθορίζεται από τους υπολογιστικούς πόρους(CPU), τους πόρους μνήμης και τους δικτυακούς πόρους(bandwidth) τόσο της ίδιας της συσκευής, όσο και του Gateway. Θεωρώντας ότι το Gateway τη τρέχουσα στιγμή έχει συνολικούς διαθέσιμους πόρους $\mathcal{C}$, $\mathcal{G}$ και $\mathcal{B}$ και ότι υπάρχουν N IoT συσκευές συνδεδεμένες σε αυτό εκείνη τη στιγμή, η συσκευή i θα λέβει επιπλέον χρήματα σύμφωνα με την εξίσωση 14.

$$Money_i = Money_i + \frac{\mathcal{C} + \mathcal{M} + \mathcal{B}}{N} + \frac{a}{C_i} + \frac{a}{M_i} + \frac{a}{B_i} \tag{14}$$

όπου a είναι μια σταθερά ίση με 10000 και έχει προσδιοριστεί έπειτα από πολλές προσομοιώσεις του συστήματος. Ο μηχανισμός αυτός ενεργοποιείται κάθε φορά που το συνολικό πλήθος των χρημάτων όλων των συσκευών που είναι συνδεδεμένες στο Gateway φτάσει σε ένα ελάχιστο κατώφλι(threshold). Αυτό το κατώφλι προσδιορίζεται δυναμικά και εξαρτάται από το πλήθος των συσκευών που είναι συνδεδεμένες στο Gateway. Για N συσκευές το κατώφλι προσδιορίζεται σύμφωνα με την εξίσωση 15.

$$\sum_{i=1}^{N} Money_i \leq 1000 * N \tag{15}$$

Με αυτό τον τρόπο διασφαλίζουμε ότι όλες οι συσκευές στο σύστημα θα έχουν χρήματα ανάλογα με τις ανάγκες τους, με αποτέλεσμα να είναι ικανές να μπορούν να ανταγωνιστούν για τους πόρους του Gateway. Αυτό είναι ιδιαίτερα σημαντικό για την επιτυχή λειτουργία του συστήματος, αφού δεν υπάρχει άλλος τρόπος να λάβουν χρήματα οι συσκευές. Ενδεχομένως, σε ένα πραγματικό σύστημα Edge Computing, ο μηχανισμός αυτός δε θα ήταν απαραίτητος, καθώς κάθε πελάτης θα είχε την υποχρέωση να αποκτήσει τα δικά του λεφτά.

Task Selection Mechanism

Η συσκευη Gateway λαμβάνει όλες τις προσφορές από τις IoT συσκευές και πρέπει να λάβει αποφάσεις σχετικά με το ποια tasks θα γίνουν αποδεκτά, μέσω του μηχανισμού *Task Selection*. Ο μηχανισμός αυτός συνδυάζει την αύξηση των οικονομικών κερδών του Gateway με την ελαχιστοποίηση του πλήθους των tasks που υπερβαίνουν το deadline. Σύμφωνα με την εξίωση 13 γνωρίζουμε ότι όσο μικρότερο είναι το sensitivity, τόσο μεγαλύτερη είναι η προσφορά που γίνεται στο Gateway. Συνεπώς, όσο υψηλότερη είναι η προσφορά για ένα task, τόσο μεγαλύτερο κέρδος θα αποκομίστει ο Edge Node. Ως αποτέλεσμα, αν τα tasks με τις υψηλότερες προσφορές επιλεγούν να εκτελεστούν πρώτα, τότε το σύστημα θα προσεγγίσει το σημείο ισορροπίας μεταξύ της μεγιστοποίησης των κερδών του Gateway και της ελαχιστοποίησης των tasks που υπερέβησαν το deadline τους.

Όταν ο μηχανισμός *Task Selection* ενεργοποιείται, τα εισερχόμενα tasks από όλες τις συσκευές στο Gateway ταξινομούνται σε φθίνουσα σειρά με βάση τις προσφορές τους(bids).

Algorithm 4: DMRM functionality on the Gateway

```
Data: Gateway, IoT devices' Tuples
Gateway-Algorithm(Gateway, NDevs, DevTuples):
    volatile curRound /* Round updated at background */
    totalMoney = 0 /* Initialize Money */
    /* Determine Money re-distribution threshold */
    threshold = determineT(NDevs)
    while ActiveDevs(DevTuples) > 0 do
        if totalMoney < threshold then
            /* Invoke Money Distribution Mechanism */ totalMoney += moneyD(NDevs,
             DevTuples)
        /* Wait for offers */
        taskBids = waitForBids(NDevs, DevTuples)
        /* Invoke Task Selection Mechanism */
        sTasks = taskSelection(taskBids)
        /* Reply to IoT nodes */
        sendAnswers(sTasks, NDevs, DevTuples)
```

Ξενικώντας από το πρώτο task στη λίστα, αν οι διαθέσιμοι πόροι του Gateway επαρκούν για να ικανοποιήσουν τις ανάγκες του task τότε αυτό επιλέγεται προς εκτέλεση. Σε δθαφορετική περίπτωση το task απορρίπτεται. Η διαδικασία αυτή επαναλαμβάνεται για όλα ληφθέντα tasks. Το αποτέλεσμα για κάθε task στέλνεται στην αντίστοιχη IoT συσκευή. Αν το task επιλέχθηκε για εκτέλεση, τότε η IoT συσκευή χρεώνεται ανάλογα με τη προσφορά που είχε κάνει σύμφωνα με τον μηχανισμό *Bidding*. Σε διαφορετική περίπτωση, αν το task απορριφθεί τότε, προφανώς, η IoT συσκευή δε χρεώνεται και μετέπειτα είναι δική της ευθύνη να αποφανθεί για το τι θα συμβεί με το συγκεκριμένο task.

Οι προαναφερθείσες λειτουργίες του DMRM αποτελούν σημαντικό μέρος της λογικής του Gateway, η οποία παρουσιάζεται σε μορφή ψευδοκώδικα στον αλγόριμο 4.

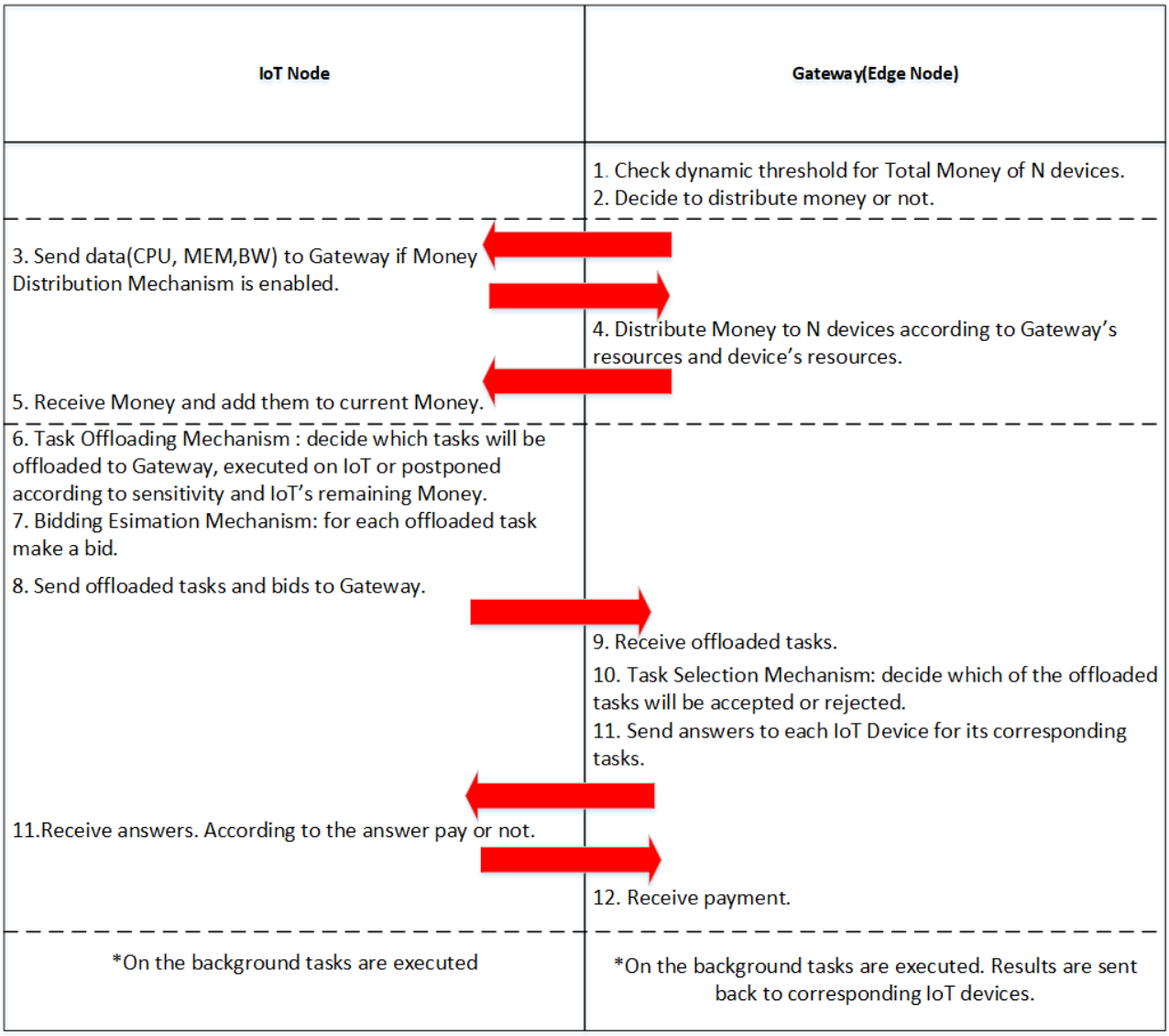
Ο αλγόριθμος αυτός εκτελείται διαρκώς, όσο υπάρχουν συνδεδεμένες ενεργές συσκευές στο Gateway.

Ο μηχανισμός *Money Distribution* εκτελείται σύμφωνα με το πλήθος των συνδεδεμένων συσκευών στο δίκτυο. Ο μηχανισμός *Task Selection* εκτελείται όταν το Gateway λάβει νέες προσφορές από τις IoT συσκευές και στη συνέχει ενημερώνει αντίστοιχα τις συσκευές για τις αποφάσεις του.

Στην εικόνα 6 παρουσιάζεται ο τρόπος με τον οποίο ο αλγόριθμος DMRM εκτελείται με την πάροδο του χρόνο τόσο στις IoT συσκευές, όσο και στο Gateway. Σύμφωνα με τη κατάσταση του δικτύου και τις επιθυμίες των συσκευών, οι διάφοροι μηχανισμοί ενεργοποιούνται και λαμβάνονται οι κατάλληλες αποφάσεις. Σε αυτό το σημείο, είναι πολύ σημαντικό να αναφερθεί το γεγονός ότι σε κάθε συσκευή, είτε Gateway, είτε IoT Node, οι εργασίες εκτελούνται στο background. Αυτό σημαίνει ότι την ίδια στιγμή οι συσκευές μπορούν να πάρουν αποφάσεις.

Πειραματικά Αποτελέσματα

Όλες οι προαναφερθείσες τεχνικές υλοποιήθηκαν στη γλώσσα προγραμματισμού C και εκτελέστηκαν σε πληθώρα ενσωματωμένων συσκευών με διαφορετικές υπολογιστικές ικανότητες. Πιο συγκεκριμένα, ο DMRM εφαρμόστηκε στη συσκευή Intel Galileo Gen 1 στα 400MHz και 256MB RAM , στη συσκευή Raspberry pi 3 Model B με 4 Cortex-A53 CPUs στα 1.2GHz, 1GB RAM και, τέλος στη συσκευή Nvidia Tegra X1 με 4 ARM Cortex A-57 processors στα 1.9GHz και 4GB RAM.



Εικόνα 6: DMRM execution through time

Έισοδος IoT Συσκευών και Tasks

Υποθέτουμε μία ποικιλία διαφορετικών σεναρίων, στα οποία τόσο οι IoT συσκευές, όσο και τα tasks τους επιλέγονται τυχαία. Πιο συγκεκριμένα, οι συνολικοί πόροι των συσκευών και οι απαιτούμενοι πόροι των tasks λαμβάνονται τυχαία από κατανομές Normal και Poisson με εύρος $[0-100]$. Με αυτόν τον τρόπο μπορούμε να δημιουργήσουμε διαφορετικά σενάρια σχετικά με CPU intensive, memory intensive και Bandwidth intensive tasksκαι συσκευές.

Ένας ιδιαίτερα σημανρικός παράγοντας είναι οι χρόνοι άφιξης και η πυκνότητα των tasks στις IoT συσκευές. Η προσοχή σε αυτή την έρευνα επικεντρώνεται σε δύο διακριτές περιπτώσεις. Στη πρώτη περίπτωση χρησιμοποιούμε κανονική κατανομή, προκειμένου να δημιουργήσουμε ταυτόχρονη άφιξη εργασιών και να επιτύχουμε έτσι υψηλή ζήτηση πόρων. Στο δεύτερο σενάριο λαμβάνουμε τους χρόνους άφιξης σύμφωνα με τη κατανομή Poisson, προκειμένου να αποφύγουμε μεγάλες ταυτόχρονες αφίξεις εργασιών. Τέλος, θεωρώντας το χρόνο άφιξης του task τ_j^i ως Ar_j^i, καθορίζουμε το deadline σύμφωνα με τη συνάρτηση 16.

$$d_j^i = C_a * \frac{(EGct_j^i + ELct_j^i + 2 \cdot Ar_j^i)}{2} \tag{16}$$

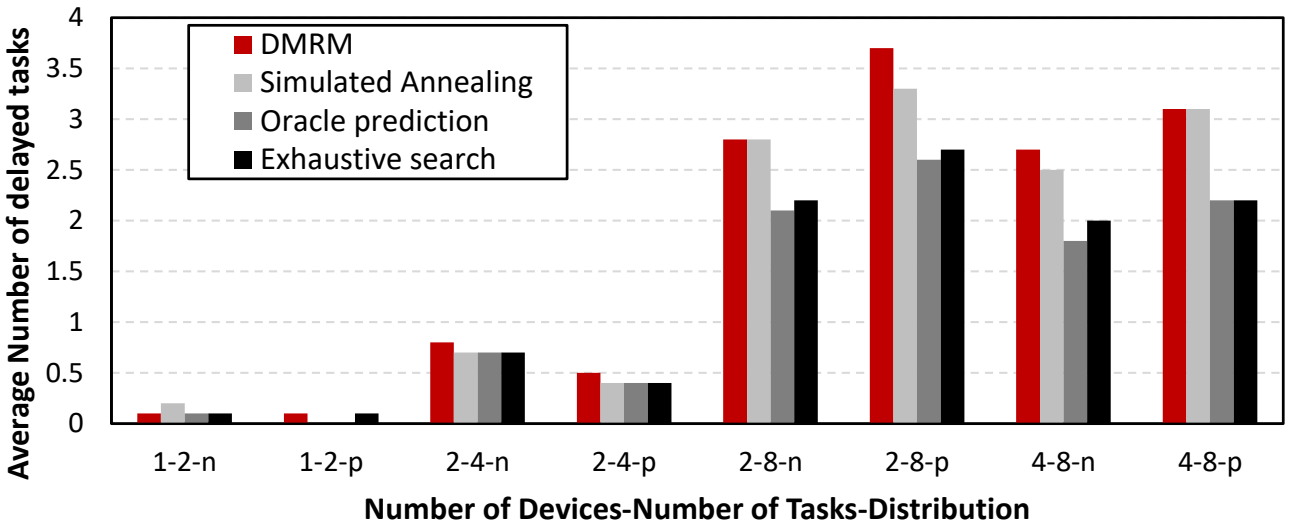
Σύμφωνα με αυτή τη συνάρτηση, το deadline ενός task εξαρτάται άμεσα από το τη χρονική στιγμή της άφιξής του και τον εκτιμώμενο χρόνο εκτέλεσής του στην IoT συσκευή και στο

Gateway αντίστοιχα. Το deadline εξαρτάται από τη τιμή της μεταβλητής C_a, η οποία για την εξαγωγή των πειραματικών αποτελεσμάτων τέθηκε στη τιμή 1.4.

Οι IoT συσκευές και τα tasks παράχθηκαν χρησιμοποιώντας τη γλώσσα προγραμματισμού Python.

Συγκριτική Μελέτη

Σε αυτή την ενότητα αποτιμάται και συκρίνεται η μέθοδος DMRM με τη μέθοδο Simulated Annealing, η οποία όπως προαναφέρθηκε είναι μια τεχνική λύσης προβλημάτων βελτιστοποίησης βασισμένη στη θεωρία πιθανοτήτων και έχει ήδη εφαρμοστεί σε προβλήματα σχετικά με task scheduling [13, 14] και application mapping [19]. Επιπλέον, υλοποιήθηκε και εφαρμόστηκε μία εξαντλητική brute-force προσέγγιση του προβλήματος, προκειμένου να μπορούμε να συγκρίνουμε τη ποιότητα των λύσεων με τις βέλτιστες λύσεις. Ωστόσο, η προσέγγιση αυτή εφαρμόζεται μόνο όταν φτάνουν νέες εργασίες. Για αυτό το λόγο, υλοποιήθηκε και ο μηχανισμός Oracle prediction, ο οποίος γνωρίζει a priori όλα τα tasks πουα θα έρθουν στο σύστημα. Συνεπώς, μπορούμε να βελτιώσουμε και άλλο τα βέλτιστα αποτελέσματα, καθυστερώντας την εκτέλεση εργασιών, προκειμένου να διατηρήσουμε πόρους για εργασίες που θα έρθουν αργότερα και θα είναι πολύ πιο απαιτητικές σε ότι αφορά το deadline τους.

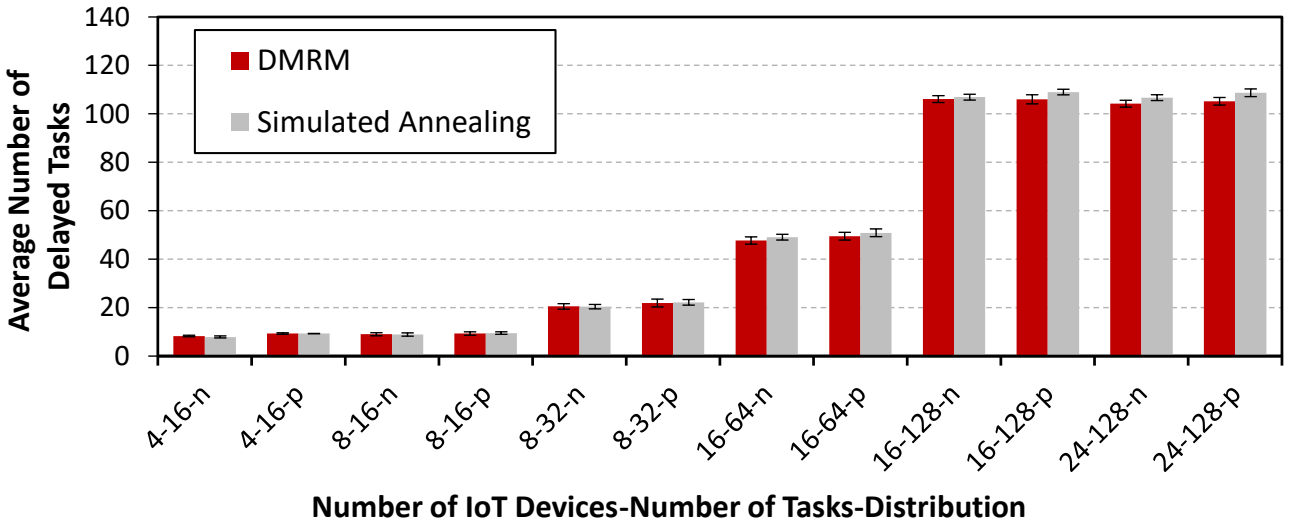


Εικόνα 7: #Tasks με χαμένα deadlines (Περιορισμένη είσοδος)

Στο πρώτο πείραμα σύγκρισης, εκτιμάμε την αποδοτικότητα του DMRM σε σχέση με όλες τις άλλες εναλλακτικές λύσεις που υλοποιήθηκαν. Τα πειράματα αυτά εφαρμόστηκαν για χαμηλό μέγεθος εισόδου, προκειμένου να αποφευχθούν τεράστιοι χρόνοι εκτέλεσης λόγω της πολυπλοκότητας των εξαντλητικών μεθόδων. Τα αποτελέσματα της σύγκρισης συνοψίζονται στην Εικόνα 7, όπου ο άξονας X προσδιρίζει το πλήθος των IoT συσκευών, το συνολικό πλήθος των εργασιών και τη κατανομή πιθανότητας του χρόνου άφιξης των εργασιών, η οποία είναι 'n' για κανονική κατανομή και 'p' για κατανομή Poisson. Ο άξονας Υ αναπαριστά το πλήθος των καθυστερημένων tasks. Σε αυτό το σημείο πρέπει να αναφερθεί ότι η εξαντλητική φύση των brute-force και Oracle prediction εισάγουν τεράστιες απαιτήσεις σε επίπεδο χρόνου και μνημης.Επιπλέον, η μέθοδος Oracle prediction είναι μία ιδανική προσέγγιση, καθώς όλες οι εργασίες θεωρούνται ήδη γνωστές εκτ των προτέρων.

Όπως είναι φανερό, τόσο η προτεινόμενη μέθοδος DMRM, όσο και η μέθοδος Simulated Annealing προσεγγίζουν ικανοποιητικά τα βέλτιστα αποτελέσματα των εξαντλητικών μεθόδων.

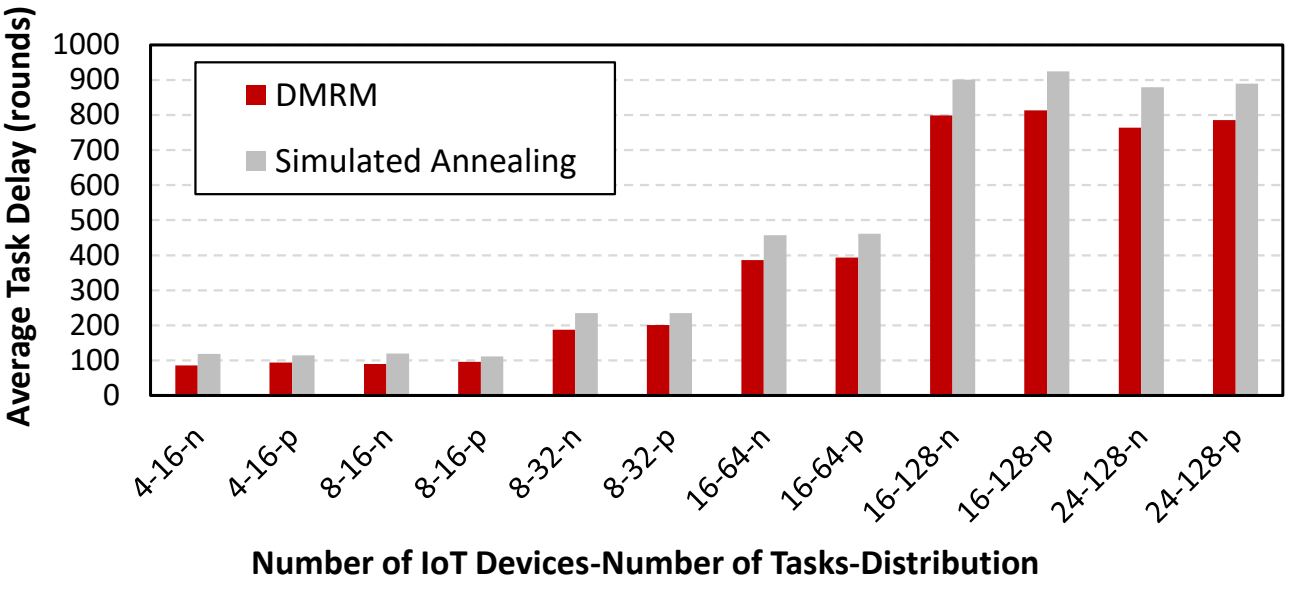
Η πρώτη εκτίμηση δείχνει ότι η προτεινόμενη μεθοδολογία προσεγγίζει τα βέλτιστα αποτελέσματα. Είναι απαραίτητο όμως να εκτελεστούν και πιο απαιτητικά πειράματα με μεγαλύτερο πλήθος



Εικόνα 8: #Tasks with missed deadlines (Demanding input)

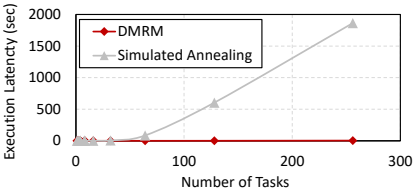
συσκευών και εργασιών. Η εικόνα 8 παρουσιάζει πειράματα για μεγαλύτερες εισόδους, διατηρώντας τα δεδομένα των αξόνων X και Υ ίδια με προηγουμένως. Παρατηρούμε ότι καθώς το μέγεθος εισόδου αυξάνει η μέθοδος DMRM φαίνεται να παρουσιάζει καλύτερα αποτελέσματα σε σχέησ με αυτά της μεθόδου Simulated Annealing. Πιο συρκεκριμένα, παρατηρείται ότι το πλήθος των καθυστερημένων εργασιών που επιστρέφει η προτεινέμενη λύση είναι κατά μέσο όρο 3.22% μικρότερο από αυτό της μεθόδου Simulated Annealing.

Από τη στιγμή που τόσο η μέθοδος DMRM, όσο και η μέθοδος Simulated Annealing εξάγουν σχετικά κοντινά αποτελέσματα, είναι αναγκαίο να εξεταστούν και άλλες παράμετροι. Πιο συγκεκριμένα, στο πείραμα που παρουσιάζεται στην εικόνα 9 φαίνεται η μέση συνολική καθυστέρηση των εργασιών που έχουν υπερβεί το deadline τους, μετρημένη σε γύρους. Τα αποτελέσματα δείχνουν ότι η μέθοδος DMRM υπερτερεί, καθώς ο συνολικός χρόνος καθυστέρησης είναι μικρότερος κατά 12.35% σε σχέση με τη προσέγγιση Simulated Annealing.

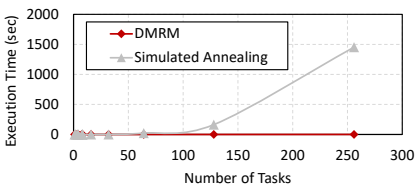


Εικόνα 9: Total rounds of exceeded deadlines (Demanding input)

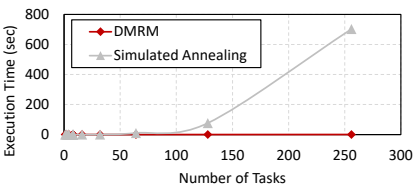
Σε ό,τι αφορά τη μέθοδο Simulated Annealing αξίζει να αναφερθεί το γεγονός ότι η πιθανοτική φύση του αλγορίθμου δεν εξασφαλίζει τη προσέγγιση των βέλτισων λύσεων πάντα. Πιο συγκεκριμένα, ενδέχεται για τις ίδιες παραμέτους συσκευών και εργασιών η μέθοδος αυτή για



(a) Χρόνος εκτέλεσης στο Intel Galileo



(b) Χρόνος εκτέλεσης στο Raspberry Pi 3

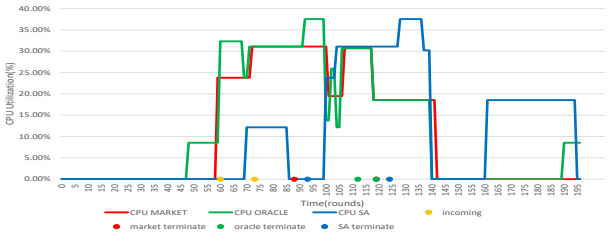


(c) Χρόνος εκτέλεσης στο Nvidia Tegra

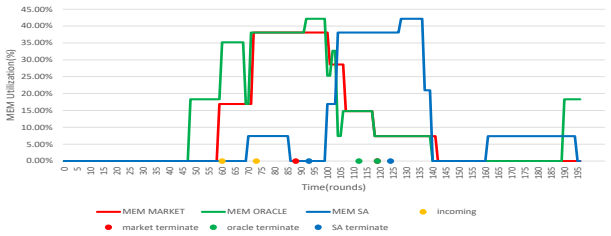
Εικόνα 10: Συγκριτικός χρόνος εκτέλεσης των μεθόδων DMRM και SA σε διαφορετικές Gateway συσκευές

διαδοχικές εκτελέσεις να επιστρέψει διαφορετικά αποτελέσματα. Επιπλέον, όπως φάνηκε και από τα πειράματα, σε περιπτώσεις, στις οποίες υπάρχουν λίγα βέλτιστα schedulings των εργασιών, μειώνεται σημαντικά η πιθανότητα να της προσεγγίσει.

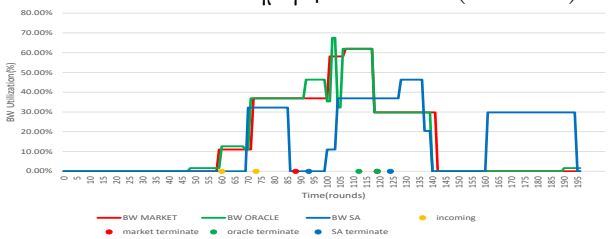
Επιπροσθέτως, ένα επιτυχημένο σύστημα δυναμικής διαχείρισης πόρων καθορίζεται σημαντικά από την ικανοτητά του να λαμβάνει αποφάσεις ταχύτατα. Για αυτό το λόγο, μελετάται το συνολικό execution latency των δύο ευριστικών προσεγγίσεων στις ενσωματωμένες συσκευές Intel Galileo Gen 1, Raspberry Pi 3 Model B και Nvidia Tegra X1.



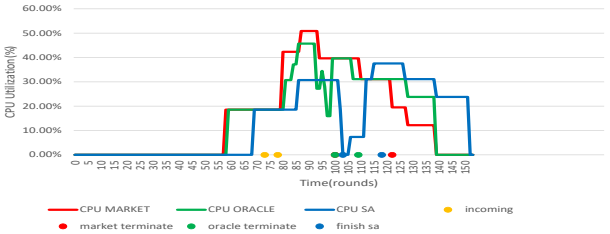
Εικόνα 11: CPU για 2 tasks(Normal)



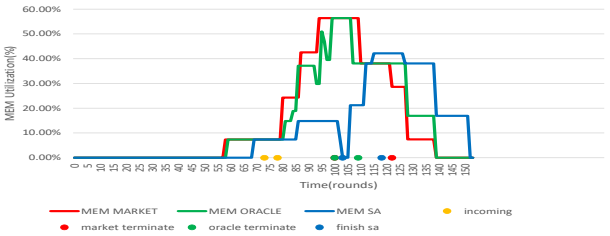
Εικόνα 12: Μνήμη για 2 tasks(Normal)



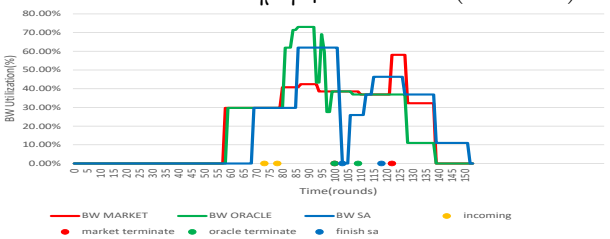
Εικόνα 13: Bandwidth για 2 tasks(Normal)



Εικόνα 14: CPU για 2 tasks(Poisson)



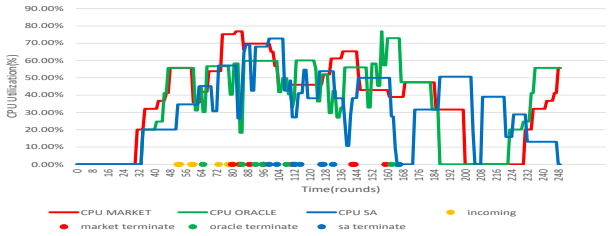
Εικόνα 15: Μνήμη για 2 tasks(Poisson)



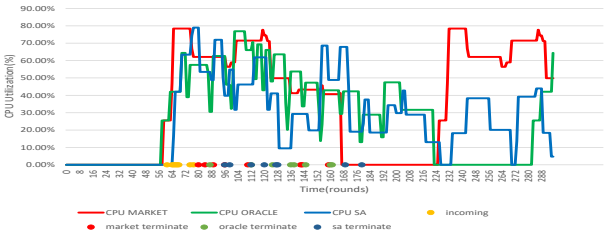
Εικόνα 16: Bandwidth για 2 tasks(Poisson)

Η εικόνα 10 παρουσιάζει τις τιμές latency που μετρήθηκαν, δείχνοντας ότι για χαμηλό πλήθος IoT tasks και οι δύο λύσεις έχουν κοντινά αποτελέσματα. Ωστόσο, καθώς το πλήθος των εργασιών αυξάνει, τότε το execution latency της μεθόδου Simulated Annealing ανεβαίνει σημαντικά, ενώ του DMRM είναι εξαιρετικά μικρή. Αυτό είναι εμφανές, καθώς στη περίπτωη που έχουμε ως είσοδο 256 tasks η μέθοδος DMRM είναι 2000× γρηγοτερη. Αυτή η συμπεριφορά οφείλεται στην ουσιώδη διαφορά των δύο μεθόδων, η οποία είναι ότι η DMRM είναι κατανεμημένη, σε αντίθεση με την SA, η οποία είναι κεντρικοποιημένη. Με αυτό τον τρόπο, το υπολογιστικό κόστος

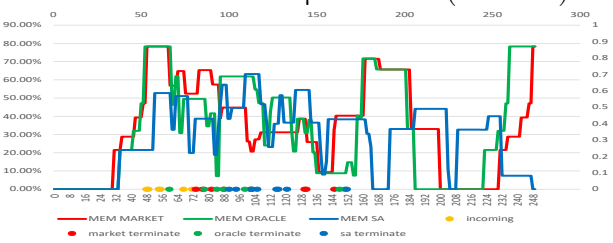
της πρώτης διαμοιράζεται στις διαφορετικές IoT συσκευές του δικτύου.



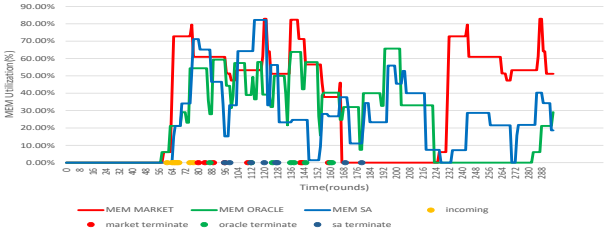
Εικόνα 17: CPU για 8 tasks(Normal)



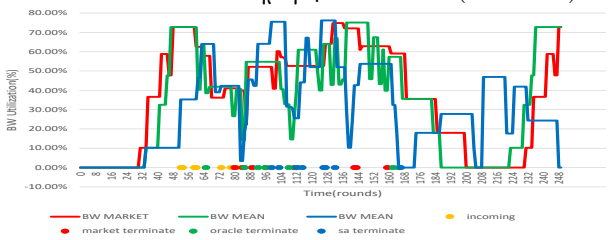
Εικόνα 20: CPU για 8 tasks(Poisson)



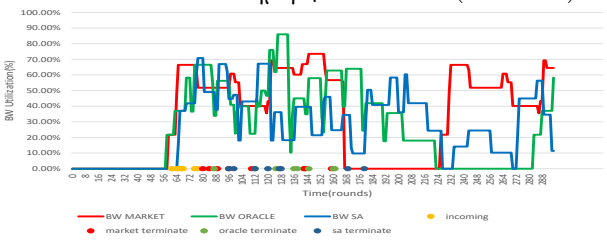
Εικόνα 18: Μνήμη για 8 tasks(Normal)



Εικόνα 21: Μνήμη για 8 tasks(Poisson)



Εικόνα 19: Bandwidth για 8 tasks(Normal)



Εικόνα 22: Bandwidth για 8 tasks(Poisson)

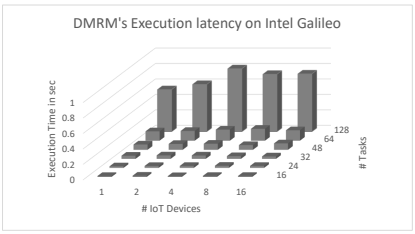
Τέλος, εκτελέστηκαν ορισμένα πειράματα, προκειμένου να εκτιμήσουμε το βαθμό χρησιμοποίησης των πόρων του Gateway και να συγκρίνουμε τον DMRM με τις υπόλοιπες μεθόδους που υλοποιήθηκαν. Πιο συγκεκριμένα, τα γραφήματα των εικόνων 11- 16 και 17- 22 παρουσιάζουν τη μέση χρήση της CPU, της μνήμης και του bandwidth του Gateway, για εισόδους 2 και 8 εφαρμογών αντίστοιχα. Όλα τα πειράματα εκτελέστηκαν τόσο για εισόδους που ακολουθούν κατανομή Poisson, όσο και για εισόδους που ακολουθούν κανονική κατανομή. Ο άξονας X παρουσιάζει το χρόνο, μετρημένο σε γύρους, ενώ ο άξονας Υ αναπαριστά το ποσοστό χρησιμοποίησης του εκάστοτε πόρου. Οι τελείες στον X άξονα υποδεικνύουν τις χρονικές στιγμές στις οποίες έρχονται οι διάφορες εφαρμογές, καθώς και εκείνες για τις οποίες οι εφαρμογές ολοκληρώνονται χρησιμοποιώντας τη μέθοδο DMRM, τη μέθοδο Oracle Prediction και τη μέθοδο Simulated Annealing.

Όπως φαίνεται στις γραφικές αυτές, η χρησιμοποίηση των πόρων φτάνει σχεδόν στον 85% των συνολικών πόρων του Gateway. Αυτό σημαίνει ότι οι IoT συσκευές προσπαθούν να εκμεταλλευτούν πλήρως τους πόρους του Gateway και να κάνουν τα tasks τους offload. Αυτό συνεπάγεται ότι οι ίδιες οι συσκευές εξοικονομούν πόρους, μπαταρία και ενέργεια. Αυτό δίνει στις συσκευές μεγαλύτερη διάρκεια ζωής σε ό,τι αφορά τη μπαταρία τους. Επιπλέον, παρατηρούμε ότι η χρησιμοποίηση των πόρων που κάνει η μέθοδος DMRM είναι πολύ κοντινή με αυτή της μεθόδου Oracle Prediction, ενώ την ίδια στιγμή η προσέγγιση Simulated Annealing φαίνεται ότι χρησιμοποιεί αισθητά λιγότερο τους πόρους του Gateway.

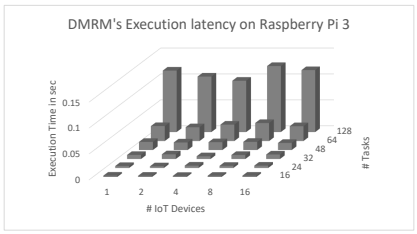
Παρατηρήσεις για το DMRM

Σε αυτή την ενότητα αναδεικνύονται ειδικά χαραστηριστικά της προτεινόμενης λύσης DMRM, προκειμένου να μπορέσουμε να αποφανθούμε για την αποδοτικότητά της.

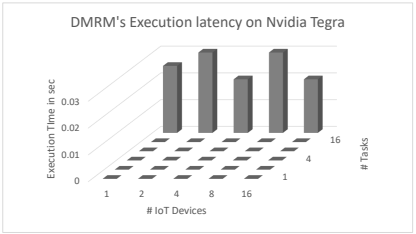
Αρχικά, γίνεται μία παρατήρηση σχετικά με το χρόνο εκτέλεσης του DMRM. Πιο συγκεκριμένα, τα πειράματα που εκτελέστηκαν στις ενσωματωμένες συσκευές έδειξαν ότι ο μέσος χρόνος εκτέλεσης του DMRM αποτελεί κυριώς συνάρτηση του πλήθους των εργασιών και όχι του πλήθους των συνδεδεμένων IoT συσκευών. Η εικόνα 23 παρουσιάζει το μέσο χρόνο εκτέλεσης του DMRM σε σχέση με το πλήθος των IoT συσκευών και των tasks.



(a) Χρόνος εκτέλεσης του DMRM στο Intel Galileo



(b) Χρόνος εκτέλεσης του DMRM στο Raspberry Pi 3



(c) Χρόνος εκτέλεσης του DMRM στο Nvidia Tegra

Εικόνα 23: Ξρόνος εκτέλεσης του DMRM σε διαφορετικές Gateway συσκευές

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσής του εξαρτάται από το πλήθος των εργασιών προς εκτέλεση, ενώ το πλήθος των συνδεδεμένων IoT συσκευών στο Gateway φαίνεται να μην επηρεάζει ιδιαίτερα. Συνεπώς, το DMRM μπορεί να εφαρμοστεί επιτυχώς για μεγάλο πλήθος IoT συσκευών και tasks. Με άλλα λόγια, το DMRM είναι κλιμακώσιμο για μεγάλες εισόδους.

Σε ό,τι αφορά τα μοντέλα της οικονομικής θεωρίας, είναι φανερό ότι τα Smart Data Pricing και Consumer Perceived Value Pricing χρησιμοποιήθηκαν για τον καθορισμό της τιμής των πόρων.

Αποτίμηση του **DMRM**

Τέλος, συνοψίζοντας όλα τα προαναφερθέντα, η DMRM προσέγγιση χαρακτηρίζεται από τα εξής:

- **Optimality:** Το προτεινόμενο σύστημα προσεγγίζει ικανοποιητικά βέλτιστες και υποβάλτιστε λύσεις του προβλήματος τόσο για περιορισμένες, όσο και για απαιτητικές εισόδους. Επιλέον, σε σχέση με άλλες μεθόδους επιφέρει καλύτερα αποτελέσμα σε ό,τι αφορά τη μέση καθυστέρηση ανά εργασία.
- **Κατανεμημένο:** Οι αποφάσεις και το υπολογιστικό κόστος αυτών είναι κατανεμημένο μεταξύ των IoT συσκευών του δικτύου.
- **Κλιμακώσιμο:** το DMRM μπορεί να εφαρμοστεί ικανοποιητικά για μεγάλες εισόδους.
- **Χαμηλό execution latency:** συγκριτικά με άλλες μεθόδους, το DMRM έχει χαμηλές απαιτήσεις σε latency. Συνεπώς, μπορεί να εφαρμοστεί σε συστήματα που απαιτούν λήψηreal-time αποφάσεων και έχουν latency-sensitive εφαρμογές.
- **Προσαρμογή σε δυναμικές αλλαγές:** Η προτεινόμενη λύση μπορεί να προσαρμοστεί σε δυναμικές αλλαγές στις του δικτύου, όπως εισαγωγή ή εξαγωγή συσκευών σε αυτό.

Chapter 1

Introduction

1.1 Internet of Things and Cloud Computing

The Internet of things(IoT) is becoming an increasingly growing topic of conversation both in academic and industrial community. Simply put, IoT is a network of physical devices, vehicles, home appliances and other items embedded with electronics, software, sensors, actuators, and connectivity which enables these things to connect and exchange data. IoT refers to a self configuring, adaptive and complex network, which allows extending internet connectivity beyond standard devices, such as desktops, laptops, smartphones and tablets, to any range of traditionally dumb or non-internet-enabled physical devices and everyday objects. In the majority of the cases, these devices need to interact and cooperate with each other, in order to reach common goals.

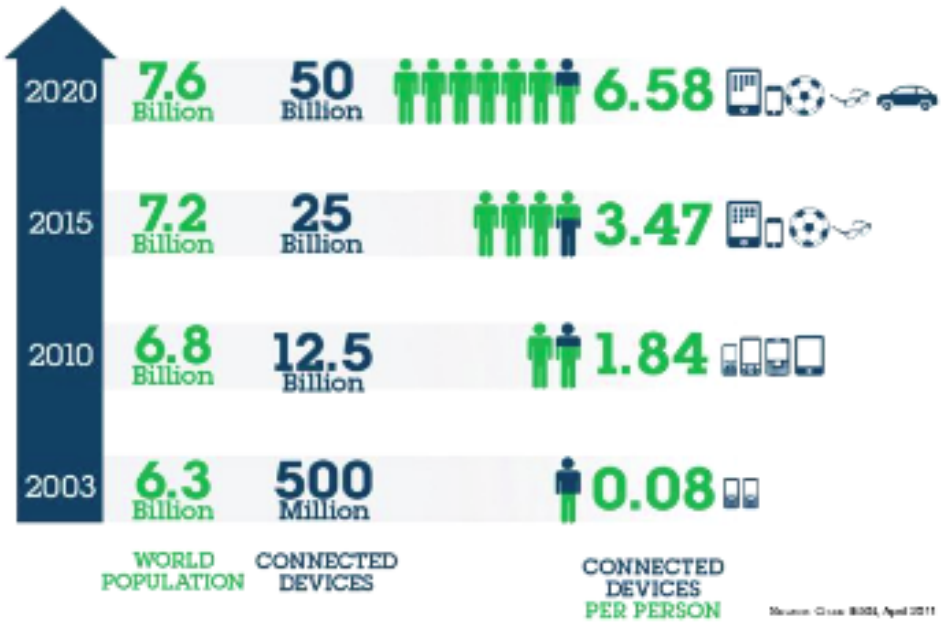


Figure 1.1: IoT growth over the last years

Looking to the future, Cisco predicts there will be approximately 50 billion devices connected to the Internet by 2020. That is a huge amount of data being generated, which need to be processed and analyzed. The current paradigm for the processing of that huge amount of data is uploading, storing and processing using Cloud computing. Cloud computing is an information technology(IT) paradigm that enables ubiquitous access to shared tools of configurable system

resources and higher-level services that can be rapidly provisioned with minimal management effort, often over the Internet.

However, many applications require real-time processing of data and decision making as well as data anonymity, which is difficult to guarantee under the current processing architecture. Cloud datacenters are geographically centralized and situated far from the proximity of the end devices. As a consequence, real-time and latency-sensitive computation services requests to be responded by the distant Cloud datacenters often endure large round-trip delay, network congestion and service quality degradation. For instance, healthcare companies don't want to stream critical data points generated by life-saving systems. That data needs to be processed locally not only for faster turnaround but also for anonymizing personally identifiable patient data. The demand for distributing the IoT workloads between the local data center and cloud has resulted an architectural pattern called Fog computing(aka Edge computing).

1.2 Fog and Edge Computing

1.2.1 Fog Computing

Fog could be described as a middle layer between the IoT devices and the Cloud, which offers compute, networking and storage facilities so that Cloud-based services can be extended closer to the IoT devices/sensors, which aims to enhance low-latency, mobility, network bandwidth, security and privacy. In figure 1.2, the fundamental architecture of Fog Computing is presented.

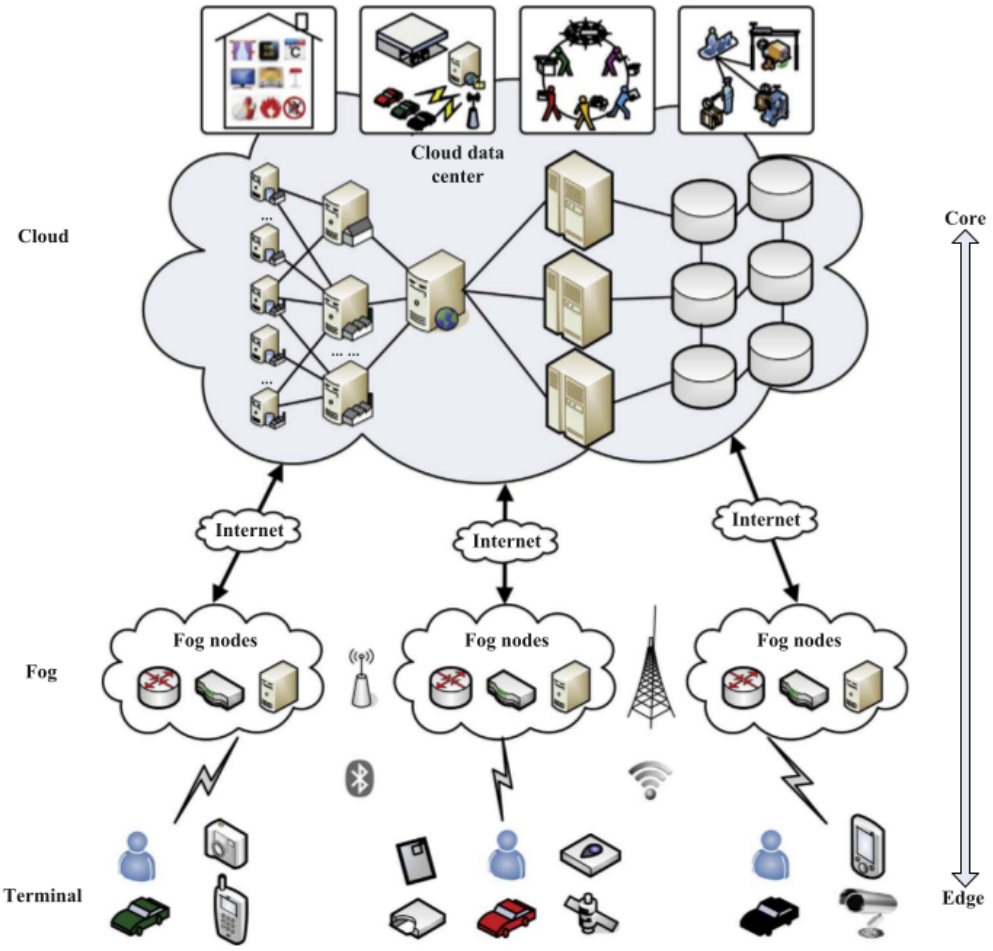


Figure 1.2: The hierarchical architecture of Edge and Fog computing [12]

The hierarchical architecture, as described in [12], is composed of the following three layers:

- Terminal Layer: This is the layer closest to the end user and physical environment. It consists of various IoT devices. These devices are widely geographically distributed in general. They are responsible for sensing the feature data of and transmitting these data to the upper layer for processing and storage.
- Fog Layer: This layer is composed of a large number of fog nodes, which generally include routers, gateways, switchers, access servers, base stations and specific fog servers.These nodes are widely distributed between the end devices and cloud. The end devices can connect with fog nodes to obtain storage and compute services. The real-time analysis and latency-sensitive applications can be accomplished in fog layer. Fog nodes are also responsible for the interaction with Cloud services.
- Cloud Layer: The cloud computing layer consists of multiple high-performance servers and storage devices and provides various application services. It has powerful computing and storage capabilities to support for extensive computation analysis and permanent storage of an enormous amount of data.

1.2.2 Edge Computing

Similar to Fog computing, Edge computing is a method of optimizing applications or Cloud computing systems by taking some portion of an application, its data, or services away from one or more central nodes (the "core") to the other logical extreme (the "edge") of the Internet which makes contact with the physical world or end users [7]. In one vision of this architecture, specifically for IoT devices, data comes in from the physical world via various sensors, and actions are taken to change physical state via various forms of output and actuators; by performing analytics and knowledge generation at the edge, communications bandwidth between systems under control and the central data center is reduced. Edge Computing takes advantage of proximity to the physical items of interest and also exploits the relationships those items may have to each other. Another, more broad way of looking at "Edge Computing" is to put any type of computer program that needs low latency nearer to the requests.

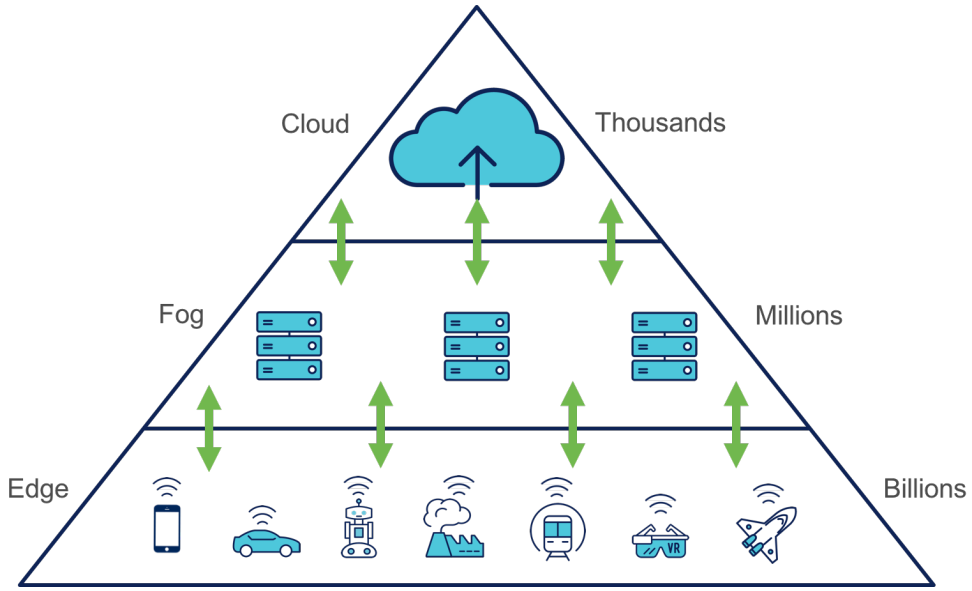


Figure 1.3: Edge Computing Architecture

Edge computing pushes applications, data and computing power (services) away from centralized points to the logical extremes of a network. Edge computing takes advantage of microservices architectures to allow some portion of applications to be moved to the edge of

the network. While Content Delivery Networks have moved fragments of information across distributed networks of servers and data stores, which may spread over a vast area, Edge Computing moves fragments of application logic out to the edge. As a technological paradigm, edge computing may be architecturally organized as peer-to-peer computing, autonomic (self-healing) computing, grid computing, and by other names implying non-centralized availability.

To ensure acceptable performance of widely dispersed distributed services, large organizations typically implement edge computing by deploying server farms with clustering and large scale storage networks. Previously available only to very large corporate and government organizations, edge computing has disseminated technology advances and cost reductions from large-scale implementations and made the technology available to small and medium-sized businesses. Small, low-cost cluster hardware and freely-available cluster management software have increased accessibility.

The target of Edge Computing is any application or general functionality needing to be closer to the source of the action where distributed systems technology interacts with the physical world. Edge Computing does not need contact with any centralized Cloud. Edge Computing does use a similar or the same distributed systems architecture as centralized Clouds but closer to or directly at the Edge.

Edge computing imposes certain limitations on the choices of technology platforms, applications or services, all of which need to be specifically developed or configured for edge computing.

It should be mentioned that each IoT device is connected to gateways wired, or wireless, and each entity in the Fog layer is linked into the Cloud services through the internet.
The architecture of Fog and Edge computing has some special characteristics and several advantages, as listed below:

- Low latency and real-time interactions
- Save bandwidth
- Support from mobility
- Geographical distribution and decentralized data analytic
- Heterogeneity
- Interoperability
- Data Security and privacy protection
- Low energy consumption

Obvious as it is, all the above-mentioned characteristics compose a huge area of research and challenges:

- Resource Management
- HW/SW co-desigm
- safety
- security
- reliability and robustness

1.3 Resource Management on Edge and Fog Computing

One of the most crucial and challenging topics of research is the way that the resources of the devices are going to be managed. It is necessary to find out efficient techniques for managing resources and allocating tasks, considering factors such as energy and time restrictions, devices limits, system and network constraints. To be more specific, we need a global policy in the network of devices, Fog and Edge entities, which will distribute computational tasks and services among IoT devices/sensors and Edge infrastructures. Resource management, in general, includes resource estimation, workload allocation and resource coordination:

- **Resource estimation** in Fog and Edge computing helps to allocate computational resources according to some policies so that appropriate resources for further computation can be allocated, desired Quality of Service(QoS) can be achieved and accurate service price can be imposed.
- **Workload allocation** in Fog and Edge computing should be done in such a way so that utilization rate of resources become maximized and longer computational idle period get minimized. More precisely, balanced load on different components is ensured.
- **Coordination** among different Fog and Edge resources is very essential as they are heterogeneous and resource constrained. Due to decentralized nature of these architectures, in most cases large scale applications are distributively deployed in different Fog nodes.

Recently, several studies have been conducted, from different points of view, as far as the resource management problem in IoT networks is concerned.

1.4 Computer Science, IoT and Economic Theory

Over the last decades, major research advances have taken place in the intersection of computer science and economics (especially microeconomic theory). There are multiple motivations for these lines of work. On the one hand, as computer systems become increasingly interconnected, their users end up competing for scarce resources, necessarily introducing economic phenomena. On the other hand, advances in computing have made the use of various novel economic mechanisms possible. The multiagent systems community has played a prominent role in these developments, as it seeks to employ techniques from economics in the design of multiagent systems, as well as to contribute to the design of new economic systems by exporting techniques from AI and multiagent systems. In this interdisciplinary research area, much of the focus has been on game theory-the theory of how to act rationally/strategically in environments with other players who strategically pursue their own objectives- and the closely related theory of mechanism design, which concerns the design of systems that result in good outcomes when used by such strategic agents, based on pricing and market theory.

Economic theory and pricing models can be also applied to IoT, Fog and Edge computing architectures. Considering a network consisted of IoT devices, these devices cannot only perform normal functions, but they also make optimal decisions without or with minimal human intervention given their constraints resources and the dynamic of the environment for the requested IoT services. In addition, with billion of devices connected to the Internet, it leads to many challenges in efficiently controlling and managing IoT's devices.

1.5 Thesis Goals and Organization

The rapid growth of IoT has exploded the number of devices connected to the Internet, a number which keeps increasing in high pace. This has led to an enormous amount of collected

data and information, traditionally offloaded to cloud computing infrastructure.

However, cloud servers and datacenters are geographically centralized, situated far from the end devices and as a consequence, real-time and latency-sensitive computation services often endure large round-trip delay, network congestion and service quality degradation. Moreover, in data-sensitive domains such as Healthcare, privacy and confidentiality concerns have been raised owned to the storage of data to third-party infrastructure. These reasons (lower latency and higher privacy) have driven the Edge computing paradigm, where the required computation is pushed to the Edge of the IoT network in order to alleviate the dependency on cloud infrastructure and enhance the privacy of identifiable personal data.

The presented work regards the well-established Edge computing architecture of IoT nodes and Gateways, where a portion of the tasks of the IoT nodes are/can be offloaded to the IoT Gateway [21]. In this setup, resources including available CPU, memory and communication bandwidth on both IoT nodes and Gateways are limited and a portion of them is shared.

Thus, an efficient resource management mechanism is required to dynamically allocate the shared Gateway resources and to designate the operating configuration of IoT nodes. The essential tuning knob of the system is the designation of tasks to be offloaded to the Gateway, an NP-complete problem which has already been addressed using centralized decision making on the Gateway [21].

The main goal of the presented work is to take advantage of the inherently distributed nature of IoT architectures in order to avoid the pitfalls of centralized decision making, i.e. increased execution latency and lack of performance scalability. **The novel contributions of this thesis are as follows:**

- An analytical model of an Edge computing setup, where tasks can be offloaded from the IoT nodes to the Gateway is presented. The model captures the properties and constraints of the involved devices and formulates the system objective as deadline miss minimization problem.
- We design and implement *DMRM*, a distributed, market-based algorithm for Edge computing resources management, which employs economic and pricing theory in order to minimize task deadline misses.
- An evaluation of the proposed algorithm is presented, which highlights its ability to designate near-optimal solutions, while achieving orders of magnitude lower execution latency and scalable performance in comparison to centralized approaches.

The remainder of this thesis is organized as follows:

- In chapter 2 is presented similar reasearch on Resource Management and similar problems.
- In chapter 3, basic economic models are discussed.
- In chapter 4, the resource management problem that will be tackled in this thesis, is presented. There is also analyzed a brute-force solution, an Oracle Prediction and a solution based on Simulated Annealing method.
- In chapter 5 *DMRM* is presented.
- In chapter 6, the results of the algorithm are extracted, analyzed and compared with other algorithms. The implemented system is evaluated.
- In chapter 7, finally, is the conclusion of this thesis and provides some ideas for future research.

Chapter 2

Related work

2.1 Resource Management on IoT and Similar Problems

Over the last decades, several works have been conducted on resource management on Edge Computing. Farzard Samie et al. [21] address the problem of QoS management for IoT devices under bandwidth, battery and processing constraints. They propose a centralized, pseudo-polynomial solution based on a dynamic programming approach, which enables reusing pre-computed solutions. Experiments show that their proposed solution improves the overall QoS by 50% compared to an unsupervised system while both meet the constraints as shown in figure 2.1.

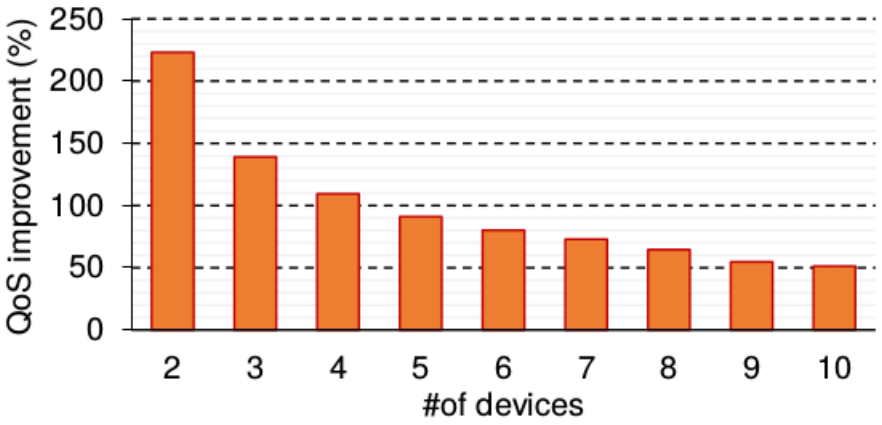


Figure 2.1: QoS imporvement [21]

In [15] the problem of computation offloading on mobile edge computing is tackled, while et al. [16] a similar problem is examined, using queuing theory.

With respect to economic theory, authors of [22] propose a distributed power management framework for heterogeneous multi-cores, based on price theory, aiming at minimizing power consumption, while satisfying performance goals under a power budget constraint. Furthermore, in [11] authors present a power budget economy for performing thermal management in many-core architectures, in order to reduce high peak temperatures. In this work, many distributed Market agents are present, serving the bids from multiple Local thermal agents. Additionally, [24] proposes a combinatorial auction-based service provider selection in Mobile Edge Computing Networks (MECs). The authors study the computation offloading, where MEC Service Providers are equipped with limited wireless and computation resources. An auction is utilized and the users' bid strategy is designed based on multi-round priority rule and the winner determination process is formulated as a two-dimensional Knapsack problem.

In [10], authors design a device-to-device offloading framework that integrates a distributed incentive scheme and a distributed reputation mechanism. In [18] Agora is presented, a resource management framework based on market principles, which deals with unavailable resources and distributes the computation burden among the chip's resources. The allocation of jobs to resources is decided by the jobs' offering price. The offering price is based on their waiting time and the price that the resources are willing to accept is based on their utilization under their current state. Additionally, the authors of [17] propose several pricing, game theory and auction-based models that can be applied in IoT issues and authors et al. [20] discuss the data offloading problem and implement an auction-based mechanism. Finally, Wong et al. [23] studied a variant of resource allocation problem at the edge of network, to control network usage from the devices.

In overview of the literature, economic models have been already used for resource management allocation purposes but have not been applied to a setup of IoT devices competing for limited, shared resources, under deadline restrictions. In addition, this thesis distinguishes from other works by introducing a more flexible, per-application awareness where the price of the resources is defined by the willingness of each distributed agent (device) to pay instead of letting the resource seller de facto define the selling price. Using this concept, a distributed market place is developed using fundamental pricing models instead of auction based approaches.

Chapter 3

Economic Theory and Models

3.1 Economic and Pricing Models on Edge Computing Architectures

Apart from classical approaches, e.g., optimization-based, economic and market based approaches have been widely applied in IoT systems. Authors et al. [17] compared the classical approaches with the economic and market based approaches. The latter provide the following benefits:

- The primary and most important benefit of economic approach is the revenue generation. The profit of IoT systems must be maximized given the revenue and cost incurred.
- Components in IoT have different objectives and constraints since they may belong to different entities. Pricing approaches can be introduced to determine optimal interactions among these self-interested and rational entities.
- In order to attract users to contribute their data, incentive mechanisms using pricing and payment strategies can be adopted. These strategies can guarantee the stable scale of participants and improve the accuracy, coverage and timeliness of the results.
- Finally, using economic and pricing models, allows selecting IoT devices with the highest remaining resources to perform sensing tasks. This can guarantee a trade-off between maximizing the network lifetime and providing the required data quality for sensors. Moreover, the pricing models can easily eliminate data redundancy without the complexity computation.

Considering all the above-mentioned benefits, economic and pricing approaches in developing IoT systems have become a great issue of research in IoT research and development industry. Figure 3.1 presents the basic economic models that can be applied on IoT networks and Edge Computing. The models are categorized into three groups based on how to set the price: economic concepts based pricing, game theory and auction based pricing, and optimization based pricing.

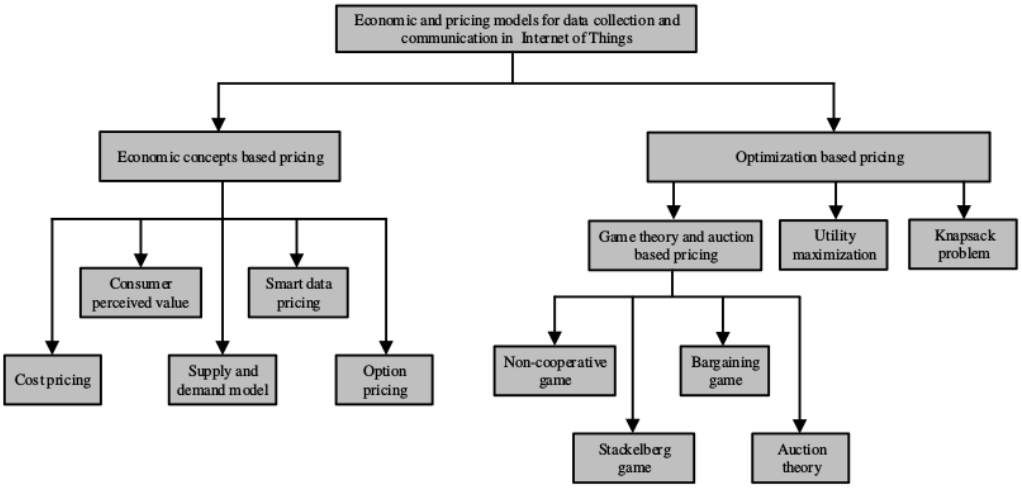


Figure 3.1: A taxonomy of economic and pricing models in IoT [17]

This diploma thesis focuses on economic concepts based pricing. These models are analyzed on the subsections below and their basic advantages are presented.

3.1.1 Cost-based Pricing

Cost-based pricing is a pricing strategy which involves determining total cost of a commodity and adding a percentage of the cost as a desired income or profit. Mathematically, the selling price of a commodity is defined by:

$$P = C * (1 + m) \tag{3.1}$$

where C is the total cost, m is the markup which is the profit percentage added to the total cost. The main advantage of the cost-based pricing is its simplicity since it requires only the internal cost information to set and adjust the selling price. However, this strategy does not consider external market factors, e.g, the competitors, the demand and the response of the buyers. Therefore, this model cannot be applied to competitive markets with various buyers and sellers.

3.1.2 Consumer Perceived Value Pricing

In order to maximize long-term profits, a seller needs to set the price by considering the buyer's perceived value from the commodity or the service rather than using traditional costs. The buyer's perceived value is the overall benefit derived at the price that the buyer is willing to pay. To set the price on the perceived value, it is necessary to identify the set of valued drivers that:

1. present value perceptions about the commodity and seller,
2. create positive attitudes and feelings,
3. provide the basis for differentiation and,
4. Provide the reason to buy the commodity.

Generally, since the perceived value pricing is the tradeoff between the perceived utility to be received and the perceived price for acquiring the information, sellers should understand these tradeoffs to maximize the buyer's value and seller's outcomes.

In IoT market, consumer perceived value pricing model is efficient to estimate the consumers potential demand for the data through determining their willingness and affordability.

3.1.3 Supply and Demand Model

Although consumer perceived value pricing considers more about the demand of buyers, it ignores the incomplete information conditions and market competition. Supply and Demand model, which is one of the most fundamental concepts of economics, can tackle the problem. In microeconomics, supply and demand is an economic model of price determination in a market. It postulates that, holding all else equal, in a competitive market, the unit price for a particular good, or other traded item such as labor or liquid financial assets, will vary until it settles at a point where the quantity demanded (at the current price) will equal the quantity supplied (at the current price), resulting in an economic equilibrium for price and quantity transacted. The supply and demand are parts of economic model in which the relations between these two factors can be exploited to determine the price of a commodity in a market. Price, therefore, is a reflection of supply and demand. The relationship between demand and supply underlie the forces behind the allocation of resources. In market economy theories, demand and supply theory will allocate resources in the most efficient way possible. Let us take a closer look at the law of demand and the law of supply as presented et al. [2] :

1. **The Law of Demand** : The law of demand states that, if all other factors remain equal, the higher the price of a good, the less people will demand that good. In other words, the higher the price, the lower the quantity demanded. The amount of a good that buyers purchase at a higher price is less because as the price of a good goes up, so does the opportunity cost of buying that good. As a result, people will naturally avoid buying a product that will force them to forgo the consumption of something else they value more. The chart below shows that the curve is a downward slope.

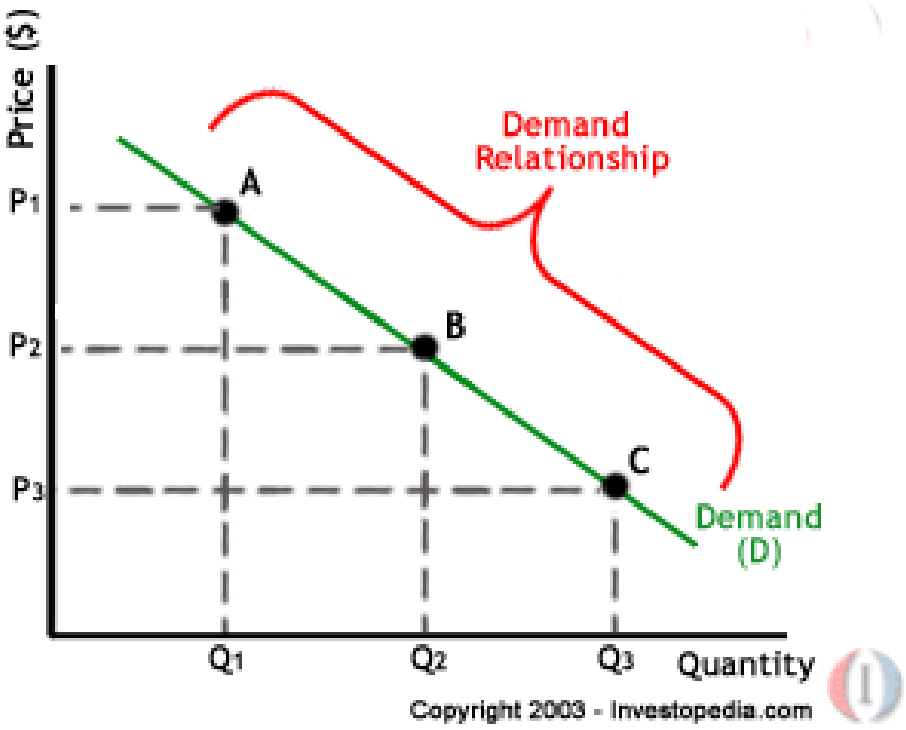


Figure 3.2: Demand Law [2]

A, B and C are points on the demand curve. Each point on the curve reflects a direct correlation between quantity demanded (Q) and price (P). So, at point A, the quantity demanded will be Q1 and the price will be P1, and so on. The demand relationship curve illustrates the negative relationship between price and quantity demanded. The higher the price of a good the lower the quantity demanded (A), and the lower the price, the more the good will be in demand (C).

2. **The Law of Supply** : Like the law of demand, the law of supply demonstrates the quantities that will be sold at a certain price. But unlike the law of demand, the supply relationship shows an upward slope. This means that the higher the price, the higher the quantity supplied. Producers supply more at a higher price because selling a higher quantity at a higher price increases revenue. A, B and C are points on the supply curve. Each point on the curve reflects a direct correlation between quantity supplied (Q) and price (P). At point B, the quantity supplied will be Q2 and the price will be P2, and so on.

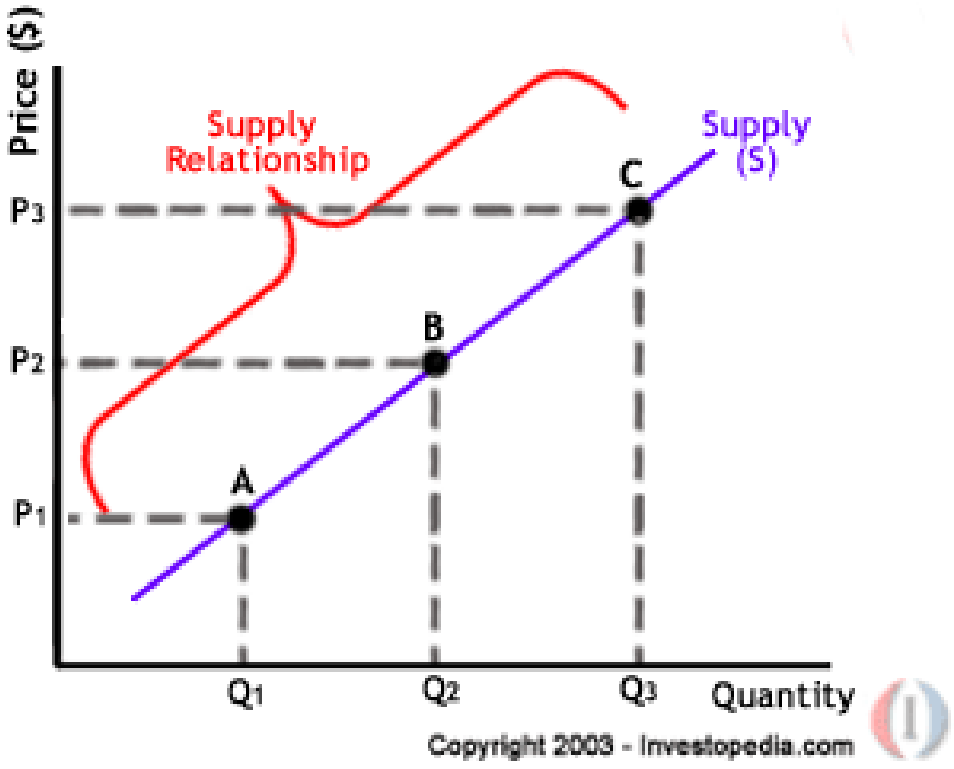


Figure 3.3: Supply Law [2]

When supply and demand are equal (i.e. when the supply function and demand function intersect) the economy is said to be at equilibrium. At this point, the allocation of goods is at its most efficient because the amount of goods being supplied is exactly the same as the amount of goods being demanded. Thus, everyone is satisfied with the current economic condition. At the given price, suppliers are selling all the goods that they have produced and consumers are getting all the goods that they are demanding.

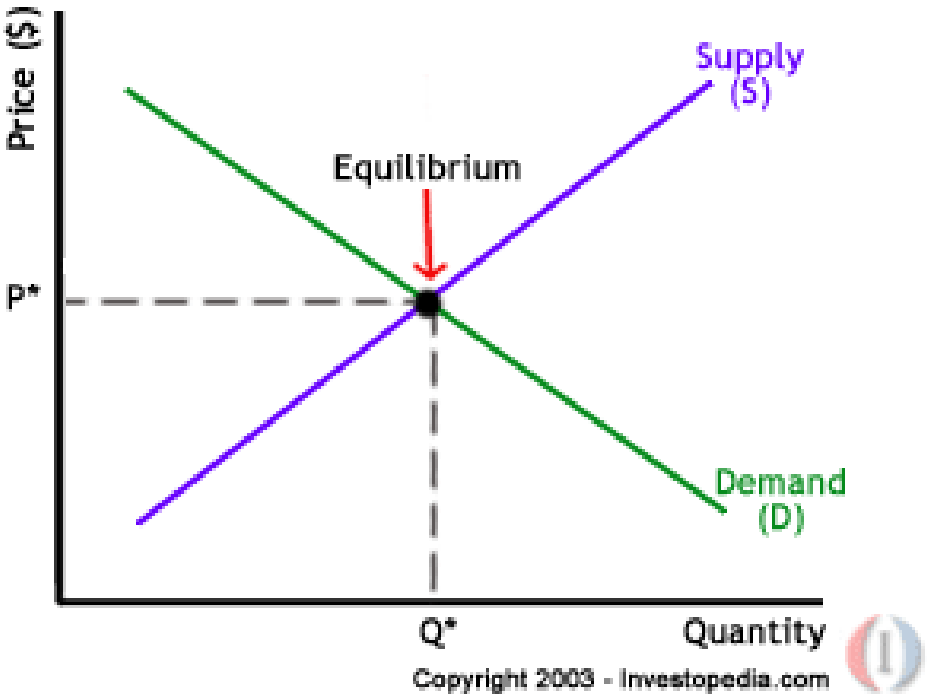


Figure 3.4: Supply Law [2]

Consider a competitive market in which multiple sellers and buyers compete for commodities. Assume that the relationship between the price P for the commodity and the quantity demanded Q_d is expressed by the linear function as $P = a - bQ_d$, where a and b are coefficients of the function. Similarly, a linear supply function which presents the relationship between P and the quantity supplied Q_s is given by $P = c + dQ_s$. Accordingly, the sellers and the buyers tend to change the price and the quantity to maximize their objective functions, e.g, profit and utility, budget to some constraints. Both buyers and sellers try to determine a unique point called market equilibrium, at which $Q_d = Q_s = Q^*$ and $P = P^* = \frac{a-c}{b+d}Q^*$. The market equilibrium has some basic properties :

1. The behavior of sellers and buyers is consistent since the demand equals supply.
2. No seller or buyer has an incentive to change its behavior since their actual trades equal to their desired trades.
3. At any non-equilibrium price, there will be excess supply or demand which leads to a movement in price towards the equilibrium.

3.1.4 Smart Data Pricing

To avoid large peak demands created by users' simultaneous consumption of scarce resources, the Smart Data Pricing(SDP) schemes are proposed.The SDP refers to various techniques such as charging buyers depending on the resource usage time, setting location based tarrifs, and imposing prices based on buyer activity levels. Two common approaches are used in IoT which are time-dependent pricing and usage-based pricing:

- **Time-dependent Pricing** : The prices of the resource have a temporal component, i.e., the prices vary over different time to discourage large peak demands. Only when buyers be incentivized to diffuse their demands over time, thus improving resource efficiency by reducing the peaks and filling up the valley periods.

- **Usage-based pricing** : On that approach, buyers are charged according to access rate on resource usage. If the price is set based on usage, a fair and efficient use of resources would be moderately promoted to some extent.

The two above approaches to managing resources can create a "win-win" solution for both buyers and sellers. The sellers benefit by reducing peak congestion while users are offered more choices and technologies to save on their costs. Compared to Supply and Demand model, the SDP has no need of the iteration to determine the market-cleaning price, meaning that the market is cleared as soon as buyers have submitted their bid for access.

On IoT and Edge computing, SDP be can introduced as an alternative solution to address resource management issues, e.g, power management, resource demand management, and price setting for resources in Cloud computing applications.

3.1.5 Option Pricing

An option concept in finance is a contract which gives the buyer, i.e., the owner of the option, the right to buy or sell an underlying asset or instrument at a fixed price, i.e., a specified strike price, at any time on or before a specified date. The buyer then pays a price, also known as the premium, to the seller in exchange for the right granted by the option. To determine the price or value of an option, the Black-Scholes model in the option pricing theory is typically used. Accordingly, the price of a real option depends on various factors such as the current value and the uncertainty of expected cash flows, the value of fixed costs, the risk free rate of return, the time to maturity of the option, and any value lost over the duration of the option.

In IoT architectures, this model can basically be applied for the IoT investment evaluation, the resource reservation in Machine-to-Machine communications and the task scheduling.

3.1.6 Other models

As shown previously, apart economic concepts based pricing models, there exist other approaches based on game theory and auction based pricing, and optimization based pricing. As authors et al. [17] mention, game theory and auctions are the formal study of decision-making where several players must make choices that potentially affect the interests of the other players.

Game theory can be defined as "the study of mathematical models of conflict and cooperation between intelligent rational decision-makers". There exists a variety of models based on game theory that can be applied on IoT networks and Edge Computing Architectures. The most significant and common used are:

1. **Non-cooperative game** : A game is known as non-cooperative if it is not possible for the players to form coalitions or make agreements. Non-cooperative games are basically used for resource allocation in wireless networks. In the Cloud computing area this model is used to maximize profits gained of the competitive cloud providers.

2. **Stackelberg game** : The Stackelberg leadership model is a strategic game in economics in which the leader firm moves first and then the follower firms move sequentially.

3. **Bargaining games** : Bargaining games refer to the situations in which two or more players must reach an agreement regarding how to distribute an object.

Additionally, an auction is a process of buying and selling goods or services by offering them up for bid, taking bids, and then selling the item to the highest bidder [8]. Similar to game theory, several auction-based theorems and models exist and can be applied on Edge Computing. Some of them are:

1. Sealed-bid auctions: In [9] sealed-bid auction is defined as a type of auction process in which all bidders simultaneously submit sealed bids to the auctioneer, so that no bidder knows how much the other auction participants have bid. The highest bidder is usually declared the winner of the bidding process.

2. Forward, reverse, and double auctions: The sealed-bid auctions mentioned above are classified based on the seller's side and they are the forward auctions. Considering the buyer's side, there are reserve and double auctions. In a forward auction, buyers bid for items by offering increasingly higher prices. In a reverse auction, sellers compete for buyers' attraction by submitting their asks to the auctioneer and in a double auction, buyers and sellers simultaneously submit their bids and asks to an auctioneer and the auctioneer tries to define the perfect match.

3. Combinatorial auction: In a combinatorial auction, a buyer submits its bid among with the need of a whole bundle of multiple items.

3.2 Economic Models and Resource Management

IoT devices are generally very limited as far as their computational, memory, bandwidth and battery resources is concerned. Traditional resource allocation algorithms often assume that the available resources in a system such as network bandwidth do not change. However, the amount of available resources on the devices is not constant. Pricing mechanisms are used as solutions in which all scarce resources, e.g., CPU, storage, bandwidth, and battery, can be best utilized with the variability of resource budget. Similar to resource allocation, task allocation is to assign sensing tasks to specific devices in the network for execution. Given the constrained resource and distributed structure of IoT devices and Fog nodes, the goal of the task allocation scheme is to achieve a fair energy and resource balance among the sensors while minimizing the delay. To address this problem, price formulations can be used as they can continuously adapt to changes of the resource availabilities. Supply and demand model can be applied for resource allocation in order to reach an equilibrium for the resources supplied and the resources demanded. Furthermore, the smart data pricing can be employed to make users aware of high cost when consuming bandwidth during the peak demand periods. Finally, auctions can be used for task allocation and game theory model for competing the available resources.

In this diploma thesis, the Supply and Demand, Smart Data Pricing and Consumer Perceived Value Pricing, which are presented in chapter 3.1 are applied in order to deal with the resource management and task allocation in a network consisted of heterogeneous IoT devices.

Chapter 4

Problem Formulation and Solutions

This diploma thesis tackles a resource management problem on Edge computing systems and IoT networks. In this chapter the problem is strictly formulated. More specifically, the architecture of the IoT network is described, the characteristics of all devices and applications are analyzed and the basic goals are presented. Afterwards, three different approaches of the problem are analyzed:

- Brute-force Approach
- Oracle Prediction
- Simulated Annealing(SA) Approach

4.1 System and Problem Formulation

The architecture of the examined network consists of two basic types of devices:

1. Edge Nodes(Gateways) : Edge computing entities, which enable the deployment of Edge and Fog services and offer computation, storage and networking resources to IoT devices.
2. IoT Devices : physical devices, vehicles, home appliances and other items embedded with electronics, software, sensors, actuators, and connectivity which enables them to connect and exchange data. These devices are resource constrained as far as their cpu, storage and networking capabilities is concerned.

On the one side, the IoT devices exist on the terminal layer of the network and they are close to end users. On the other side, Gateway exists between Fog layer and terminal layer. Considering the networking and the communication of the devices, all IoT devices are connected with different network technologies(wired or wireless) to the Gateway, with which they can exchange data. However, IoT devices cannot communicate among them directly.

Our target architecture envisions a system of n IoT nodes, one Gateway device and a number tasks per device, which can be executed either on the Gateway or locally on the IoT node, as shown in Fig. 4.1. The IoT nodes are resource constrained as far as their CPU, memory capacity and communication capabilities are concerned, and can offload their tasks to the Gateway device. Considering the communication of the devices, IoT devices can be connected with any wired or wireless communication protocol supported by the Gateway.

Each IoT device is uniquely defined by a value $i \in \{1, ..., n\}$ and is specified by the tuple $D_i = \{C_i, M_i, B_i, Money_i, \mathcal{T}_i\}$.

Variables C_i, M_i, B_i denote the CPU, memory and bandwidth resources of device i, respectively. $Money_i$ denotes the total money of the device and $\mathcal{T}_i$ is the set containing all the tasks

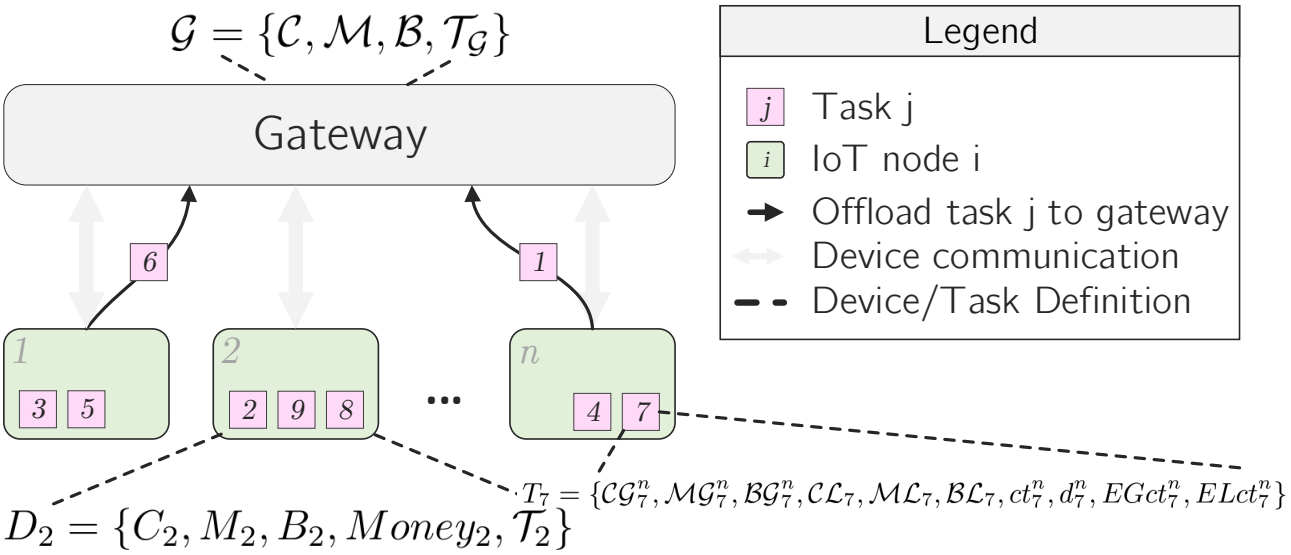


Figure 4.1: Target system architecture

τ_j^i of device i. For every task, two auxiliary variables characterize its execution. For the task j of device i, o_j^i is set to 1 if the task is executed on the Gateway or set to 0, otherwise. Similarly, l_j^i is set to 1 if the task is executed locally or set to 0, otherwise. The case that both variables are set to zero is valid, meaning that that execution of the task has been postponed.

On the Gateway, the IoT devices can acquire resources, in order to have their tasks executed. Similar to the IoT devices, the Gateway is defined by a tuple $\mathcal{G} = \{\mathcal{C}, \mathcal{M}, \mathcal{B}, \mathcal{T}_\mathcal{G}\}$.

Variables $\mathcal{C}, \mathcal{M}, \mathcal{B}$ denote the CPU, memory and bandwidth resources of the Gateway, respectively. $\mathcal{T}_\mathcal{G}$ is the set of tasks offloaded from the IoT devices to the Gateway, at any moment. Every task of device i is defined by its own unique id j and is specified by the tuple:

$$T_j = \{\mathcal{CG}_j^i, \mathcal{MG}_j^i, \mathcal{BG}_j^i, \mathcal{CL}_j, \mathcal{ML}_j, \mathcal{BL}_j, ct_j^i, d_j^i, EGct_j^i, ELct_j^i\} \tag{4.1}$$

where $\mathcal{CG}_j^i, \mathcal{MG}_j^i, \mathcal{BG}_j^i$ denote the utilization requirements of task j if executed on the Gateway and $\mathcal{CL}_j, \mathcal{ML}_j, \mathcal{BL}_j$ the respective requirements if executed locally. The variable ct_j^i is the time that a task completes its execution, while d_j^i is its associated deadline. Variables $EGct_j^i$, $ELct_j^i$ also indicate the estimated execution time of each task of device j on the Gateway and locally, respectively.

The key parameters of the system model are also summarized in Table 4.1.

The optimization objective of the system as expressed in Eq. 4.2, is to minimize the number of tasks of IoT nodes that exceed their deadline. The solution is constrained by the available CPU, memory and bandwidth resources both in IoT nodes and Gateway (Eq. 4.4 to Eq. 4.9).

$$\text{minimize} \quad N_{missed} = \sum_{\forall i}(\sum_{\forall j} miss(\tau_j^i)) \tag{4.2}$$

$$\text{where} \quad miss(\tau_j^i) = \begin{cases} 0 & if\ ct_j^i \leq d_j^i \\ 1 & otherwise \end{cases} \quad \text{subject to:} \tag{4.3}$$

Denotation	Description
τ_j^i	task j of device i
o_j^i	1 if task j of device i is executed (offloaded) to Gateway, 0 otherwise
l_j^i	1 if task j of device i is executed locally, 0 otherwise
$\mathcal{C}$, $\mathcal{M}$, $\mathcal{B}$	CPU, Memory and Bandwidth resources of Gateway
C_k, M_k, B_k	CPU, Memory and Bandwidth resources of device k
$\mathcal{CL}_j$, $\mathcal{ML}_j$, $\mathcal{BL}_j$	CPU, Memory and Bandwidth resources required for task j executed locally
$\mathcal{CG}_j^i$, $\mathcal{MG}_j^i$, $\mathcal{BG}_j^i$	CPU, Memory and Bandwidth resources required for task j of device i executed on Gateway
ct_j^i, d_j^i	Completion time and deadline of task j of device i
$EGct_j^i$, $ELct_j^i$	Estimated completion time of task j of device i on Gateway and locally, respectively

Table 4.1: Key system model parameters

$$\textit{Gateway} \quad \sum_{\forall i}(\sum_{\forall j} o_j^i \cdot \mathcal{CG}_j^i) \leq \mathcal{C} \qquad \textit{CPU} \tag{4.4}$$

$$\textit{constraints:} \quad \sum_{\forall i}(\sum_{\forall j} o_j^i \cdot \mathcal{MG}_j^i) \leq \mathcal{M} \qquad \textit{Memory} \tag{4.5}$$

$$\sum_{\forall i}(\sum_{\forall j} o_j^i \cdot \mathcal{BG}_j^i) \leq \mathcal{B} \qquad \textit{Bandwidth} \tag{4.6}$$

$$\textit{Device } k \quad \sum_{\forall j} l_j^k \cdot \mathcal{CL}_j \leq C_k \qquad \textit{CPU} \tag{4.7}$$

$$\textit{constraints:} \quad \sum_{\forall j} l_j^k \cdot \mathcal{ML}_j \leq M_k \qquad \textit{Memory} \tag{4.8}$$

$$\sum_{\forall j} l_j^k \cdot \mathcal{BL}_j \leq B_k \qquad \textit{Bandwidth} \tag{4.9}$$

$$\textit{Unique execution} \quad o_j^k + l_j^k \leq 1, \forall j \in \mathcal{T}_k \tag{4.10}$$

The specified requirement of *Unique Execution* in Eq. 4.10 means that a task is prohibited to be executed both on an IoT node and Gateway, while the postponing of a task i remains a valid choice. Our proposed market-based solution aims at solving the optimization problem by dynamically designating where each task of the IoT nodes will be executed on.

Last but not least, it should be mentioned that the tasks can be grouped in three categories according to their required resources:

- CPU intensive tasks, which are those that require high CPU resources in order to be executed,
- Memory intensive tasks, which are those that require high memory resources in order to be executed and
- Bandwidth intensive tasks, which are those that require high bandwidth rate resources in order to be executed on time.

4.2 Brute-Force Solution

In computer science, brute-force search or exhaustive search, also known as generate and test, is a very general problem-solving technique that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement. While a brute-force search is simple to implement, and will always find a solution if it exists, its cost is proportional to the number of candidate solutions – which in many practical problems tends to grow very quickly as the size of the problem increases. Therefore, brute-force

search is typically used when the problem size is limited, or when there are problem-specific heuristics that can be used to reduce the set of candidate solutions to a manageable size. The method is also used when the simplicity of implementation is more important than speed. For instance, a brute-force algorithm to find the divisors of a natural number n would enumerate all integers from 1 to n, and check whether each of them divides n without remainder.

4.2.1 Algorithm

The brute-force approach was implemented on the resource management problem of this diploma thesis, in order to be able to find for any set of given IoT devices and tasks the best possible scheduling among all possible schedulings. More Specifically, this algorithm finds all possible schedulings and returns the best possible scheduling, for which the number of tasks that exceeded their deadline d is minimized and, on the same time, the delaytion of these tasks is minimized. Additionally, it must be mentioned that this exhaustive solution is executed when new tasks arrive and we don't anything about the set of tasks in future rounds.

Given a set of tasks T, a set of IoT devices I and a Fog node F we need to find all possible schedulings. Assume that the time is counted on rounds R, starting from round 0. Firstly, we find all possible combinations that exist on round i. Then we insert each of these combinations to the corresponding devices and start the execution. Afterwards, we get to the next round to find again all possible combinations. This process happens recursively until all tasks have been executed and terminated. Finally, the function returns the current minimum number of delayed tasks to the previous call and the next combination is inserted. The brute-force recursive algorithm is presented in Algorithm 5.

Algorithm 5: Brute-force Algorithm

```
Result: Find the minimum number of delayed tasks.
Data: Gateway, DevTuple, Tasks
1  minimum = numberOfTasks + 1;   // in worst case all tasks will be delayed
   Brute-Force(DevTuple,Gateway,Tasks,round):
2    checkForTerminatedTasks(DevTuple,Gateway,Tasks,round);          // check of
       terminated tasks
3    if all tasks terminated then
4      delayed = countDelayedTasks(); // count the tasks tat exceeded deadline
5      return min(delayed,minimum);
     // find all possible combinations of task scheduling at current round
6    combinations = findAllCombinations(DevTuple,Gateway,Tasks,round);
7    if combinations = NULL & exist undone tasks then
8      minimum = Brute-Force(DevTuple,Gateway,Tasks,round+1);   // recursive
         call
9      return minimum;
10   while combinations!=NULL do
11     insertNextCombination(DevTuple,Gateway,Tasks,combinations);  // on fog or
         IoT device
12     minimum = Brute-Force(DevTuple,Gateway,Tasks,round+1);   // recursive
         call
13   return minimum;
```

4.2.2 Results

From the algorithm presented in subsection 4.2.1, if we have a number of N tasks, M devices and Fog node at round i , the number of possible combinations extends from 0 to 2^{N+1}. The upper limit of the consists of a huge number of combinations, which means that the complexity and the execution time of the algorithm. The number of combinations is exponentially related to the number of tasks and the possible combination is calculated. almost on every round of the recursion. Consider an input of 100 tasks. At round 0, there can exist from 0 to 2^{101} different combinations and this process will be executed recursively. Therefore, the memory needed and the complexity of the algorithm becomes prohibitively high. The results of this algorithm are further analyzed and compared at chapter 6.2.

4.3 Oracle Prediction

Similar to brute-force solution, the Oracle prediction is an exhaustive search of all possible candidates for the solution. The basic difference with brute-force is that Oracle prediction mechanism knows a priori all the tasks that will arrive at the system. Therefore, the Oracle scheduler can lead to less deadline misses by postponing the execution of a task in order to save resources for a task arriving in later rounds and has stricter deadline. However, the time and the memory required for this solution skyrocket.

4.4 Simulated Annealing

4.4.1 Theoretical Background

According to [6], simulated annealing (SA) is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a metaheuristic to approximate global optimization in a large search space. It is often used when the search space is discrete (e.g., all tours that visit a given set of cities). For problems where finding an approximate global optimum is more important than finding a precise local optimum in a fixed amount of time, simulated annealing may be preferable to alternatives such as gradient descent. The simulated annealing algorithm was originally inspired from the process of annealing in metal work. Annealing involves heating and cooling a material to alter its physical properties due to the changes in its internal structure. As the metal cools its new structure becomes fixed, consequently causing the metal to retain its newly obtained properties. In simulated annealing we keep a temperature variable to simulate this heating process. We initially set it high and then allow it to slowly 'cool' as the algorithm runs. While this temperature variable is high the algorithm will be allowed, with more frequency, to accept solutions that are worse than our current solution. This gives the algorithm the ability to jump out of any local optimums it finds itself in early on in execution. As the temperature is reduced so is the chance of accepting worse solutions, therefore allowing the algorithm to gradually focus in on a area of the search space in which hopefully, a close to optimum solution can be found. This gradual 'cooling' process is what makes the simulated annealing algorithm remarkably effective at finding a close to optimum solution when dealing with large problems which contain numerous local optimums.

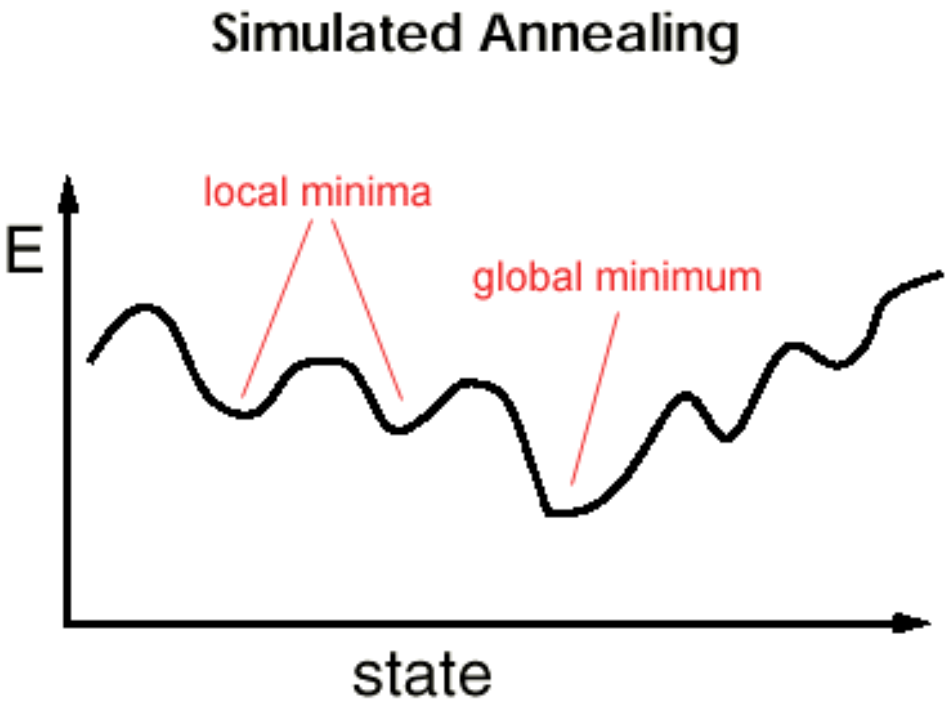


Figure 4.2: Simulated Annealing [1]

Although other similar algorithms for global optimums, such as hill climbing algorithm, can be surprisingly effective at finding a good solution, they also have a tendency to get stuck in local optimums. The simulated annealing algorithm is excellent at avoiding this problem and is much better on average at finding an approximate global optimum. That is the basic advantage of simulated annealing over other algorithms.

Simulated annealing (SA) has been already used for task scheduling [13, 14] and application mapping [19].

4.4.2 Algorithm

The basic iteration of the algorithm is that at each step, the simulated annealing heuristic considers some neighboring state s* of the current state s, and probabilistically decides between moving the system to state s* or staying in-state s. These probabilities ultimately lead the system to move to states of lower energy. Typically this step is repeated until the system reaches a state that is good enough for the application, or until a given computation budget has been exhausted. More specifically, in order to accept a solution(state) first we check if the neighbour solution is better than our current solution. If it is, we accept it unconditionally. If however, the neighbour solution isn't better we need to consider a couple of factors. Firstly, how much worse the neighbour solution is; and secondly, how high the current 'temperature' of our system is. At high temperatures the system is more likely accept solutions that are worse. The math for this is pretty simple:

$$exp(\frac{solutionEnergy - neighbourEnergy}{temperature}) \tag{4.11}$$

Basically, the smaller the change in energy (the quality of the solution), and the higher the temperature, the more likely it is for the algorithm to accept the solution. Given the acceptance function on the above, the basic implementation of the algorithm is pretty simple:

- First we need set the initial temperature and create a random initial solution.
- Then we begin looping until our stop condition is met. Usually either the system has sufficiently cooled, or a good-enough solution has been found.
- From here we select a neighbour by making a small change to our current solution.
- We then decide whether to move to that neighbour solution.
- Finally, we decrease the temperature and continue looping.

As far as the temperature initialization is concerned, for better optimisation, when initialising the temperature variable we should select a temperature that will initially allow for practically any move against the current solution. This gives the algorithm the ability to better explore the entire search space before cooling and settling in a more focused region.

In order to apply the simulated annealing method to a specific problem, one must specify the following parameters: the state space, the energy (goal) function E(), the candidate generator procedure neighbour(), the acceptance probability function P(), and the annealing schedule temperature() and the initial temperature <init temp>. These choices can have a significant impact on the method's effectiveness. Unfortunately, there are no choices of these parameters that will be good for all problems, and there is no general way to find the best choices for a given problem.

The algorithm is presented using pseudocode on Algorithm 6. The constants and the initial values of the algorithm can be set empirically and can determine the quality of the final solution and the speed of convergence, in order to reach this solution.

Algorithm 6: Simulated-Annealing Algorithm

```
Result: Minimuze the number of delayed tasks
Data: Gateway, DevTuple, Tasks
Simulated-Annealing(Gateway, DevTuple, Tasks):
1   initialize(temperature, solution);        // initial temperature and a solution
2   while coolIteration <= maxIterations do
3       coolIteration = coolIteration + 1;
4       tempIteration = 0;
5       while tempIteration <= nrep do
6           tempIteration = tempIteration + 1;
7           newSol = createNewSolution();                  // generate new solution
8           currentEnergy = computeEnergy(newSol);         // energy of new solution
9           d = currentEnergy - previousEnergy;    // compare previous and current
             energy
10          if d < 0 then
11              Accept new solution
12          else
13              Accept new soluiton with probability exp(-d/temperature)
14      T = a * T;                                                        // 0<a<1
```

The simulated annealing method was applied on the resource management problem. The goal is to find a task scheduling among the devices, which will minimize the number of tasks that exceeded their deadline. In other words, we search for a global minimum. Firstly, an initial solution(an initial task scheduling) is randomly defined. Afterwards, at each step a new solution is generated. Each solution is estimated, according to the number of delayed tasks

that is extracted and the result compared with the already accepted solution by previous steps. According to the comparison of the solutions with the acceptance function, the best solution is chosen. Finally, after a specific number of iterations(when the system has cooled) the algorithm terminates.

4.4.3 Results

The Simulated Annealing method can guarantee that the algorithm will not get stuck to local optimums. However, it cannot guarantee that it will reach the global optimum. Additionally, not all the solution space is examined. This means that the simulated annealing will be executed faster compared to the brute-force approach. The execution time and the convergence of the algorithm depends on the initial temperature, the initial solution and the randomness of the solutions generated. The results of this method are analyzed further and compared with other approaches in chapter 6.2.

Chapter 5

DMRM: Distributed Market-based Resource Management

In order to achieve an efficient solution to the problem that has been formulated in chapter 4, *DMRM* has been implemented: distributed market-based resource management approach. In this chapter the theoretical background and the basic idea of *DMRM* is presented and analyzed. Afterwards, the basic mechanisms, which contribute the DMRM solution are analyzed.

5.1 Theoretical Background-Main Proposed Concepts

The key idea of the proposed algorithm is to create a distributed marketplace, where buyers and sellers interact with each other and compete for shared resources. The IoT devices act as buyers, who demand resources according to the virtual money in their possession. The Edge Gateway is the seller of the market, who supplies its computing, memory and bandwidth resources in a specific price. The advantage of this approach is that (i) decision making burden is distributed to numerous agents and (ii) each IoT node is empowered with the ability to specify the priority and importance of its tasks. The downside of these attributes is that the final decision maker (in our case the Gateway) may have knowledge of only a subset of the IoT tasks. This might lead to sub-optimal scheduling decisions and this explains our meticulous design of the reasoning in the IoT nodes, to avoid this pitfall. Our proposed solution is built according to the following fundamental pricing and economic models.

- **Supply and Demand model**, which postulates that, holding all else equal, in a competitive market, the unit price for a particular good or commodity will vary until it settles at a point where the quantity demanded equals the quantity supplied, resulting in an economic equilibrium. In the examined case, the relationship between demand and supply underlie the forces behind the allocation of resources.

- **Consumer Perceived Value Pricing**, where in order to maximize long-term profits, a seller needs to set the price by considering the buyer's perceived value from the commodity or the service rather than using traditional costs. The buyer's perceived value is the overall benefit derived at the price that the buyer is willing to pay.

- **Smart Data Pricing**, where buyers are charged according to access rate on resource usage. If the price is set based on usage, a fair and efficient use of resources is promoted.

Consumer perceived value pricing and smart data pricing are combined to create a pricing mechanism. In particular, the bidding price for a resource is set by the buyers (IoT nodes) according to their estimated resource usage each time. The final decisions are made on the

Gateway, where all bids are accumulated. There, the supply and demand model is used to reach an equilibrium between the demanded and supplied resources, while maximizing the Gateway profit and minimizing deadline misses of the IoT tasks. In order for a market to operate efficiently, disequilibrium and excess supply and demand must be avoided.

The basic goals of the algorithm for each entity of the system are listed on the below:

- IoT Devices(buyers) : Have all their tasks done on time, under the resources and deadline constraints.
- Gateway(seller) : Maximize its profit from the resources sold.
- System Goal : Execute the tasks under their resources and deadline constraints and reach a market equilibrium.

The market implemented is presented in figure 5.1

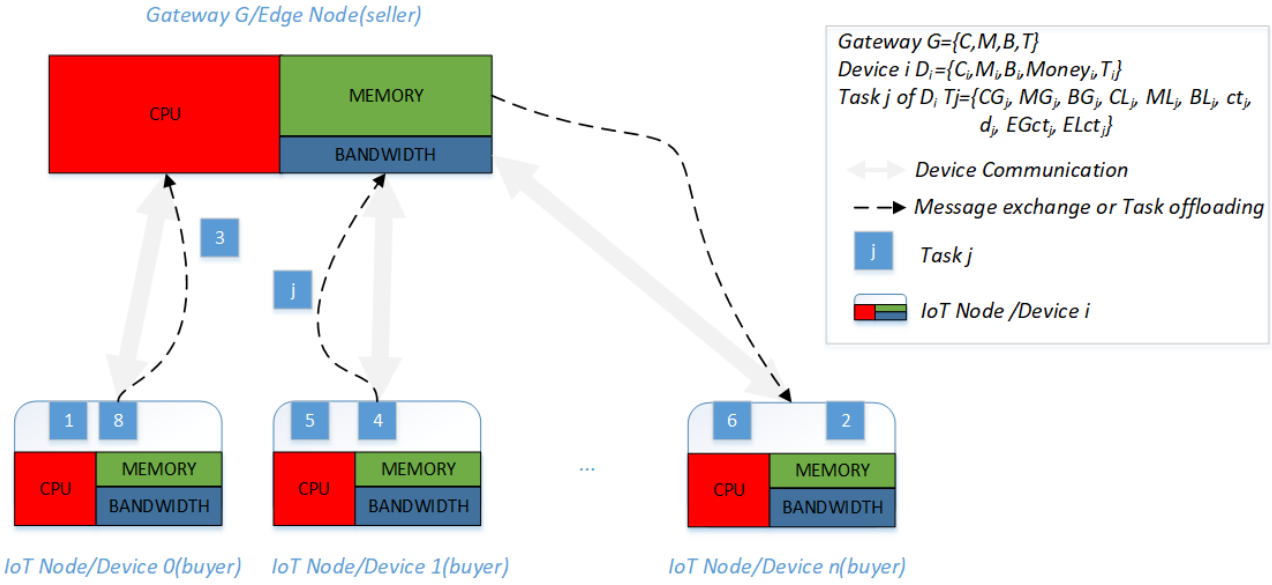


Figure 5.1: Market Architecture

5.2 DMRM's Mechanisms

To achieve the aforementioned goals, *DMRM* is introduced, composed of four core mechanisms divided amongst IoT nodes and Gateways.

- Money Distribution Mechanism
- Task Selection Mechanism
- Offloading Mechanism
- Bidding Mechanism

Money Distribution and *Task Selection* mechanisms are executed on the IoT Gateway, while the *Offloading* and *Bidding* mechanisms are executed on each IoT node. The combination of these mechanisms via data exchange between IoT nodes and the Gateway completes DMRM leading to its outcome, i.e. the decision of where each task of the IoT nodes will be executed, either on the Gateway or locally.

It is important to note the following advantage of the market-based approach to resource management in Edge computing systems. With the continuous integration and adoption of Edge computing by commercial enterprises, it is foreseen that the provided services, while follow a business model similar to the one of cloud resources. According to this model, the services and resources of an IoT Gateway, will be available for sale according to the specific needs of each customer. Consequently, a market-based resource management model can be seamlessly customized to such a business model, since pricing of resources is the principle idea.

5.2.1 Decision making functions of DMRM on the IoT node

The portion of DMRM executed on an IoT node is composed of two critical parts, i.e. the *Task Offloading* and *Bidding* mechanisms.

Task Offloading Mechanism

The *Task offloading* mechanism, defines whether a task will be executed locally or offloaded to the Gateway, with the aim to minimize the number of the tasks that exceed their deadline. To achieve that, for every single task a metric is defined, which estimates the probability of the specific task to miss its deadline. We introduce this metric as **sensitivity (s)** and for the task τ_j^i of device i it is defined as (Chapter 4):

$$s_{\tau_j^i} = \frac{(d_j^i - EGct_j^i - r) + (d_j^i - ELct_j^i - r)}{2} \tag{5.1}$$

where r is the current execution moment (measured in rounds) of the system. The lower the sensitivity, the more likely it is for the task to outrun its deadline.

Subsequently to the per task calculation of sensitivity, the tasks are sorted in ascending order, according to it. Therefore, the first task in each device is the most likely to exceed the time deadline. Starting from the first task and traversing through all of them, the IoT node has three options:

- offload the task to the Gateway,
- execute the task locally or
- suspend the execution of the task for future execution on the Gateway or on the IoT device.

We assume, that the tasks examined at each round are those that have not been executed yet and assume no knowledge about future tasks. In other words, the decisions made at each round depend only on the past and present condition of the device. According to the following criteria, each device decides if a task will be offloaded or not.

- If an IoT device does not possess the required resources of the task then it decides to offload it to Gateway.
- If an IoT device has all the required resources, but the estimated offloading latency is lower than the estimated local execution latency, then it decides to offload it.
- Otherwise, the IoT device executes the task locally, or postpones its execution.

For those tasks that are decided to be offloaded a bid should be defined via the *Bidding* mechanism, in order for the IoT node to buy the required resources from the Gateway.

Bidding Mechanism

The *Bidding* mechanism is one of the most crucial factors of the market-based approach, in order for the system to operate efficiently. This process defines the bidding that each IoT device is willing to pay for the required resources and by design it can efficiently cooperate with the *Money distribution* mechanism, i.e. the selling price of the resources is adapted to the total available money in the market. In practice, the price that a resource is sold is defined so that it is affordable by the majority of the IoT devices. This assures that resource prices and biddings are not be prohibitively high, so that devices are able to pay for those resources, and not extremely low, thus creating a competitive marketplace. The bidding estimation for a task is calculated on the corresponding IoT device and the final bid and task details are sent to Gateway. The actual bidding for a specific task is calculated according to its required resources on the Gateway, its sensitivity and, of course, the current bidding capabilities of the IoT device. The bidding calculation function for task τ_j^i is provided in Eq. 5.2.

$$B_{\tau_j^i} = \begin{cases} \mathcal{CG}_j^i + \mathcal{MG}_j^i + \mathcal{BG}_j^i - s_{\tau_j^i} * b, & \text{if } s_{\tau_j^i} \leq 0 \\ \mathcal{CG}_j^i + \mathcal{MG}_j^i + \mathcal{BG}_j^i + \frac{1}{s_{\tau_j^i}} * c, & \text{if } s_{\tau_j^i} > 0 \end{cases} \tag{5.2}$$

Constants b and c are set to 1000 and 800 and have been defined according to extensive system simulation. The estimated bidding is compared to the total money $Money$ of the device. There exist two possible scenarios:

1. If $B_{\tau_j^i} \leq Money$ then the task is offloaded to Gateway with the estimated bidding.
2. Otherwise, if $B_{\tau_j^i} > Money$, the IoT device cannot afford the bidding price that has been calculated Eq. 5.2. Then, the task is sent with bidding price equal to $Money$, i.e. the device bids all its remaining money.

Algorithm 7: DMRM functionality on IoT nodes

```
   Data: Gateway, IoT device Tuple, Set of Tasks
   IoT-Algorithm(Gateway, DevTuple, Tasks, Money):
1    volatile curRound /* Round updated at background */
2    /* Remaining Tasks */
3    RemTasks = checkTasks(Tasks, curRound)
4    while length(RemTasks) > 0 do
5        /* Invoke Task Offloading Mechanism */
6        oTasks = offloading(Tasks, DevTuple, curRound)
7        /* Invoke Bidding Mechanism */
8        bids = bidding(oTasks, Money)
9        /* Send offloading proposition to Gateway */
10       sendTasksGateway(oTasks, bids)
11       /* Wait for answer */
12       answer = waitAnswers(Gateway, offloaded)
13       for t in oTasks do
14           if answer(t) = "Accept" then
15               payGateway(Money) /* Pay Gateway */
16           else
17               RemoveFromList(t, oTasks)
18       RemTasks = checkTasks(Tasks, curRound)
```

Algorithm 7 summarizes the DMRM software functionality on an IoT node, which possess an amount of *Money* and a set of *Tasks* to be executed under specific deadlines.

It specifies the interaction of the *Task Offloading* and *Bidding* mechanisms, as well as how bids and responses are transmitted to and from the Gateway and appropriately handled.

5.2.2 Decision making functions of DMRM on the Gateway

The portion of DMRM executed on the Gateway is also dominated by two parts, i.e. the *Money Distribution* and *Task Selection* mechanisms.

Money Distribution Mechanism

The *Money Distribution* mechanism, ensures that the IoT devices will not run of money by frequently re-distributing money to them in a fair way, thus ensuring the existence of a competitive market. This distribution takes into account their CPU, memory and bandwidth resources as well as the respective resources of the Gateway. Assuming, an IoT Gateway with $\mathcal{C}$, $\mathcal{G}$ and $\mathcal{B}$ available resources and N IoT devices connected to it, the device i will receive extra money according to Eq. 5.3.

$$Money_i = Money_i + \frac{\mathcal{C} + \mathcal{M} + \mathcal{B}}{N} + \frac{a}{C_i} + \frac{a}{M_i} + \frac{a}{B_i} \tag{5.3}$$

where a is a constant equal to 10000 and has been defined according to multiple simulations of the system. This mechanism is activated every time the total money of all N devices connected to the Gateway reach a minimum threshold. This threshold is dynamic in the sense that it takes N into account, i.e. varying number of connected devices. In this way, it is guaranteed that no device will run out of money and thus will be able to compete for resources. This is crucial for a system, where there is no means for the IoT nodes to earn more money. In an actual price-based Edge computing system, this mechanism would not be necessary as each customer would have the obligation to renew its money reserve.

Task Selection Mechanism

Algorithm 8: DMRM functionality on the Gateway

```
   Data: Gateway, IoT devices' Tuples
   Gateway-Algorithm(Gateway, NDevs, DevTuples):
1    volatile curRound /* Round updated at background */
2    totalMoney = 0 /* Initialize Money */
3    /* Determine Money re-distribution threshold */
4    threshold = determineT(NDevs)
5    while ActiveDevs(DevTuples) > 0 do
6        if totalMoney < threshold then
7            /* Invoke Money Distribution Mechanism */ totalMoney += moneyD(NDevs,
              DevTuples)
8        /* Wait for offers */
9        taskBids = waitForBids(NDevs, DevTuples)
10       /* Invoke Task Selection Mechanism */
11       sTasks = taskSelection(taskBids)
12       /* Reply to IoT nodes */
13       sendAnswers(sTasks, NDevs, DevTuples)
```

The Gateway receives all biddings from the IoT nodes and must decide which will be accepted via the *Task Selection* mechanism.

This mechanism, combines the profit maximization of the Gateway with the minimization of the delayed tasks. From Eq. 5.2 it follows that the lower the sensitivity is, the higher the bidding price is. Similar to that, the higher the bidding price is, the higher the profit gained by the Gateway is. As a result, if the tasks with higher bids are chosen to run first, the system is approaches an equilibrium between the maximization of profits of the Gateway and the minimization of delayed tasks of the IoT nodes.

Inside the *Task Selection* mechanism, after the offloading propositions are received by the Gateway, they are sorted according to their bids in descending order. Starting from the first task on the list, if the resources of the Gateway are enough to satisfy the requirements of the task, then it is selected for execution. In the opposite case, the task is rejected. This process is repeated for all the received tasks, and the outcome is transmitted back to the corresponding devices. If a task was chosen to be executed, then the IoT device is charged according to the bid made. Otherwise, if a task is rejected then the IoT device is not charged and it is a responsibility of the IoT node to decide its execution fate.

The aforementioned DMRM functionality is executed in the software stack of the Gateway and it is summarized in Algorithm 8. The algorithm is executed as long as there are active connected devices to Gateway.

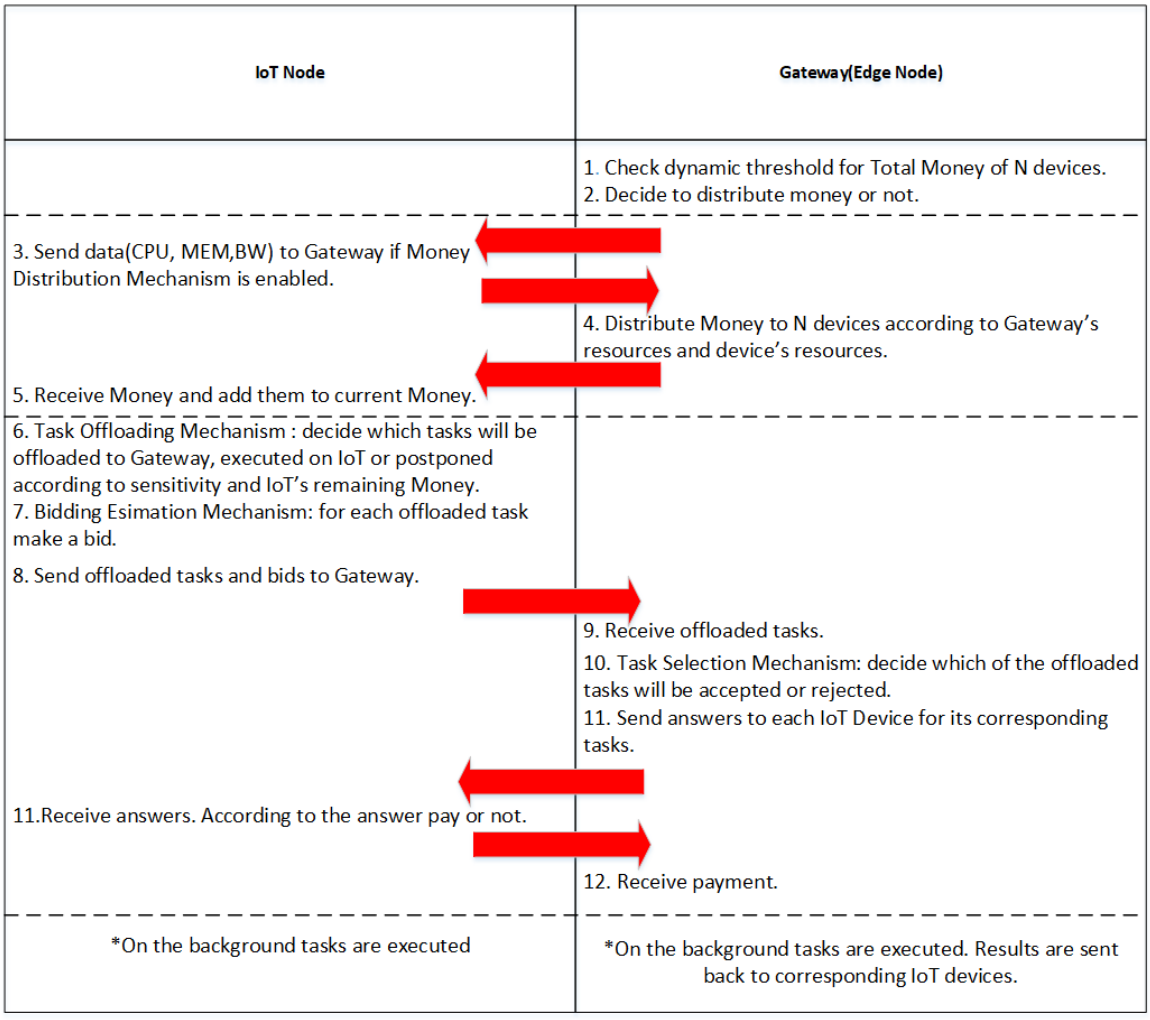


Figure 5.2: DMRM execution through time

The *Money Distribution* is executed conditionally according to the total amount of money

of the IoT nodes. The *Task Selection* function is executed when the Gateway has received new bids by the IoT nodes and then the nodes are informed accordingly.

Figure 5.2 presents how DMRM is executed through time on both IoT Nodes and Gateway. According to the state of the network and devices will mechanisms are activated and decisions are made. At this point, it should be mentioned that each device, Gateway or IoT Node, executes jobs on the background. Therefore, decisions can be made on the same time.

Chapter 6

Experimental Results

In this chapter, we present the numerical results that illustrate the validity of the proposed approaches to implement the market-based algorithm for resource management on Edge Computing and IoT architectures. Firstly, the experimental setup and the specifications of the boards, on which the experiments were conducted, is presented. Afterwards, the extracted results are analyzed and compared among them. More specifically, the brute-force approach, the Oracle Prediction, the Simulated Annealing and the DMRM solution are evaluated. Finally, the results based on economic models are extracted.

6.1 Experimental Setup

All the techniques presented in chapters 4.2, 4.3, 4.4 and 5 are implemented in C language and executed on variety of contemporary embedded devices with increasing computational capabilities, which represent diverse design choices with respect to the specifications of the Gateway device. Specifically, the utilized devices are Intel Galileo Gen 1 at 400 MHz, 256 MB RAM, Raspberry pi 3 Model B with 4 Cortex-A53 CPUs at 1.2 GHz, 1 GB of RAM and Nvidia Tegra X1 with 4 ARM Cortex A-57 processors running at 1.9 GHz and 4 GB of RAM. The basic specifications of each of the aforementioned boards are presented on the next subsections.

6.1.1 Raspberry pi 3 Model B

The Raspberry Pi is a low-cost Linux and ARM-based computer on a small circuit board sponsored by the charitable Raspberry Pi Foundation in the UK. The basic specs of this board are:

- **Processor Chipset:** Broadcom BCM2837 64-bit quad-core processor
- **Processor Speed:** 1.2GHz
- **RAM:** 1GB
- **Storage:** MicroSD
- **GPU:** 4x Cortex-A53 1.2GHz
- **On-board network:** 10/100 Mbit/s Ethernet, 802.11n wireless, Bluetooth 4.1
- **Operating System:** Raspbian Stretch Lite

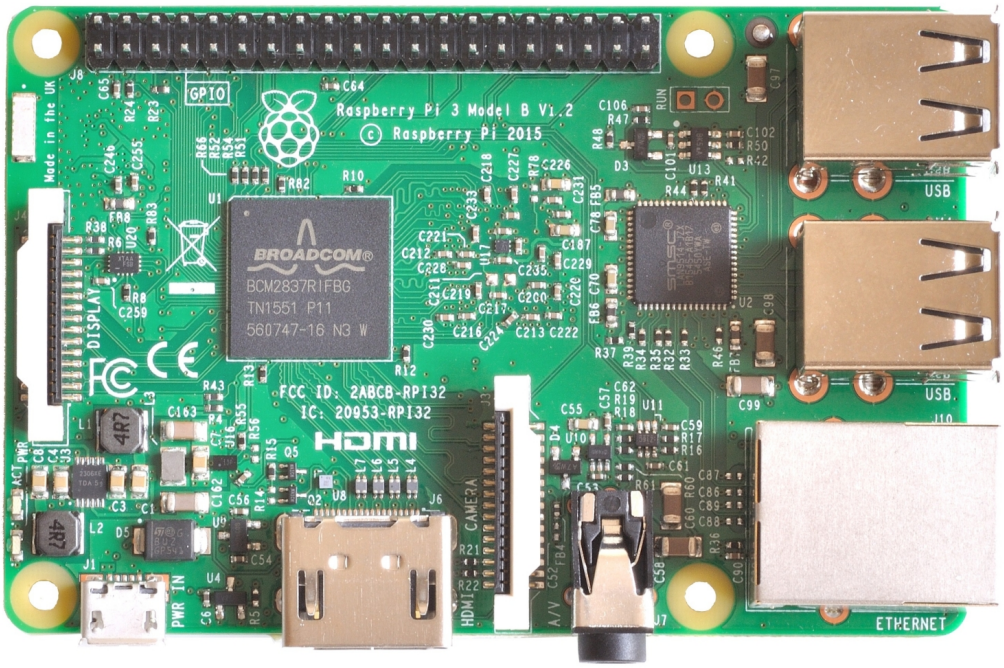


Figure 6.1: Raspberry pi 3 Model B [3]

6.1.2 Intel Galileo 1

Intel Galileo is the first in a line of Arduino-certified development boards based on Intel x86 architecture and is designed for the maker and education communities. Intel Galileo combines Intel technology with support for Arduino ready-made hardware expansion cards (called "shields") and the Arduino software development environment and libraries. The development board runs an open source Linux operating system with the Arduino software libraries, enabling re-use of existing software, called "sketches". The sketch runs every time the board is powered. Intel Galileo can be programmed through OS X, Microsoft Windows and Linux host operating software. The board is also designed to be hardware and software compatible with the Arduino shield ecosystem.

The Galileo is the first product to feature the Intel Quark SoC X1000, a chip designed for small-core products and low power consumption, and targeted at markets including the Internet of Things and wearable computing. The Quark SoC X1000 is a 32-bit, single core, single-thread, Pentium (P54C/i586) instruction set architecture (ISA)-compatible CPU, operating at speeds up to 400 MHz. The use of the Pentium architecture gives the Galileo the ability to run a fully-fledged Linux kernel. What's more, an on-board Ethernet port provides network connectivity, while also the underside provides a mini-PCI Express slot, designed for use with Intel's wireless network cards to add Wi-Fi connectivity to designs.

The Galileo board's technical specifications:

- **Operating System:** Yocto Project-based Linux
- **Processor:** Single-Core 400MHz Intel Quark X1000
- **Memory:** 256MB RAM
- **Networking:** 1x Wired 10/100 Ethernet, Optional PCIe Wireless

The Quark X1000 features:

- Up to 400MHz clock speed

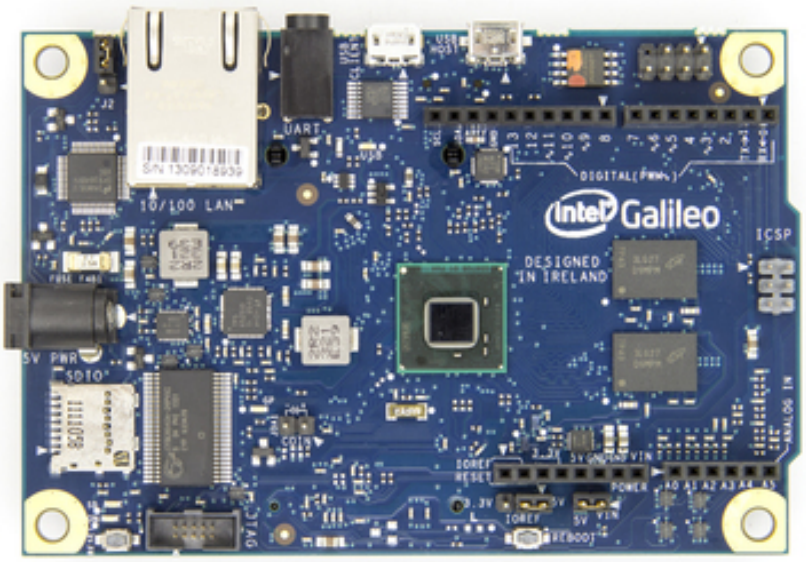


Figure 6.2: Intel Galileo Gen 1 [4]

- 16KB L1 Cache
- 512KB SRAM
- Single core, single thread
- Integraded SDIO, UART, SPI, USB, I2C, Ethernet, RTC

The CentOS release 6.9 operating system is applied on Tegra X1 for this study.

6.1.3 Tegra X1

Tegra is a SoC developed by NVIDIA and integrates an ARM architecture central processing unit (CPU), graphics processing unit (GPU)-sharing a common DRAM memory with CPU-, northbridge, southbridge, and memory controller onto one package. More specifically, Nvidia's Tegra X1 (codenamed "Erista") features four ARM Cortex-A57 cores and four ARM Cortex-A53 cores (not to be accessed by the operating system and are used automatically in very low power scenarios), as well as a Maxwell-based graphics processing unit. Nvidia's Tegra X1 is composed by:

- ARMv8 ARM Cortex-A57 quad-core + ARM Cortex-A53 quad-core (64-bit) at 1.9GHz
- Maxwell-based 256 core GPU
- MPEG-4 HEVC and VP9 encoding/decoding support
- 4GB of RAM
- TSMC 20 nm process
- TDP 15 watts, with average power consumption less than 10 watts

Tegra X1 is NVIDIA's newest mobile processor, and includes NVIDIA's highest performing, and power efficient Maxwell GPU architecture.

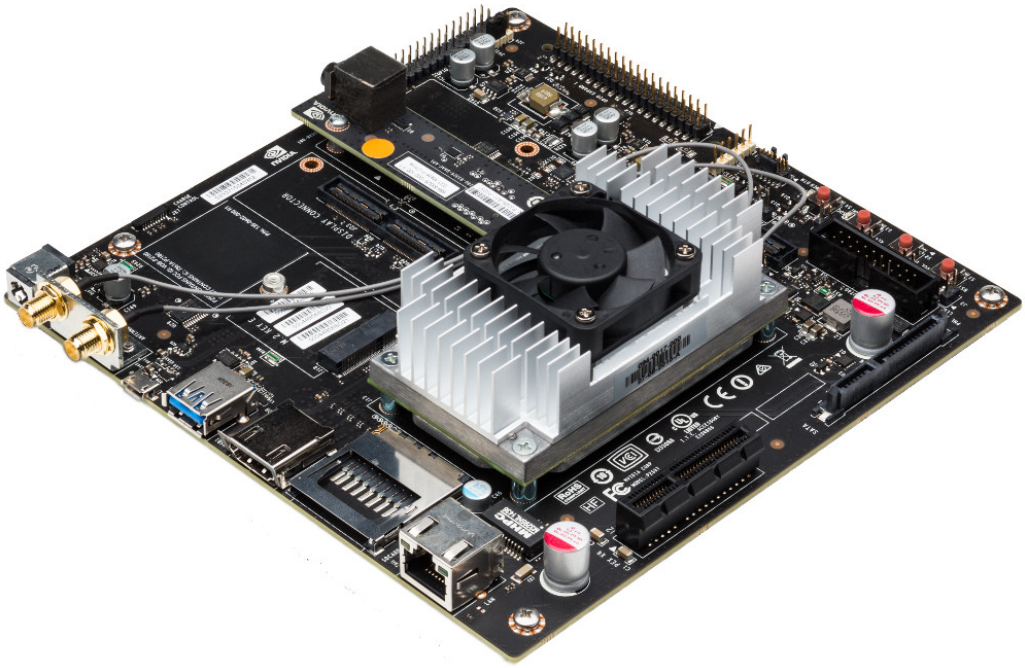


Figure 6.3: Nvidia's Tegra X1 [5]

6.2 Results Comparison and Evaluation

In this section, the results of the implemented approaches are presented. Each solution is analyzed and evaluated and, afterwards, they are compared among them.

6.2.1 IoT Devices and Tasks Inputs

We consider a variety of scenarios where the IoT devices and the tasks are randomly selected. The resources of the devices and the required resources of the tasks are randomly sampled from Normal or Poisson distributions in the range of $[0 - 100]$. In this way, we are able to construct diverse scenarios of CPU intensive, memory intensive and bandwidth intensive tasks and devices.

A second important experimental parameter is the arrival time and density of incoming tasks at IoT nodes.

We focus our attention on two distinct cases. In the first one, created using the Normal distribution, tasks are arriving almost simultaneously, thus creating a huge demand for resources. The second scenario involves cases, where simultaneous high peaks of demand are avoided and Poisson distribution is selected for this goal. Last, assuming that the arrival time of task τ_j^i is Ar_j^i, we calculate its deadline according to Eq. 6.1.

$$d_j^i = C_a * \frac{(EGct_j^i + ELct_j^i + 2 \cdot Ar_j^i)}{2} \tag{6.1}$$

According to this function, the deadline of the task is correlated to its arrival time and its estimated execution latency both on the Gateway and locally. The actual deadline is scaled according to coefficient C_a, which in the context of this evaluation was set equal to 1.4.

The IoT devices and the tasks were for each experiment were generated with Python.

6.2.2 Comparative Study

We evaluate DMRM against an offloading decision mechanism based on Simulated annealing (SA), a well-known probabilistic metaheuristic, which has been already used for task schedul-

ing [13, 14] and application mapping [19]. SA is configured to solve the optimization problem presented in Chapter 4 and has been analyzed further in section 4.4. We also calculate the optimal solution of the problem using an exhaustive, brute-force approach, to quantify the quality of the solutions provided by the heuristic ones as shown in section 4.2. However, this exhaustive solution is executed only when new tasks arrive. Consequently, we augment the comparison by adding an Oracle prediction mechanism as presented in 4.3, which knows a priori all the tasks that will arrive at the system. Therefore, the Oracle scheduler can lead to less deadline misses by postponing the execution of a task in order to save resources for a task arriving in later rounds and has stricter deadline.

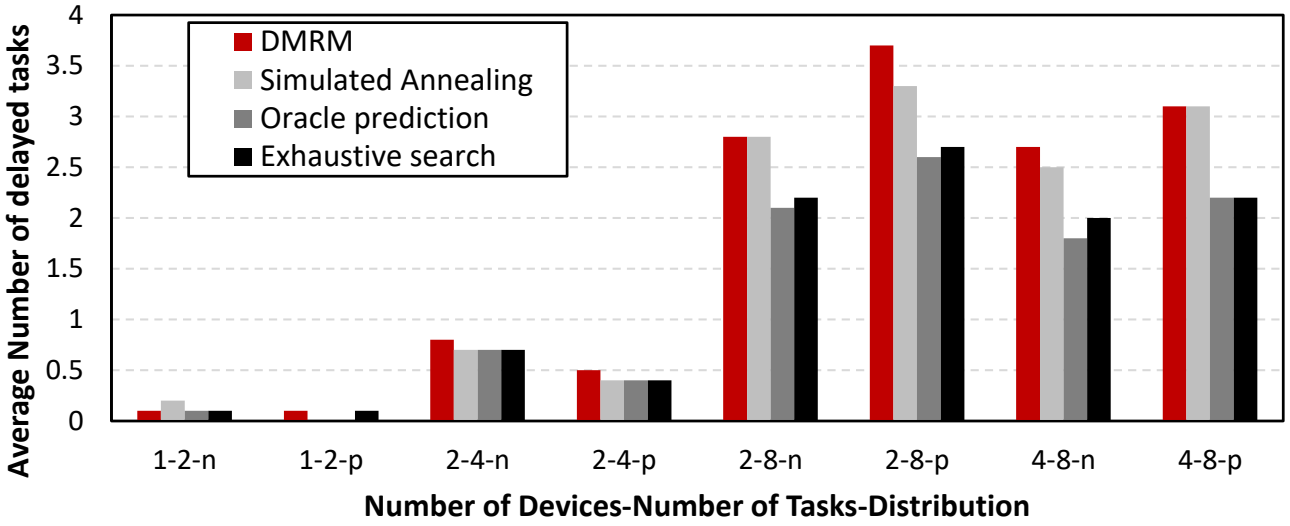


Figure 6.4: #Tasks with missed deadlines (Constrained input)

In the first comparative experiment, we evaluate DMRM against all other decision making alternatives. However, due to the exhaustive approaches, the experiments are conducted only for inputs with limited number of IoT tasks and devices, to avoid exploding the available scheduling combinations. The results of the comparison are summarized in Fig. 6.4, where the values of the X axis include triplets of the number of IoT devices of the examined scenario, the total number of tasks and the probability distribution function of tasks' arrival time, denoted as 'n' for Normal and 'p' for Poisson. The Y axis represents the number of delayed tasks. At this point, it should be mentioned that the exhaustive nature of both Oracle prediction and brute-force approach implies huge memory and time complexity. Furthermore, the Oracle prediction is an ideal approach, as all tasks are considered as known. In a real-life system this cannot happen. Obviously, these two methods cannot be executed for higher inputs and are only used for comparative study on small inputs.

As shown, both DMRM and SA algorithms achieve similar results, which are very close to the misses of the exhaustive experiments.

This first evaluation, shows that the proposed methodology provides results close to the optimal ones, but further more demanding experiments with scaled number of IoT devices and tasks are necessary. Fig. 6.5 presents the results of such an evaluation, maintaining the same format for X and Y axes. We observe that as the input workload of the system is higher, both in number and devices and tasks, DMRM outperforms the SA approach, achieving an average reduction of 3.22% in the number of delayed tasks.

Since both DMRM and SA are approximate solutions, we further evaluate their comparative properties in terms of the severity of their deadline misses. Specifically, the experiment presented in Fig. 6.6 shows the total amount of rounds that the delayed tasks exceeded their deadline, using the same input as the one of the experiments presented in Fig. 6.5. The results

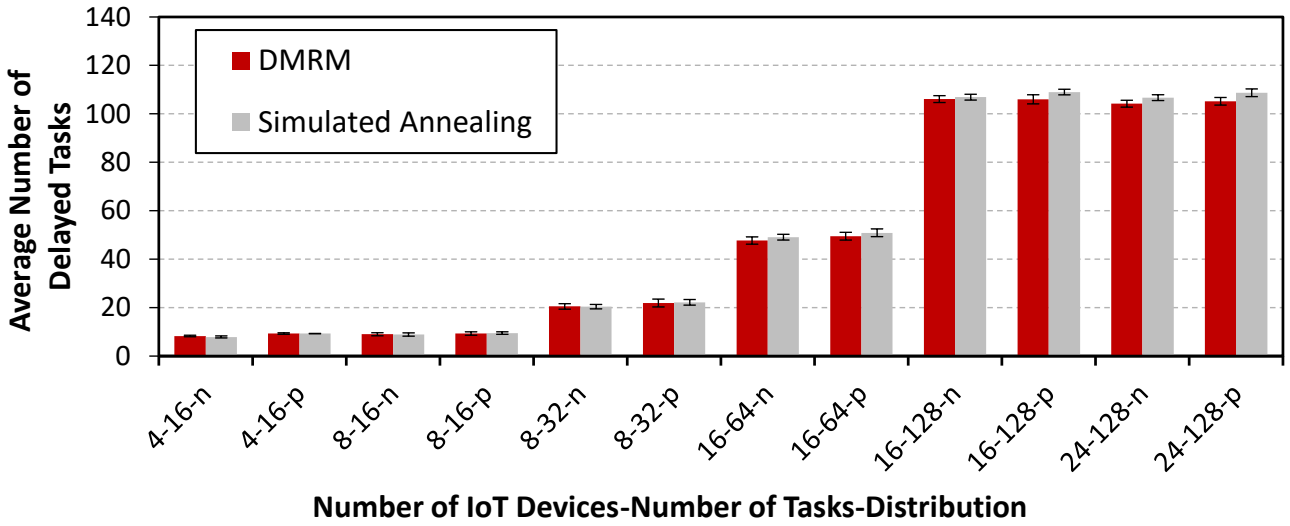


Figure 6.5: #Tasks with missed deadlines (Demanding input)

show, that DMRM also decreases this value up to 12.35 % compared to the SA approach.

As far as the simulated annealing is concerned, the probabilistic nature of the algorithm, cannot guarantee in every execution that optimal solutions will be found. More specifically, given the same input of tasks and IoT devices, the simulated annealing approach may extract different solutions. Additionally, for cases, in which there exist few optimal schedulings, this approach is less possible to reach optimal, or close to optimal, solutions.

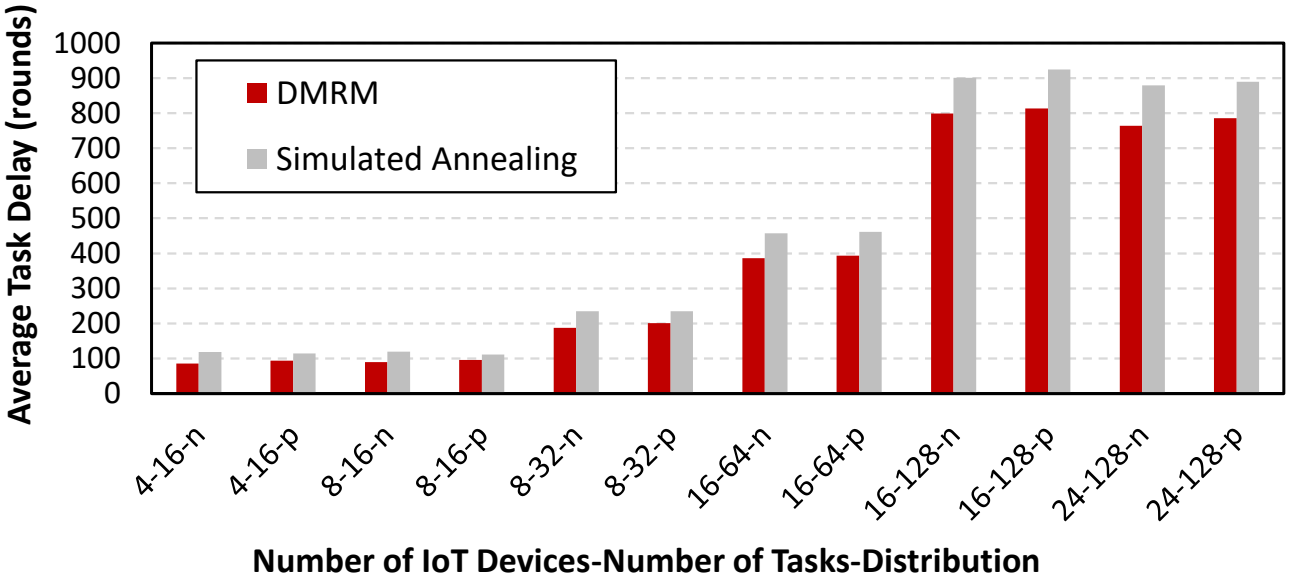
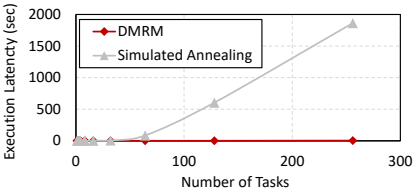


Figure 6.6: Total rounds of exceeded deadlines (Demanding input)

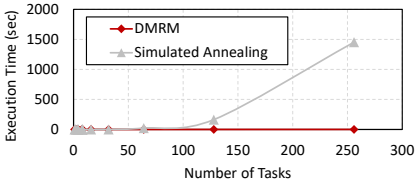
Last, a successful dynamic resource management scheme should also be characterised by the ability to make decisions in an online manner. Therefore, we perform a comparison of the execution latency of the two heuristic approaches on the embedded devices presented in Section 6.1.

Fig. 6.7 illustrates the measured latency values, showing that for low number of IoT tasks, both solutions are characterised by similar performance.

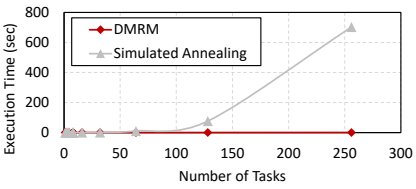
However, as the number of tasks rises, the execution latency of the SA approach skyrockets, while DMRM is insubstantially affected. This gap in the execution latency requirements is highlighted in the case of 256 tasks, where DMRM concludes 2000× faster. This behaviour is



(a) Execution latency on Intel Galileo



(b) Execution latency on Raspberry Pi 3



(c) Execution latency on Nvidia Tegra

Figure 6.7: Comparative performance evaluation of DMRM and SA based resource management on embedded Gateway alternatives

attributed to the inherent differences of the centralized nature of the SA approach as opposed to the scalable approach of the proposed market based solution, where the computational burden is distributed to the all involved devices of the system. In total, the execution overhead of DMRM is less than 1 second in the average case, while it only surpasses this in the least computationally powerful device (Intel Galileo), where at the worst case it reaches an overhead of 4 seconds.

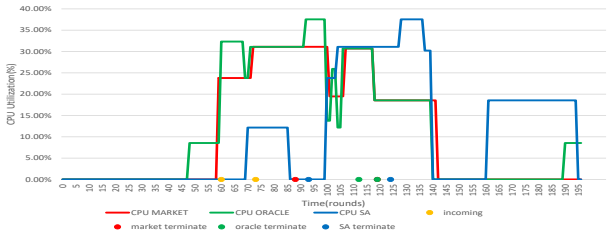


Figure 6.8: CPU for 2 tasks(Normal)

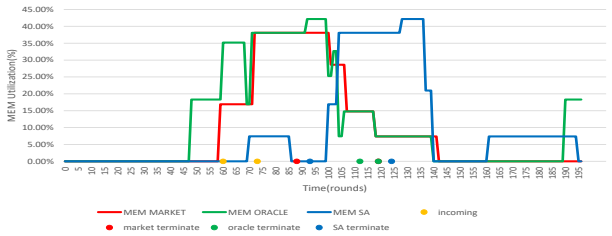


Figure 6.9: Memory for 2 tasks(Normal)

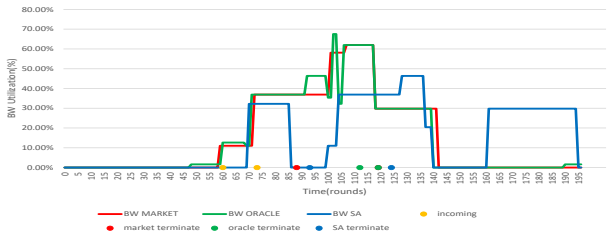


Figure 6.10: Bandwidth for 2 tasks(Normal)

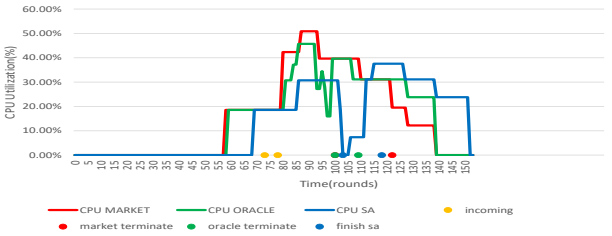


Figure 6.11: CPU for 2 tasks(Poisson)

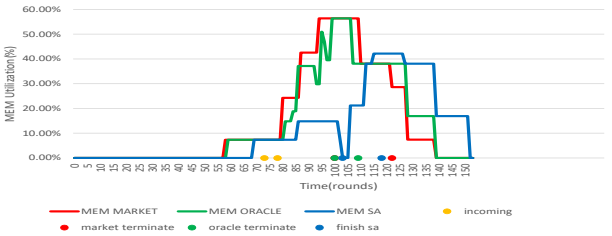


Figure 6.12: Memory for 2 tasks(Poisson)

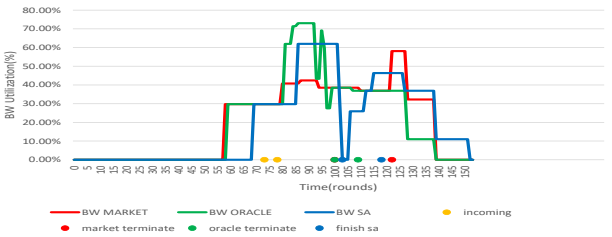


Figure 6.13: Bandwidth for 2 tasks(Poisson)

Last but not least, there are some experiments conducted, in order to evaluate the gateway's cpu, memory and bandwidth utilization and compare DMRM to other approaches. More specifically, charts in figures 6.8- 6.13 and 6.14- 6.19 illustrate the average CPU, memory and bandwidth utilization of Gateway for 2 and 8 tasks, respectively. Both experiments were executed for Normal and Poisson inputs. X axis denotes the time, measured in rounds, while Y axis describes the percentage of usage of the corresponding resource. The dots on X axis, illustrate the incoming and termination time of tasks for DMRM, Oracle prediction and Simulated Annealing approach.

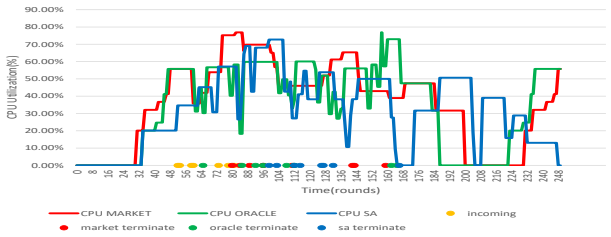


Figure 6.14: CPU for 8 tasks(Normal)

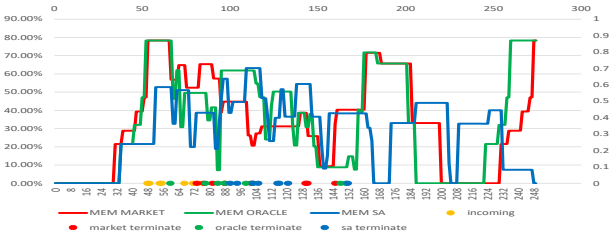


Figure 6.15: Memory for 8 tasks(Normal)

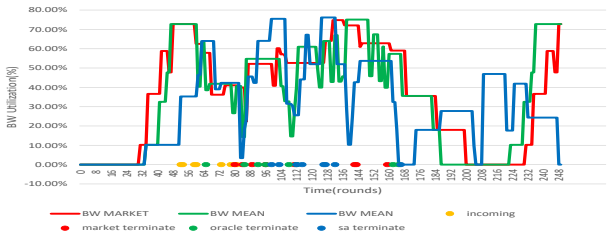


Figure 6.16: Bandwidth for 8 tasks(Normal)

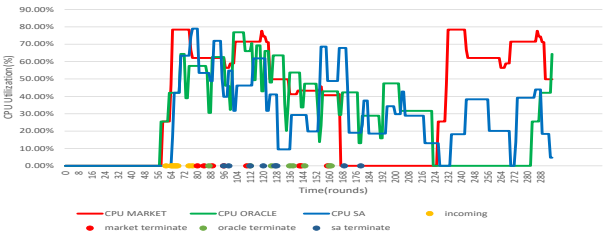


Figure 6.17: CPU for 8 tasks(Poisson)

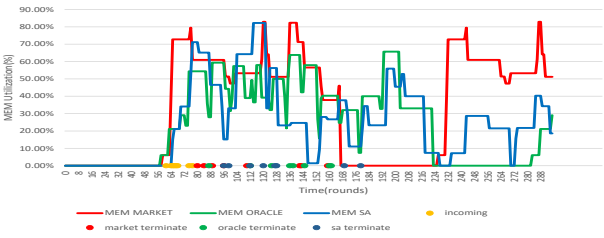


Figure 6.18: Memory for 8 tasks(Poisson)

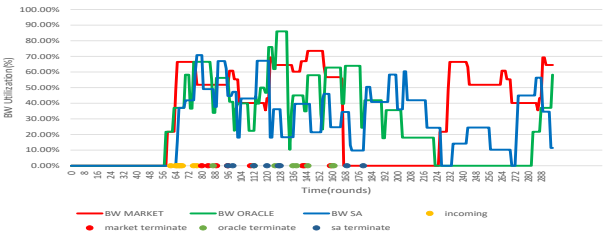


Figure 6.19: Bandwidth for 8 tasks(Poisson)

As shown on these graphs, the utilization of the resources of DMRM reaches almost 85% of the total resources of the Gateway. This means that, the IoT devices of the system try to offload their tasks, firstly, on the Gateway. Therefore, they can save battery and energy. Additionally, we can extract the fact that the utilization of DMRM is similar to the Oracle Prediction, while Simulated Annealing seems that it does not take advantage of all the resources of the Gateway.

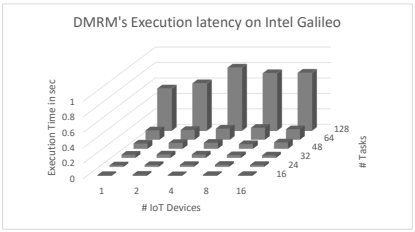
6.3 DMRM Special Results

In this section we present several characteristics of DMRM, in order to illustrate the validity of this approach.

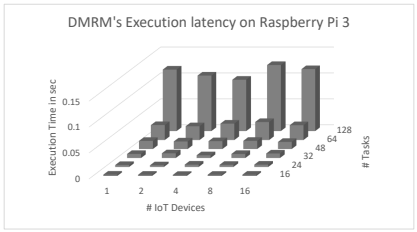
Firstly, an observation is extracted as far as the execution time of DMRM is concerned. More specifically, the examined testcases on the embedded devices showed that the average execution latency of the DMRM approach basically depends on the total number of the tasks. Figure 23illustrates the average execution time of DMRM implementation depending on the number of IoT devices and the number of tasks.

We are able to extract that the execution latency of DMRM depends basically on the number of total tasks, while the number of IoT devices in the network seems that it does not influence. Therefore, DMRM can afford higher number of IoT devices and tasks without having serious time overhead. In other words, we claim that DMRM is scalable for more than 256 tasks and 16 IoT devices.

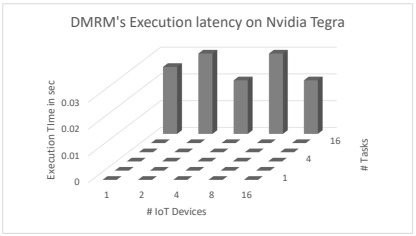
As far as the pricing models that were applied in DMRM, it is obvious that Smart Data Pricing and Consumer Perceived Value Pricing were used in order to define the selling price of the resources each time.



(a) Execution latency of DMRM on Intel Galileo



(b) Execution latency of DMRM on Raspberry Pi 3



(c) Execution latency of DMRM on Nvidia Tegra

Figure 6.20: Execution latency of DMRM on embedded devices according to number of IoT devices and tasks

6.4 DMRM Evaluation

Finally, to sum up all the aforementioned in previous sections, DMRM solution has several evaluations that can be extracted:

- **Optimality:** The proposed system reaches optimal and suboptimal solutions for low and high inputs, while on the same time, compared to SA reaches more optimized solutions. Additionally, DMRM reaches better results as fas as the delaytion per task is concerned.
- **Distributed:** The decision making is distributed among all IoT devices of the system.
- **Scalable:** DMRM is scalable for demanding number of IoT devices and tasks.
- **Low execution latency:** compared to other approaches, DMRM has low latency demands. Therefore, it could be applied in systems which require real-time execution and have latency-sensitive applications.
- **Adapt to dynamic changes of the network:** the proposed solution can easily adapt to dynamic changes of the network, such as IoT devices insertion and extraction.

Chapter 7

Conclusions

7.1 Thesis Summary

Resource management on Edge Computing systems and IoT architectures is a very demanding and challenging field, where both academic and scientific industries try to find efficient ways and solutions in order to tackle it.

In this diploma thesis, DMRM was presented, a novel resource management scheme for Edge Computing under CPU, memory, bandwidth and deadline constraints. A distributed solution based on economic and pricing models is proposed, implemented and evaluated on embedded devices. The decentralized nature of the algorithm distributes the computation burden among the devices and allows the system to behave efficiently and adapt in dynamic changes of the network. The experiments showed that the proposed solution, not only reaches results very close to optimal, but its distributed nature exhibits high scalability and tightly constrained execution latency.

7.2 Future work

The resource management in IoT and Edge Computing systems has a variety of open issues that need to be studied. At first, the same, or a similar problem could be solved using other pricing and market models, auctions and game theory. Other architectures can be implemented and studied. Additionally, the DMRM could be executed in real distributed systems, in order to examine how our approach responds and study parameters, such as fault tolerance, which do not show up in a simulation. Issues such as energy consumption need to be examined. Last but not least, issues that concern the security, the reliability and the confidentiality of data and users need to be addressed.

Bibliography

[1] https://philosophy.stackexchange.com/questions/10782/rational-irrationality-of-emotions-and-simulated-annealing.

[2] https://www.investopedia.com/university/economics/economics3.asp.

[3] https://raspi.tv/2016/raspberry-pi-3-model-b-launches-today-64-bit-quad-a53-1-2-ghz

[4] https://www.arduino.cc/en/ArduinoCertified/IntelGalileo.

[5] http://linuxgizmos.com/rugged-tegra-x1-module-stack-is-loaded-with-io-options/.

[6] https://en.wikipedia.org/wiki/Simulated_annealing.

[7] https://en.wikipedia.org/wiki/Edge_computing.

[8] https://en.wikipedia.org/wiki/Auction.

[9] https://www.investopedia.com/terms/s/sealed-bid-auction.asp.

[10] D. Chatzopoulos, M. Ahmadi, S. Kosta, and P. Hui. Have you asked your neighbors? a hidden market approach for device-to-device offloading. In *2016 IEEE 17th International Symposium on*, pages 1–9. IEEE, 2016.

[11] T. Ebi, D. Kramer, W. Karl, and J. Henkel. Economic learning for thermal-aware power budgeting in many-core architectures. In *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2011.

[12] P. Hu, S. Dhelim, H. Ning, and T. Qiu. Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of Network and Computer Applications*, 2017.

[13] D. E. Jeffcoat and R. L. Bulfin. Simulated annealing for resource-constrained scheduling. *European Journal of Operational Research*, 70(1):43–51, 1993.

[14] M. H. Kashani and M. Jahanshahi. Using simulated annealing for task scheduling in distributed systems. In *Computational Intelligence, Modelling and Simulation, 2009. CSSim'09. International Conference On*, pages 265–269. IEEE, 2009.

[15] M. A. Khan. A survey of computation offloading strategies for performance improvement of applications running on mobile devices. *Journal of Network and Computer Applications*, 56:28–40, 2015.

[16] L. Liu, Z. Chang, X. Guo, S. Mao, and T. Ristaniemi. Multiobjective optimization for computation offloading in fog computing. *IEEE Internet of Things Journal*, 5(1):283–294, 2018.

[17] 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*, 18(4):2546–2590, 2016.

[18] T. Melissaris, I. Anagnostopoulos, D. Soudris, and D. Reisis. Agora: Agent and market-based resource management for many-core systems. In *Electronics, Circuits and Systems (ICECS), 2016 IEEE International Conference on*, pages 400–403. IEEE, 2016.

[19] H. Orsila, T. Kangas, E. Salminen, T. D. Hämäläinen, and M. Hännikäinen. Automated memory-aware application distribution for multi-processor system-on-chips. *Journal of Systems Architecture*, 53(11):795–815, 2007.

[20] S. Paris, F. Martignon, I. Filippini, and L. Chen. An efficient auction-based mechanism for mobile data offloading. *IEEE Transactions on Mobile Computing*, 14(8):1573–1586, 2015.

[21] F. Samie, V. Tsoutsouras, S. Xydis, L. Bauer, D. Soudris, and J. Henkel. Distributed qos management for internet of things under resource constraints. In *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, page 9. ACM, 2016.

[22] T. Somu Muthukaruppan, A. Pathania, and T. Mitra. Price theory based power management for heterogeneous multi-cores. *ACM SIGPLAN Notices*, 49(4):161–176, 2014.

[23] F. M. F. Wong, C. Joe-Wong, S. Ha, Z. Liu, and M. Chiang. Improving user qoe for residential broadband: Adaptive traffic management at the network edge. In *Quality of Service (IWQoS), 2015 IEEE 23rd International Symposium on*, pages 105–114. IEEE, 2015.

[24] H. Zhang, F. Guo, H. Ji, and C. Zhu. Combinational auction-based service provider selection in mobile edge computing networks. *IEEE Access*, 5:13455–13464, 2017.