ΠΑΡΟΧΗ ΥΠΗΡΕΣΙΩΝ ΠΡΑΓΜΑΤΙΚΟΥ ΧΡΟΝΟΥ ΣΕ ΚΡΙΣΙΜΑ ΠΕΡΙΒΑΛΛΟΝΤΑ ΚΙΝΗΤΟΥ ΥΠΟΛΟΓΙΣΜΟΥ

Η

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Πληροφορικής
Εξεταστική Επιτροπή

από τον

Φίλιππο Παπαδόπουλο

ως μέρος των Υποχρεώσεων
για τη λήψη
του
ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ
ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΟ ΛΟΓΙΣΜΙΚΟ

Μάρτιος 2006

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να απευθύνω τις ευχαριστίες μου στον επιβλέποντα καθηγητή μου, κ. Απόστολο Ζάρρα για την πολύτιμη καθοδήγησή του καθ' όλη τη διάρκεια της παρούσας εργασίας. Η συνεργασία μαζί του ήταν ιδιαίτερα δημιουργική και ευχάριστη.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

ΠΕΡΙΛΗΨΗ

Φίλιππος Παπαδόπουλος του Γεωργίου και της Όλγας. MSc, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Μάρτιος 2006. Παροχή Υπηρεσιών Πραγματικού Χρόνου σε Κρίσιμα Περιβάλλοντα Κινητού Υπολογισμού. Επιβλέπων: Απόστολος Ζάρρας.

Ο στόχος αυτής της εργασίας είναι η επανεξέταση του ζητήματος της παροχής υπηρεσιών, που εξαρτώνται από χρονικούς περιορισμούς, στα πλαίσια ενός υπολογιστικού περιβάλλοντος ασύρματης δικτύωσης. Συγκεκριμένα προτείνουμε διαφορετικές πολιτικές που υλοποιούνται στο βασικό τμήμα μιας υπηρεσίας ενδιάμεσου λογισμικού που είναι ενσωματωμένη μέσα σε κάθε κινητή οντότητα του υπολογιστικού περιβάλλοντος. Οι προτεινόμενες πολιτικές λαμβάνουν υπόψη τους τις χρονικές απαιτήσεις των υπηρεσιών, σε συνάρτηση με την κινητικότητα των οντοτήτων που αποτελούν το υπολογιστικό περιβάλλον και την ύπαρξη πολλαπλών εναλλακτικών οντοτήτων, που παρέχουν σημασιολογικά συμβατές υπηρεσίες. Για να ικανοποιήσουμε τις ιδιαιτερότητες και τους περιορισμούς των οντοτήτων σε τέτοια περιβάλλοντα υιοθετούμε μια γενική έννοια για την μοντελοποίηση της συμπεριφοράς των κινητών οντοτήτων. Πιο συγκεκριμένα, υποθέτουμε ότι η γενικότερη συμπεριφορά των κινητών οντοτήτων διαμορφώνεται βάσει της διάρκειας ζωής (lifetime) των οντοτήτων. Η διάρκεια ζωής μιας οντότητας ορίζεται ως το χρονικό διάστημα κατά τη διάρκεια του οποίου η οντότητα είναι διαθέσιμη σε άλλες οντότητες.

EXTENDED ABSTRACT IN ENGLISH

Papadopoulos G. Filippos. MSc, Computer Science Department, University of Ioannina, Greece. March 2006. Provisioning of Real-time Services in Critical Mobile Computing Environments. Thesis Supervisor: Apostolos Zarras

Timeliness in conventional real-time systems is addressed by employing well-known scheduling techniques that guarantee the execution of a number of tasks within certain deadlines. However, these classical scheduling techniques do not take into account basic features that characterize todays mobile critical computing environments.

In this work, we revisit the issue of timeliness in the context of mobile computing environments. We propose a middleware service that addresses the timely provisioning of services, while taking into account both the mobility of the entities that constitute these environments and the existence of multiple alternative entities, providing semantically compatible services. To facilitate the timely provisioning of services in mobile environments we employ a generic notion for modeling the behavior of mobile entities. This notion shall serve as input to the scheduling techniques that we propose in this paper. Specifically, we assume that the overall behavior of mobile entities is modeled in terms of the entities' lifetime. The lifetime of an entity is defined as the time interval during which the entity is available to other entities. The lifetime is generic enough and can be approximated using a combination of different means, reflecting various characteristics such as the entity's physical motion and the entity's available resources.

Given a new request coming from a mobile client and a number of semantically compatible mobile entities that can fulfill the request, one of them must be selected. The existence of more than one alternative services also plays an important role towards the timely service provisioning in mobile environments. This sort of redundancy must be considered in a systematic way. Before issuing a service request to a mobile entity that shall serve it, the different alternatives must be evaluated so as to select the mobile entity that may possibly guarantee correct service provisioning. The proposed work realizes four different policies that facilitate the se-

lection. With respect to the first policy, the selection is realized solely on the basis of the client's and the server's lifetimes. The second and the third policies additionally consider the load of each server towards selecting the one that guarantees to serve the new request within the lifetime of both the client and the server. The fourth policy further deals with periodic service requests. It serves both periodic and aperiodic requests by using two classical real-time scheduling algorithms, EDF and TB.

ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ

Με την τρέχουσα τεχνολογική πρόοδο στους τομείς των φορητών υπολογιστών και των ασύρματων επικοινωνιών, γίνεται όλο και περισσότερο διαδεδομένη η χρήση των φορητών υπολογιστικών συσκευών που επικοινωνούν μέσω ασυρμάτων δικτύων, όπως φορητοί υπολογιστές, κινητά τηλέφωνα, προσωπικοί ψηφιακοί βοηθοί (personal digital assistants (PDAs)), κτλ. Οι υπολογιστικές δυνατότητες των παραπάνω συσκευών βελτιώνονται σημαντικά όσο και το μέγεθος τους, που γίνεται όλο και μικρότερο, με αποτέλεσμα να γίνονται σταδιακά αναπόσπαστο κομμάτι της καθημερινής μας ζωής. Οι συσκευές αυτές μπορούν να συνδέονται σε ασύρματα δίκτυα διαφορετικών τύπων χρησιμοποιώντας μια σειρά πρωτοκόλλων, όπως για παράδειγμα είναι το πρότυπο IEEE 802.11 [4]. Επίσης είναι διαθέσιμες ολοκληρωμένες πλατφόρμες ανάπτυξης λογισμικού για τις συγκεκριμένες συσκευές [21].

Συνεπώς, σταδιακά το όραμα της ασύρματης δικτύωσης κινητών κόμβων (mobile nodes) για την παροχή ολοκληρωμένων και αποδοτικών υπηρεσιών και εφαρμογών, γίνεται πραγματικότητα. Τελευταία μάλιστα, παρατηρείται μια προσπάθεια αλληλεπίδρασης των υπολογιστών με το φυσικό περιβάλλον που βρίσκονται, αποτελώντας στην ουσία μέρος αυτού. Η προσπάθεια αυτή παρουσιάστηκε και εδραιώθηκε με τον όρο *πανταχού παρών υπολογισμός* (*ubiquitous computing ή pervasive computing*). Ο πανταχού παρών υπολογισμός αποτελεί μία ιδέα που υποδηλώνει ότι καθώς η τεχνολογία προοδεύει οι υπολογιστές θα γίνονται μικρότεροι σε μέγεθος και περισσότερο ισχυροί, με αποτέλεσμα να μπορούν να ενσωματωθούν με αφανή τρόπο στο περιβάλλον παρέχοντας ένα πανταχού παρόν δίκτυο υπολογισμού και παροχής υπηρεσιών [5]. Οι κόμβοι σε ένα τέτοιο δίκτυο ποικίλουν από ετικέτες RFID με αισθητήρες, κινητά τηλέφωνα, PDA, φορητοί υπολογιστές μέχρι και υπερυπολογιστές με μεγάλες χωρητικότητες αποθήκευσης δεδομένων. Ο πανταχού παρών υπολογισμός περικλείει ένα με-

γάλο εύρος ερευνητικών πεδίων όπως τα κατανεμημένα συστήματα, τα ασύρματα δίκτυα, τα δίκτυα αισθητήρων και η τεχνητή νοημοσύνη.

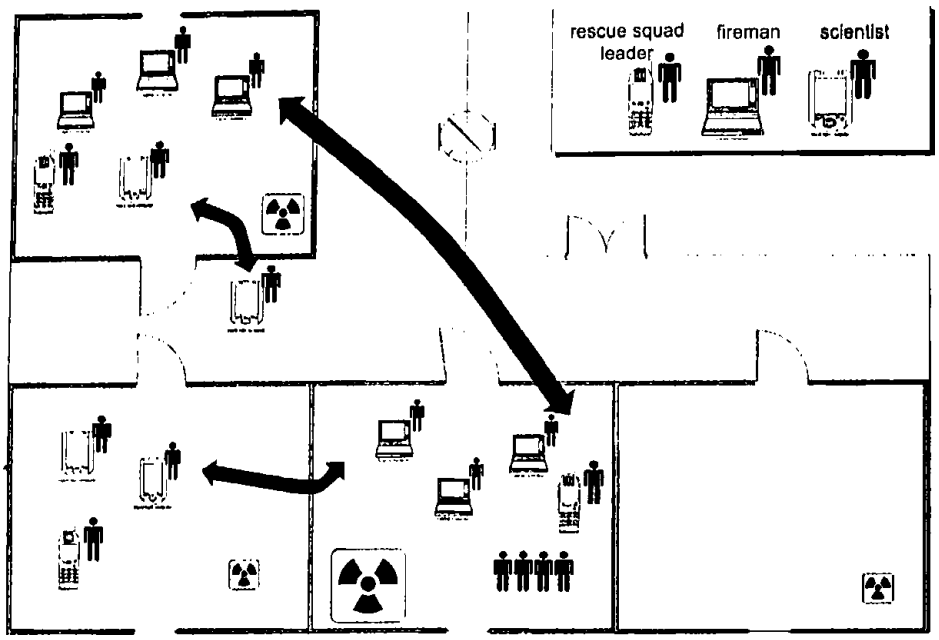
Η εξέλιξη των προαναφερθέντων τομέων κάνει πραγματικότητα την υλοποίηση και χρήση ενός πλήθους νέων εφαρμογών. Όσον αφορά συνήθεις και καθημερινές εφαρμογές μπορούμε να αναφέρουμε ότι σήμερα πολλές επιχειρήσεις έχοντας εγκαταστήσει την κατάλληλη υποδομή παρέχουν τις υπηρεσίες τους σε κινητούς χρήστες που είναι εξοπλισμένοι με φορητές συσκευές με ασύρματες δυνατότητες, οι οποίοι μπορούν να εισέρχονται και να αποχωρούν από το δίκτυο ανά πάσα στιγμή ανάλογα με τις προτιμήσεις τους. Επιπρόσθετα, πέρα από τις συνήθεις εφαρμογές, σε ένα περιβάλλον ασύρματης δικτύωσης μπορούν να παρέχονται υπηρεσίες για τον χειρισμό κρίσιμων καταστάσεων όπως ατυχήματα, φυσικές καταστροφές, κ.τ.λ. Αυτού του είδους οι εφαρμογές έχουν πιο εξειδικευμένες απαιτήσεις σε σχέση με τις συνήθεις εφαρμογές και συνεπώς επιβάλλουν περισσότερους περιορισμούς στον τρόπο αξιοποίησης των υπολογιστικών και δικτυακών πόρων των συσκευών που μετέχουν στο περιβάλλον.

1.1. Το Πρόβλημα

Σε ένα ασύρματο δίκτυο οι συνήθεις καθώς και οι κρίσιμες εφαρμογές χαρακτηρίζονται από τα παρακάτω βασικά χαρακτηριστικά:

- Αποτελούνται από ένα σύνολο κινητών οντοτήτων που παρέχουν έναν αριθμό υπηρεσιών τις οποίες μπορούν να χρησιμοποιήσουν άλλες κινητές οντότητες του δικτύου.
- Περισσότερες της μίας κινητής οντότητας μπορούν να παρέχουν σημασιολογικά ισοδύναμες υπηρεσίες.

Ας φανταστούμε ένα σενάριο εφαρμογής που είναι απαραίτητη για την αντιμετώπιση μιας κρίσιμης κατάστασης. Πιο συγκεκριμένα υποθέτουμε ότι έχουμε να κάνουμε με ένα ατύχημα σε ένα πυρηνικό εργοστάσιο [14]. Το εργοστάσιο αποτελείται από έναν αριθμό διαφορετικών εργαστηρίων όπως φαίνονται στο Σχήμα 1.1.



Σχήμα 1.1 Σύνοψη της Κατάστασης του Πυρηνικού Ατυχήματος.

Το ατύχημα προκάλεσε μια απότομη αύξηση στα επίπεδα της ραδιενέργειας στον χώρο του εργοστασίου. Οι ακριβείς μετρήσεις της ραδιενέργειας ποικίλουν από εργαστήριο σε εργαστήριο, ανάλογα με την φυσική θέση τους, π.χ οι μετρήσεις της ραδιενέργειας είναι μεγαλύτερες σε εργαστήρια που είναι πιο κοντά στην περιοχή που έγινε το ατύχημα. Το ατύχημα έλαβε χώρα τις πρωινές ώρες και έτσι ορισμένοι εργαζόμενοι εγκλωβίστηκαν μέσα σε διάφορα εργαστήρια. Για αυτόν τον λόγο ομάδες διάσωσης εισέρχονται στην περιοχή του ατυχήματος για να αντιμετωπίσουν αυτήν την κατάσταση. Κάθε ομάδα διάσωσης αναλαμβάνει διαφορετικά εργαστήρια και προσπαθεί να εντοπίσει τους εγκλωβισμένους εργαζόμενους.

Η κάθε ομάδα αποτελείται από πυροσβέστες που είναι εξοπλισμένοι με ασύρματους αισθητήρες ραδιενέργειας με περιορισμένες υπολογιστικές δυνατότητες. Η επικοινωνία μεταξύ των ομάδων είναι εφικτή μόνο μέσω των αρχηγών της κάθε ομάδας που επιπρόσθετα είναι εξοπλισμένοι με μικρούς φορητούς υπολογιστές που αποτελούν τα σημεία πρόσβασης για τα δίκτυα που δημιουργούνται σε κάθε δωμάτιο.

Επίσης, στον χώρο του ατυχήματος εισέρχεται και μια ομάδα από επιστήμονες. Ο σκοπός των επιστημόνων είναι να λάβουν διάφορες μετρήσεις στα επίπεδα της ραδιενέργειας ανάμεσα στα εργαστήρια και να τις χρησιμοποιήσουν για να αναλύσουν την κατάσταση και να βγάλουν συμπεράσματα για τον αντίκτυπο του ατυχήματος στην συνολική περιοχή του εργοστασίου. Κάθε επιστήμονας αναλαμβάνει από ένα ξεχωριστό εργαστήριο και είναι εξοπλισμένος με ένα PDA που χρησιμοποιείται για την επικοινωνία με τους αισθητήρες των πυροσβεστών.

Οι πυροσβέστες και οι επιστήμονες δεν μπορούν να μείνουν για μεγάλο χρονικό διάστημα στον χώρο του ατυχήματος. Κάθε ένας από αυτούς πρέπει να φύγει σε μια συγκεκριμένη αυστηρή χρονική στιγμή και μέχρι μια συγκεκριμένη χρονική *προθεσμία (deadline)* θα πρέπει να έχει ολοκληρώσει τις εργασίες που του έχουν ανατεθεί. Αν πάρουμε για παράδειγμα τους πυροσβέστες που δέχονται αιτήσεις για μετρήσεις της ραδιενέργειας από τους επιστήμονες, είναι σημαντικό να αποδεχτούν αυτές τις αιτήσεις μόνο εάν μπορούν να εξυπηρετηθούν μέσα στο χρονικό διάστημα που θα παραμείνουν τόσο αυτοί όσο και στο χρονικό διάστημα των επιστημόνων. Η αποτυχία εξυπηρέτησης αυτών των αιτήσεων μέσα στα χρονικά όρια θα καθυστερήσει και θα μειώσει την ακρίβεια της ανάλυσης του ατυχήματος, κάτι που είναι κρίσιμο για τον προσδιορισμό της συγκεκριμένης στρατηγικής που θα ακολουθηθεί για την άμεση αντιμετώπιση των συνεπειών του ατυχήματος.

Τα άτομα που αποχωρούν από την περιοχή του ατυχήματος μπορούν να αντικατασταθούν από νέα, ανάλογα με τις διαθέσιμες αναπληρώσεις. Οι πυροσβέστες μπορούν να μετακινηθούν από εργαστήριο σε εργαστήριο ανάλογα με τις περιστάσεις. Για παράδειγμα, μπορεί να ζητηθεί σε έναν πυροσβέστη από τον αρχηγό του να μετακινηθεί σε άλλο εργαστήριο όπου βρίσκονται τραυματισμένοι εργαζόμενοι. Τοπικά, ο αρχηγός μπορεί να αναθέτει διάφορα καθήκοντα στους πυροσβέστες του.

Σε αυτό το παράδειγμα τα μέλη της ομάδας διάσωσης και οι επιστήμονες είναι κινητές οντότητες. Συγκεκριμένα οι πυροσβέστες παρέχουν υπηρεσίες που χρησιμοποιούνται από τους επιστήμονες και τους αρχηγούς των ομάδων, που αποτελούν τους κινητούς πελάτες στην κρίσιμη αυτή κατάσταση.

Όπως φάνηκε και στο παραπάνω παράδειγμα, οι κρίσιμες εφαρμογές σε ένα υπολογιστικό περιβάλλον ασύρματης δικτύωσης πολλές φορές *εξαρτώνται από χρονικές παραμέτρους όπως είναι ο χρόνος εκτέλεσης τους και η προθεσμία μέσα στην οποία πρέπει να έχει ολοκληρωθεί μία υπηρεσία*. Σε ένα τέτοιο υπολογιστικό περιβάλλον ασύρματης δικτύωσης οι κινητές οντότητες χωρίζονται σε οντότητες που παρέχουν υπηρεσίες και σε οντότητες που χρησιμοποιούν υπηρεσίες, δηλαδή υπάρχουν οντότητες που παίζουν τον ρόλο του *πελάτη (client)* και οντότητες που παίζουν τον ρόλο του *εξυπηρετητή (server)*. Η κάθε αίτηση για εξυπηρέτηση από μία οντότητα-πελάτη σε μία οντότητα-εξυπηρετητή χαρακτηρίζεται από μια συγκεκριμένη προθεσμία που πρέπει να τηρηθεί από τον εξυπηρετητή. Όπως έχουμε προαναφέρει περισσότερες της μίας κινητής οντότητας-εξυπηρετητή μπορούν να παρέχουν σημασιολογικά ισοδύναμες υπηρεσίες. Τέλος, η κάθε οντότητα πελάτη ή εξυπηρετητή, έχει πιθανότατα περιορισμένο χρόνο στον οποίο θα βρίσκεται μέσα στο δίκτυο. Άρα προκύπτει το πρόβλημα του ποια οντότητα από όλες θα πρέπει να επιλεγεί από τις οντότητες-πελάτες η οποία θα εκ-

πληρώσει μια αίτηση μέσα στον χρόνο ζωής και των δύο.

Στα τυπικά *συστήματα πραγματικού χρόνου (real-time systems)* για την τήρηση των προθεσμιών και άλλων χρονικών περιορισμών χρησιμοποιούνται γνωστοί αλγόριθμοι χρονοπρογραμματισμού όπως είναι ο *EDF (earliest deadline first)* και ο *RM (rate-monotonic)* [9]. Αυτοί οι αλγόριθμοι εγγυώνται ότι, εφόσον είναι δυνατόν, η εκτέλεση ενός συνόλου από *εργασίες (tasks)* θα ολοκληρωθεί μέσα στο χρονικό διάστημα που καθορίζουν οι προθεσμίες της κάθε εργασίας. Παρόλα αυτά οι κλασικοί αυτοί αλγόριθμοι χρονοπρογραμματισμού δεν λαμβάνουν υπόψη βασικά χαρακτηριστικά και ιδιαιτερότητες ενός υπολογιστικού περιβάλλοντος ασύρματης δικτύωσης όπως το ότι οι κινητές οντότητες που απαρτίζουν το υπολογιστικό περιβάλλον μπορεί να έχουν ιδιαίτερα χαρακτηριστικά, όπως για παράδειγμα περιορισμένες υπολογιστικές δυνατότητες ή να λειτουργούν με μπαταρία, ότι οι κινητές οντότητες μπορούν να εισέρχονται και να αποχωρούν από το δίκτυο και ότι πολλές κινητές οντότητες μπορούν να παρέχουν τις ίδιες υπηρεσίες.

1.2. Στόχοι

Λαμβάνοντας υπόψη τα παραπάνω, ο στόχος αυτής της εργασίας είναι η επανεξέταση του ζητήματος της παροχής υπηρεσιών, που εξαρτώνται από χρονικούς περιορισμούς, στα πλαίσια ενός υπολογιστικού περιβάλλοντος ασύρματης δικτύωσης. Συγκεκριμένα προτείνουμε διαφορετικές πολιτικές που υλοποιούνται στο βασικό τμήμα μιας υπηρεσίας ενδιάμεσου λογισμικού που είναι ενσωματωμένη μέσα σε κάθε κινητή οντότητα. Οι προτεινόμενες πολιτικές λαμβάνουν υπόψη τους τις χρονικές απαιτήσεις των υπηρεσιών, σε συνάρτηση με:

i. την κινητικότητα των οντοτήτων που αποτελούν το υπολογιστικό περιβάλλον και
ii. την ύπαρξη πολλαπλών εναλλακτικών οντοτήτων, που παρέχουν σημασιολογικά συμβατές υπηρεσίες.

Δοθείσας μιας νέας αίτησης που προέρχεται από μια κινητή οντότητα-πελάτη και ενός αριθμού σημασιολογικά συμβατών εξυπηρετητών που μπορούν να εξυπηρετήσουν την αίτηση, θα πρέπει να επιλεγεί ένας από τους εξυπηρετητές. *Ο βασικός στόχος της διαδικασίας επιλογής είναι να εγγυηθεί πως μια απάντηση θα αποσταλεί πίσω στην οντότητα-πελάτη μέσα στην διάρκεια ζωής της.*

Γενικά, η συμπεριφορά των κινητών οντοτήτων μπορεί να είναι σχετικά περίπλοκη και εξαρτάται από αρκετούς παράγοντες. Παραδείγματος χάριν, μέχρι τώρα, έχει υπάρξει σχετική μελέτη προς τον υπολογισμό της φυσικής κίνησης των κινητών οντοτήτων [12]. Μια οντότητα μπορεί να γίνει απρόσιτη με την κίνηση της σε περιοχές που δεν ανήκουν στο εύρος κάλυ-

ψης ενός ασύρματου δικτύου. Ένα άλλο σημαντικό χαρακτηριστικό γνώρισμα που διακρίνει τις κινητές οντότητες από τις βασικές δομικές μονάδες των συμβατικών κατανεμημένων συστημάτων είναι οι περιορισμένοι πόροι τους. Παραδείγματος χάριν, η περιορισμένη μπαταρία μιας κινητής οντότητας μπορεί να καταστήσει την οντότητα απρόσιτη σε μόνιμη ή προσωρινή βάση. Επιπλέον, η οντότητα μπορεί ρητά να θέσει εκτός λειτουργίας την ικανότητα επικοινωνίας ή υπολογισμού για την εξοικονόμηση ισχύος, ή λόγω της λήψης συγκεκριμένων διαταγών από κάποια εξωτερική πηγή.

Για να ικανοποιήσουμε τις ιδιαιτερότητες και τους περιορισμούς των οντοτήτων σε ασύρματα περιβάλλοντα υιοθετούμε μια γενική έννοια για την μοντελοποίηση της συμπεριφοράς των κινητών οντοτήτων. Αυτή η έννοια θα χρησιμεύσει σαν βάση στις τεχνικές χρονοπρογραμματισμού που προτείνονται σε αυτήν την διατριβή.

Πιο συγκεκριμένα, υποθέτουμε ότι η γενικότερη συμπεριφορά των κινητών οντοτήτων διαμορφώνεται βάσει της *διάρκειας ζωής (lifetime)* των οντοτήτων. *Η διάρκεια ζωής μιας οντότητας ορίζεται ως το χρονικό διάστημα κατά τη διάρκεια του οποίου η οντότητα είναι διαθέσιμη σε άλλες οντότητες*. Η διάρκεια ζωής είναι ένας αρκετά γενικός όρος και μπορεί να αποτιμηθεί χρησιμοποιώντας έναν συνδυασμό διαφορετικών τρόπων, απεικονίζοντας διάφορα χαρακτηριστικά όπως η φυσική κίνηση της οντότητας και οι διαθέσιμοι πόροι της. Η αποτίμηση της διάρκειας ζωής των οντοτήτων είναι διαφανής στις τεχνικές χρονοπρογραμματισμού, που προτείνονται στην παρούσα εργασία, και αποτελεί ευθύνη των οντοτήτων καθώς εξαρτάται από τις ιδιομορφίες τους και δεν μπορεί να είναι μέρος μιας υποδομής ενδιάμεσου λογισμικού που πρόκειται να χρησιμοποιηθεί για την υλοποίηση των παρεχόμενων υπηρεσιών.

Είναι σημαντικό να σημειωθεί ότι θεωρούμε τη διάρκεια ζωής των κινητών οντοτήτων ως το συμβόλαιο μεταξύ των οντοτήτων και των τεχνικών χρονοπρογραμματισμού. Λόγω του γεγονότος ότι η διάρκεια ζωής αποτελεί βασική παράμετρο των τεχνικών χρονοπρογραμματισμού, θα πρέπει να τηρείται από τις κινητές οντότητες (π.χ οι οντότητες δεν θα πρέπει να καθιστούνται μη διαθέσιμες πριν τη δεδηλωμένη διάρκεια ζωής τους). Αυτή η απαίτηση εξ αρχής μπορεί να φαίνεται αρκετά περιοριστική για μια οποιαδήποτε ομάδα κινητών οντοτήτων που είναι διασυνδεδεμένη με ad-hoc δικτύωση. Όμως η προτεινόμενη προσέγγιση στοχεύει σε ομάδες κινητών οντοτήτων που έχουν απαιτήσεις πραγματικού χρόνου και είναι φυσικό να υπάρχουν περιορισμοί στην γενικότερη συμπεριφορά των οντοτήτων για την ικανοποίηση των περιορισμών.

Η ύπαρξη περισσότερων από μίας εναλλακτικών υπηρεσιών παίζει σημαντικό ρόλο σε κρίσιμες εφαρμογές σε ένα περιβάλλον κινητών οντοτήτων. Προτού μια οντότητα-πελάτης

αποστείλει μια αίτηση προς μια οντότητα-εξυπηρετητή, θα πρέπει πρώτα να αποτιμηθούν οι παρεχόμενες εναλλακτικές υπηρεσίες για να γίνει η επιλογή της οντότητας που θα εγγυηθεί την σωστή εξυπηρέτηση της αίτησης.

Η πρώτη από τις προτεινόμενες πολιτικές λαμβάνει υπόψη της μόνο τις διάρκειες ζωής των οντοτήτων-πελατών και των οντοτήτων-εξυπηρετητών. Εγγυάται ότι η απάντηση σε μια αίτηση μιας οντότητας-πελάτη θα αποσταλεί πίσω στην οντότητα εφόσον όμως ο εξυπηρετητής καταφέρει να εξυπηρετήσει την συγκεκριμένη αίτηση του πελάτη. Η δεύτερη και η τρίτη πολιτική παρέχουν ισχυρότερες εγγυήσεις για την εκπλήρωση της επιλογής του εξυπηρετητή που μπορεί να εξυπηρετήσει την αίτηση μέσα στην διάρκεια ζωής εξίσου του πελάτη και του εξυπηρετητή. Αυτό πραγματοποιείται λαμβάνοντας υπόψη και το φόρτο των διαθέσιμων εξυπηρετητών που παρέχουν τις υπηρεσίες. Η τέταρτη πολιτική χειρίζεται επιπρόσθετα και αιτήσεις για υπηρεσίες που εκτελούνται περιοδικά. Για τον χειρισμό υπηρεσιών που εκτελούνται περιοδικά, επεκτείνουμε συγκεκριμένους κλασικούς αλγορίθμους χρονοπρογραμματισμού συστημάτων πραγματικού χρόνου. Η επέκταση αυτή γίνεται για την κάλυψη των αναγκών που προκύπτουν σε ένα περιβάλλον κινητών οντοτήτων.

1.3. Δομή της Διατριβής

Η διατριβή περιέχει 7 κεφάλαια: Στο Κεφάλαιο 2 παρουσιάζεται το υπόβαθρο στο οποίο βασίζεται η εργασία μας το οποίο περιλαμβάνει το πεδίο του κινητού υπολογισμού και το πεδίο των συστημάτων πραγματικού χρόνου. Επίσης, στο ίδιο κεφάλαιο, παρουσιάζεται και η τυπική περιγραφή του προβλήματος. Το Κεφάλαιο 3 παρουσιάζει την γενική αρχιτεκτονική της προτεινόμενης υπηρεσίας ενδιάμεσου λογισμικού. Στο Κεφάλαιο 4 γίνεται ανάλυση των εναλλακτικών πολιτικών που προτείνουμε. Στο Κεφάλαιο 5 παρουσιάζονται αναλυτικά οι πειραματικές μετρήσεις της επίδοσης των πολιτικών, σε περιβάλλον προσομοίωσης. Στο Κεφάλαιο 6 παρουσιάζεται και αναλύεται η υλοποίηση μιας από τις πολιτικές, σε περιβάλλον Java. Τέλος, στο Κεφάλαιο 7 παρουσιάζονται τα συμπεράσματα της εργασίας μας.

ΚΕΦΑΛΑΙΟ 2. ΥΠΟΒΑΘΡΟ

Σε αυτό το κεφάλαιο παρουσιάζεται το υπόβαθρο στο οποίο βασίζεται η εργασία μας. Αρχικά, στην Ενότητα 2.1 θα αναφερθούμε και θα μελετήσουμε το πεδίο της *κινητικότητας (mobility)* των κόμβων σε ένα υπολογιστικό περιβάλλον και στην Ενότητα 2.2 θα μελετήσουμε το πεδίο των *συστημάτων πραγματικού χρόνου (real-time systems)*.

2.1. Κινητικότητα (Mobility)

Σε αυτήν την ενότητα θα αναφερθούμε και θα μελετήσουμε δικτυακά περιβάλλοντα που αποτελούνται από κινητούς κόμβους οι οποίοι επικοινωνούν μεταξύ τους. Αρχικά θα αναφερθούμε στο γενικότερο περιβάλλον του *κινητού υπολογισμού (mobile computing)*, που ουσιαστικά αποτελεί ένα κατανεμημένο σύστημα στο οποίο όμως συμμετέχουν κόμβοι που κινούνται και επικοινωνούν μεταξύ τους με ασύρματο τρόπο. Στην συνέχεια θα αναφερθούμε στο περιβάλλον MANET στο οποίο δεν υπάρχει σταθερή υποδομή και οι κινητοί κόμβοι δημιουργούν μεταξύ τους συστάδες κατά περίσταση. Τέλος αναφερόμαστε σε μια σύγχρονη προσπάθεια που γίνεται για την ενσωμάτωση των τεχνολογιών του κινητού υπολογισμού στο φυσικό περιβάλλον με τρόπο όμως που να μην γίνεται αντιληπτή η παρουσία τους στον χρήστη.

2.1.1. Κινητός Υπολογισμός (Mobile Computing)

Με την εμφάνιση και την συνεχή βελτίωση των φορητών υπολογιστών και των ασύρματων LAN άρχισε να γίνεται σταδιακά πραγματικότητα η ανάπτυξη κατανεμημένων συστημάτων που αποτελούνταν από κινητούς πελάτες, με αποτέλεσμα να δημιουργηθεί ένα νέο επιστημο-

νικό πεδίο, αυτό του *κινητού υπολογισμού (mobile computing)*. Παρόλο που πολλές βασικές αρχές των κατανεμημένων συστημάτων συνεχίζουν να ισχύουν και στα περιβάλλοντα κινητού υπολογισμού, τέσσερις σημαντικοί περιορισμοί, λόγω της *κινητικότητας (mobility)* των κόμβων του συστήματος, δημιούργησαν την ανάγκη για την ανάπτυξη νέων εξειδικευμένων τεχνικών. Οι περιορισμοί αυτοί είναι:

i. Απρόβλεπτες μεταβολές στην ποιότητα του δικτύου.
ii. Μεγαλύτερα προβλήματα ασφαλείας λόγω άμεσης πρόσβασης στο φυσικό μέσο (αέρας), με συνέπεια να υπάρχει μικρότερος βαθμός "εμπιστοσύνης" μεταξύ των κινητών κόμβων.
iii. Περιορισμένοι πόροι λόγω του βάρους και του μεγέθους των κόμβων.
iv. Περιορισμένος χρόνος λειτουργίας των συσκευών λόγω της χρησιμοποίησης μπαταρίας για την λειτουργία τους.

Οι νέες τεχνικές που αναπτύχθηκαν για την επίλυση των παραπάνω περιορισμών, μπορούν να κατηγοριοποιηθούν στις παρακάτω γενικές περιοχές [16]:

- Τεχνικές *κινητής δικτύωσης (mobile networking)*, συμπεριλαμβανόμενων του Mobile IP, των ad-hoc πρωτοκόλλων και των τεχνικών για την βελτίωση της απόδοσης του TCP σε ασύρματα δίκτυα.
- Τεχνικές για την *κινητή πρόσβαση πληροφοριών (mobile information access)*, που περιλαμβάνουν τεχνικές για τη *λειτουργία χωρίς σύνδεση (disconnected operation)* και τεχνικές για τον *επιλεκτικό έλεγχο της συνοχής των δεδομένων (selective control of data consistency)*.
- Τεχνικές υποστήριξης *προσαρμοστικών εφαρμογών (adaptative applications)* και τεχνικές *προσαρμοστικής διαχείρισης πόρων (adaptive resource management)*.
- Τεχνικές εξοικονόμησης ενέργειας, όπως είναι η *προσαρμοστικότητα βάσει της ενέργειας (energy-aware adaptation)* και η *διαχείριση της μνήμης ανάλογα με την ενέργεια (energy-sensitive memory management)*.
- Τεχνικές για την "αίσθηση" μιας τοποθεσίας (location sensitivity).

2.1.2. Περιβάλλοντα MANET

Σε ένα περιβάλλον MANET (*Mobile Ad hoc Networking*) το δίκτυο έχει δυναμική, τυχαία, τοπολογία που συνήθως αποτελείται από ασύρματους συνδέσμους (links) με περιορισμένο εύρος ζώνης (bandwidth). Ο στόχος της ασύρματης "ad hoc" δικτύωσης είναι να υποστηρίζει την κινητικότητα των αυτόνομων κόμβων που ουσιαστικά αποτελούν το "ad hoc" δίκτυο [3].

Ένα MANET αποτελείται από κινητούς κόμβους οι οποίοι είναι ελεύθεροι να μετακινούνται με οποιονδήποτε τρόπο. Οι κόμβοι μπορεί να βρίσκονται σε αεροπλάνα, πλοία, αυτοκίνητα, ακόμη και σε ανθρώπους που κουβαλάνε πολύ μικρές συσκευές. Συνεπώς ένα MANET είναι ένα αυτόνομο σύστημα κινητών κόμβων που μπορεί να λειτουργεί σε απομόνωση ή να συνδέεται και να επικοινωνεί μέσω μιας *πύλης (gateway)* με ένα σταθερό δίκτυο [11].

Οι κόμβοι σε ένα MANET είναι εξοπλισμένοι με ασύρματους πομπούς και δέκτες χρησιμοποιώντας κεραίες που μπορεί να εκπέμπουν προς όλες τις κατευθύνσεις (broadcast), ή προς συγκεκριμένη κατεύθυνση (από σημείο-σε-σημείο). Σε μια συγκεκριμένη χρονική στιγμή, θα υπάρχει μια ασύρματη σύνδεση ανάμεσα στους κόμβους με την μορφή ενός τυχαίου γράφου. Αυτή η ad hoc τοπολογία μπορεί να αλλάζει με τον χρόνο καθώς οι κόμβοι μετακινούνται. Συνεπώς ένα MANET έχει διάφορα σημαντικά χαρακτηριστικά:

i. Δυναμικές τοπολογίες: Οι κόμβοι μπορούν να κινούνται ελεύθερα και έτσι να μεταβάλλεται τυχαία η τοπολογία, που μπορεί να αποτελείται από συνδέσμους μονής και διπλής κατεύθυνσης.
ii. Σύνδεσμοι με περιορισμένο εύρος ζώνης και μεταβλητή χωρητικότητα: οι ασύρματοι σύνδεσμοι θα συνεχίσουν να έχουν σημαντικά χαμηλότερη χωρητικότητα συγκριτικά με τους ενσύρματους συνδέσμους. Επιπρόσθετα, λόγω της φύσης των ασύρματων επικοινωνιών, η πραγματική απόδοση των ασύρματων συνδέσμων είναι μικρότερη από όσο επιτρέπει ο μέγιστος ρυθμός εκπομπής (maximum transmission rate).
iii. Εξάρτηση από ενέργεια: Μερικοί ή όλοι οι κόμβοι σε ένα MANET μπορεί να βασίζονται σε μπαταρίες ή σε άλλα εξαντλητά μέσα για να καλύψουν τις ενεργειακές τους ανάγκες.

Οι εφαρμογές της τεχνολογίας MANET εκτείνονται σε ευρύ φάσμα πεδίων, από την βιομηχανία, το εμπόριο, μέχρι και την καθημερινή ζωή των ανθρώπων ιδιαίτερα σε τομείς που περιλαμβάνουν παροχή υπηρεσιών και ανταλλαγή δεδομένων που εξαρτώνται από τον χρόνο και την θέση. Επιπρόσθετα, η όλο και μεγαλύτερη ανάπτυξη και χρήση των φορητών συσκευών επιτάσσουν την χρήση της τεχνολογίας MANET.

2.1.3. Περιβάλλον Πανταχού Παρόντος Υπολογισμού (Pervasive Computing)

Ένα περιβάλλον πανταχού παρόντος υπολογισμού μπορεί να χαρακτηριστεί ως ένα περιβάλλον που είναι διαποτισμένο με υπολογιστικές δυνατότητες και δυνατότητες επικοινωνιών, αλλά με τέτοιο τρόπο έτσι ώστε να δίνεται η εντύπωση στους χρήστες, που βρίσκονται μέσα στο περιβάλλον, ότι η τεχνολογική υποδομή είναι αόρατη. Λόγω του ότι η κίνηση είναι ανα-

πόσπαστο κομμάτι της καθημερινής μας ζωής, η τεχνολογία του πανταχού παρόντος υπολογισμού πρέπει να υποστηρίζει την κινητικότητα, διαφορετικά ο χρήστης θα αντιλαμβάνεται την ύπαρξη της τεχνολογίας με την έλλειψή της, καθώς μετακινείται. Συνεπώς η τεχνολογία του πανταχού παρόντος υπολογισμού βασίζεται σε αυτήν του κινητού υπολογισμού, αλλά την επεκτείνει πολύ περισσότερο [16]. Πιο συγκεκριμένα, ενσωματώνει τέσσερις επιπρόσθετες ερευνητικές περιοχές που αφορούν στην:

i. Ανάπτυξη των *ευφυών χώρων (smart spaces)*: Ένας χώρος μπορεί να είναι μια περιορισμένη περιοχή όπως μια αίθουσα συνεδριάσεων ή ένας διάδρομος, ή μπορεί να είναι μια καλώς ορισμένη ανοιχτή περιοχή όπως ένα προαύλιο ή ένα οικοδομικό τετράγωνο. Με την ενσωμάτωση της υπολογιστικής υποδομής μέσα στην κτιριακή υποδομή, ένας ευφυής χώρος συνενώνει δύο ξεχωριστούς κόσμους. Η συνένωση αυτών των κόσμων επιτρέπει την αίσθηση και τον έλεγχο του ενός κόσμου από τον άλλον. Ένα απλό παράδειγμα είναι η αυτόματη ρύθμιση των επιπέδων κλιματισμού και φωτισμού ενός σπιτιού βάσει ενός ηλεκτρονικού προφίλ του κατοίκου του σπιτιού. Από την άλλη κατεύθυνση, το λογισμικό σε έναν φορητό υπολογιστή μπορεί να συμπεριφέρεται διαφορετικά ανάλογα με την τρέχουσα θέση του χρήστη. Η ευφυΐα μπορεί επίσης να εφαρμοστεί και σε ξεχωριστά αντικείμενα, ανεξάρτητα από το εάν αυτά βρίσκονται σε έναν ευφυή χώρο ή όχι.

ii. Παροχή *αφάνειας (invisibility)*. Η ιδεατή προσδοκία σε ένα τέτοιο περιβάλλον είναι η ολοκληρωτική εξαφάνιση της τεχνολογίας του pervasive computing όσον αφορά την συναίσθηση του χρήστη. Πρακτικά, μια λογική προσέγγιση σε αυτήν την ιδεατή προσδοκία είναι η επίτευξη της μηδαμινής περίσπασης της προσοχής του χρήστη. Εάν ένα περιβάλλον πανταχού παρόντος υπολογισμού ικανοποιεί διαρκώς τις προσδοκίες του χρήστη και δεν του προκαλεί συχνά εκπλήξεις, τότε του επιτρέπει να αλληλεπιδρά με το περιβάλλον σχεδόν υποσυνείδητα.

iii. Διαχείριση της *τοπικής κλιμάκωσης (localized scalability)*. Καθώς η ιδέα και η υλοποίηση των ευφυών χώρων μορφοποιείται όλο και περισσότερο, αντίστοιχα αυξάνονται και οι αλληλεπιδράσεις ανάμεσα στον χρήστη και στο υπολογιστικό περιβάλλον. Κάτι τέτοιο επιφέρει σημαντικές επιπτώσεις για έναν κινητό χρήστη όσον αφορά το εύρος ζώνης του ασύρματου δικτύου, την ενέργεια των συσκευών και την περίσπαση της προσοχής του χρήστη. Η παρουσία πολλών χρηστών στο περιβάλλον κάνει ακόμη πιο σύνθετο το πρόβλημα. Συνεπώς η κλιμάκωση αποτελεί ένα κρίσιμο πρόβλημα σε ένα περιβάλλον πανταχού παρόντος υπολογισμού. Σε ένα τυπικό κατανεμημένο σύστημα η έννοια της κλιμάκωσης τυπικά δεν λαμβάνει υπόψη της την φυσική απόστα-

ση. Για παράδειγμα, ένας διακομιστής ιστού ή ένας διακομιστής αρχείων θα πρέπει να μπορεί να εξυπηρετεί όσο περισσότερους πελάτες μπορεί, παρά το ότι μπορεί να βρίσκονται στο διπλανό δωμάτιο ή σε μια γειτονική χώρα. Η κατάσταση είναι πολύ διαφορετική σε ένα pervasive περιβάλλον. Εδώ, η πυκνότητα των αλληλεπιδράσεων πρέπει να ελαττώνεται καθώς ο χρήστης μετακινείται μακριά. Διαφορετικά, και ο χρήστης αλλά και το υπολογιστικό σύστημα θα αναγκάζονται να πραγματοποιούν μακρινές αλληλεπιδράσεις με μικρή ικανότητα άντλησης πληροφορίας και συνεπώς με μικρή καταλληλότητα. Το μεγαλύτερο μέρος των αλληλεπιδράσεων του χρήστη σε ένα pervasive περιβάλλον θα πραγματοποιείται τοπικά. Συνεπώς, σε ένα τέτοιο περιβάλλον η κλιμάκωση θα επιτυγχάνεται μειώνοντας δραστικά τις αλληλεπιδράσεις μεταξύ μακρινών οντοτήτων.

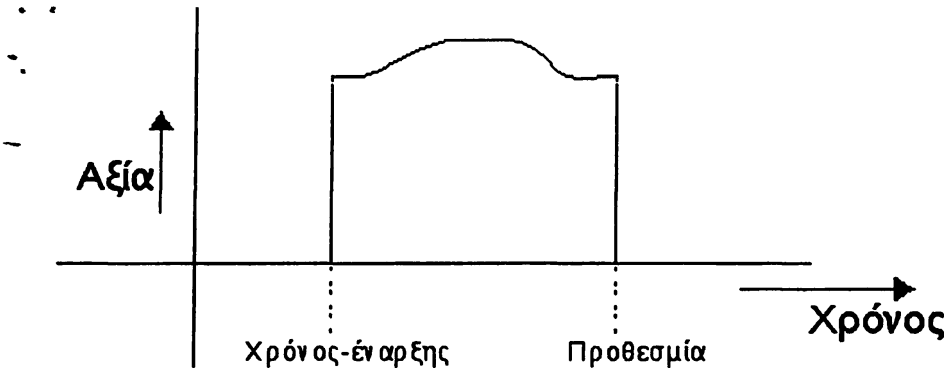
iv. Ανάπτυξη τεχνικών για την *απόκρυψη ανομοιογενών καταστάσεων του περιβάλλοντος (masking uneven conditioning of environments)*. Ο βαθμός διείσδυσης της τεχνολογίας του πανταχού παρόντος υπολογισμού μέσα στην υπόλοιπη υποδομή, εξαρτάται σημαντικά και από πολλούς μη-τεχνικούς παράγοντες όπως οικονομικούς, επιχειρηματικούς και οργανωτικούς. Σύμφωνα με την τρέχουσα κατάσταση, δεν έχει ακόμη επιτευχθεί ομοιόμορφη διείσδυση της συγκεκριμένης τεχνολογίας. Μέχρι να επιτευχθεί, θα υπάρχουν πολύ μεγάλες διαφορές στην ευφυΐα των διαφορετικών περιβάλλοντων. Ό,τι υποδομή μπορεί να είναι διαθέσιμη σε ένα καλώς εξοπλισμένο γραφείο ή σε μια αίθουσα συνεδριάσεων, μπορεί να έχει πολλές περισσότερες δυνατότητες σε σχέση με αντίστοιχη υποδομή σε άλλες τοποθεσίες. Αυτή η ανομοιομορφία στην ευφυΐα, μπορεί να είναι πολύ δυσάρεστη για έναν χρήστη διότι θα συντελεί αρνητικά στον απώτερο στόχο της αφάνειας ενός pervasive περιβάλλοντος. Ένας τρόπος για την μείωση της ανομοιογένειας, είναι η εξισορρόπησή της μέσω του προσωπικού υπολογιστικού συστήματος του χρήστη. Για παράδειγμα, ένα σύστημα, το οποίο προσφέρει αποσυνδεδεμένη λειτουργία, μπορεί να αποκρύψει την απουσία του ασύρματου δικτύου στο περιβάλλον που βρίσκεται.

2.2. Συστήματα Πραγματικού Χρόνου

Σε αυτήν την ενότητα θα αναφέρουμε έννοιες που συναντώνται ευρέως στην βιβλιογραφία των συστημάτων πραγματικού χρόνου.

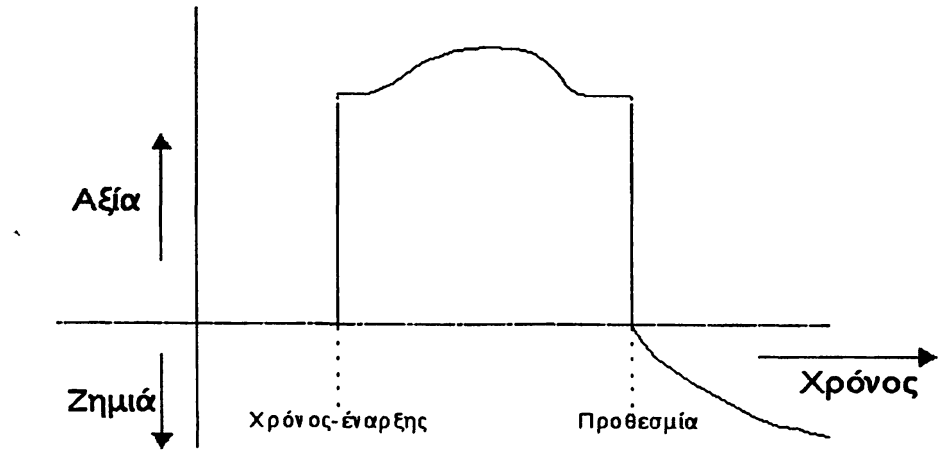
Ένα κοινό χαρακτηριστικό πολλών συστημάτων πραγματικού χρόνου είναι ότι οι προδιαγραφές των απαιτήσεών τους περιλαμβάνουν απαιτήσεις συγχρονισμού υπό την μορφή προ-

θεσμιών. Η έννοια μιας *αυστηρής προθεσμίας (hard deadline)* απεικονίζεται γραφικά στο Σχήμα 2.1. Ο χρόνος που απαιτείται για να ολοκληρωθεί ένα γεγονός αντιστοιχίζεται με την "αξία" που έχει αυτό το γεγονός στο σύστημα. Η έννοια της "αξίας", όπως φαίνεται στο σχήμα, καθορίζεται αόριστα και εξαρτάται από την σημασία που έχει το συγκεκριμένο γεγονός στο σύστημα. Για ένα υπολογιστικό γεγονός με αυστηρή προθεσμία, η αξία είναι μηδέν πριν από τον "χρόνο-έναρξης" και η τιμή της αξίας επιστρέφει στο μηδέν μόλις λήξει η προθεσμία.



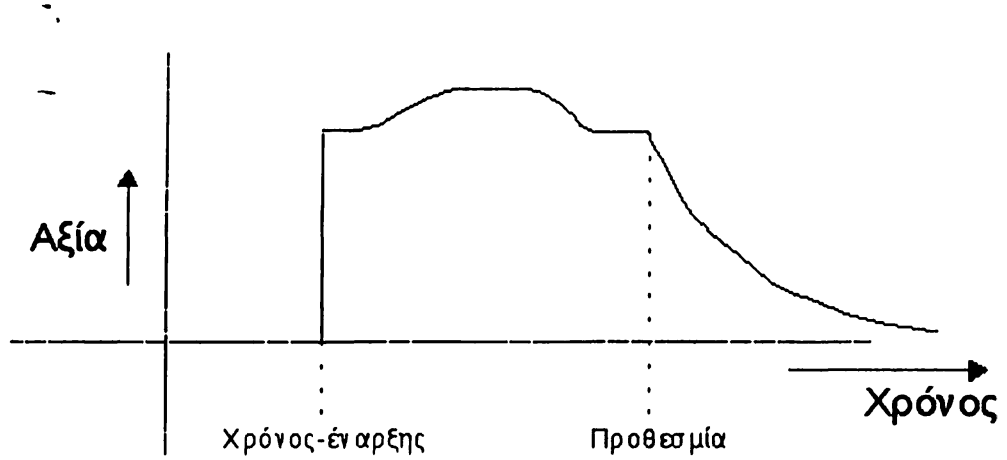
Σχήμα 2.1 Μια Αυστηρή Προθεσμία.

Σε ένα κρίσιμο σύστημα η κατάσταση μπορεί να είναι ακόμα χειρότερη, έτσι ώστε να έχουμε αρνητική αξία σχετιζόμενη με μια πρόωρη ή μια εκπρόθεσμη εκτέλεση ενός γεγονότος, όπως απεικονίζεται στο Σχήμα 2.2. Η αποτυχία γίνεται μεγαλύτερου βαθμού καθώς περνάει ο χρόνος μετά την λήξη της προθεσμίας.



Σχήμα 2.2 Ενα Κρίσιμο Σύστημα με Αρνητικές Συνέπειες.

Ένα σύστημα πραγματικού χρόνου ορίζεται ως *αυστηρό (hard real-time system)* εάν η ζημία που μπορεί να προκληθεί, από μια εκπρόθεσμη εκτέλεση, μπορεί να είναι καταστροφική για το σύστημα [1]. Στα περισσότερα συστήματα πραγματικού χρόνου δεν είναι όλες οι εκτελέσεις των διεργασιών κρίσιμες ή ισχυρά εξαρτώμενες από τον χρόνο. Μερικές δε, δεν σχετίζονται με κάποια προθεσμία και άλλες έχουν *χαλαρές προθεσμίες (soft deadlines)*. Μια χαλαρή προθεσμία, είναι μια προθεσμία που μπορεί να χαθεί χωρίς να παραβιαστεί η ακεραιότητα του συστήματος. Το Σχήμα 2.3 παρουσιάζει την έννοια μιας χαλαρής προθεσμίας. Όπως φαίνεται στο Σχήμα 2.3, παρόλο που η αξία πραγματοποίησης του γεγονότος μικραίνει μετά από την λήξη της προθεσμίας, είναι πάντα θετική (καθώς ο χρόνος πηγαίνει στο άπειρο).



Σχήμα 2.3 Μια Χαλαρή Προθεσμία.

Η διάκριση μεταξύ αυστηρών και χαλαρών προθεσμιών είναι χρήσιμη όταν αναφερόμαστε σε συστήματα πραγματικού χρόνου. Ένα αυστηρό σύστημα πραγματικού χρόνου απαιτείται για τον χειρισμό καταστάσεων ισχυρά εξαρτώμενων από το χρόνο, όπως είναι ο έλεγχος εναέριας κυκλοφορίας και ο χειρισμός οχημάτων. Για να μπορέσει ένα σύστημα πραγματικού χρόνου να εγγυηθεί την έγκαιρη ολοκλήρωση των λειτουργιών και των υπηρεσιών που προσφέρει, μέσα στο χρονικό διάστημα που καθορίζουν οι προθεσμίες, απαιτείται η υιοθέτηση τεχνικών και αλγορίθμων χρονοπρογραμματισμού. Πριν αναφερθούμε αναλυτικότερα σε συγκεκριμένες τεχνικές χρονοπρογραμματισμού, θα εξετάσουμε το μοντέλο των συστημάτων πραγματικού χρόνου και τα γενικότερα χαρακτηριστικά των τεχνικών.

Οι *διεργασίες* ή *εργασίες (processes, tasks)* αποτελούν τις θεμελιώδεις μονάδες υπολογισμού σε έναν επεξεργαστή. Σε ένα σύστημα πραγματικού χρόνου, οι χρονικοί περιορισμοί αντιστοιχίζονται σε προθεσμίες διεργασιών. Ανάλογα με την χρονική συμπεριφορά τους οι διεργασίες χωρίζονται σε δύο κατηγορίες:

i. *Περιοδικές* διεργασίες

ii. *Απεριοδικές* διεργασίες (ή μη-περιοδικές)

Οι περιοδικές διεργασίες εκτελούνται σε συνεχή βάση και χαρακτηρίζονται από

i. Την *περίοδο* τους.

ii. Τον απαιτούμενο *χρόνο εκτέλεσης (execution time)* ανά περίοδο.

Ο χρόνος εκτέλεσης είναι συνήθως ο μέσος χρόνος εκτέλεσης ή ο χρόνος εκτέλεσης της χειρότερης περίπτωσης. Για παράδειγμα μια περιοδική εργασία μπορεί να εκτελείται κάθε ένα δευτερόλεπτο, συνεπώς έχει περίοδο ίση με 1 sec, και να απαιτεί περίπου 100 ms χρόνου CPU στην γενική περίπτωση. Μπορεί να υπάρχουν όμως και ακραίες περιπτώσεις όπου απαιτεί 300 ms.

Όσον αφορά τις απεριοδικές εργασίες, μπορούν να ενεργοποιηθούν τυχαία και συνήθως αυτό προκύπτει ως συνέπεια ενός εξωτερικού για το σύστημα γεγονότος. Οι απεριοδικές εργασίες έχουν και αυτές χρονικούς περιορισμούς καθότι θα πρέπει να έχουν ολοκληρώσει την εκτέλεση τους μέσα σε ένα συγκεκριμένο χρονικό διάστημα. Γενικά, όπως αναφέρθηκε, οι απεριοδικές εργασίες θεωρούνται ότι ενεργοποιούνται τυχαία και δεν έχουν αυστηρές προθεσμίες. Στην περίπτωση που είναι γνωστή η ελάχιστη περίοδος ανάμεσα σε οποιαδήποτε δύο απεριοδικά γεγονότα (από την ίδια πηγή), τότε η αντίστοιχη διεργασία ονομάζεται *σποραδική* [2]. Ο όρος "σποραδική" χρησιμοποιείται συνήθως σε περιπτώσεις που απαιτούνται αυστηρές προθεσμίες.

Θεωρούμε ότι η προθεσμία μιας διεργασίας ορίζεται ως ο μέγιστος *χρόνος απόκρισης (response time)* της. Ανάμεσα στον χρόνο εκκίνησης της και στην προθεσμία της, η εργασία απαιτεί συγκεκριμένο *χρόνο εκτέλεσης (execution time)* για να ολοκληρώσει την εκτέλεση της. Ο χρόνος εκτέλεσης μπορεί να είναι σταθερός ή να κυμαίνεται μεταξύ μιας ελάχιστης και μιας μέγιστης τιμής. Ο χρόνος εκτέλεσης, στον οποίο δεν συμπεριλαμβάνεται ο συναγωνισμός μεταξύ των διεργασιών για την εκτέλεση τους, δεν θα πρέπει να είναι μεγαλύτερος από τον χρόνο ανάμεσα στην έναρξη της διεργασίας και της προθεσμίας της. Συνεπώς η ακόλουθη σχέση θα πρέπει να ισχύει για όλες τις διεργασίες:

$$C \leq D$$

όπου

- C είναι ο χρόνος εκτέλεσης
- D είναι η προθεσμία

Επιπρόσθετα, για κάθε περιοδική διεργασία η περίοδος της πρέπει να είναι τουλάχιστον ίση με την προθεσμία της. Αυτό σημαίνει ότι μια *αίτηση (request)* για την εκτέλεση της διεργασία πρέπει να ολοκληρωθεί προτού γίνουν επακόλουθες αιτήσεις για αυτήν. Αυτό ονομάζεται *περιορισμός εκτελεσιμότητας (runnability constraint)* [9]. Συνεπώς για τις περιοδικές διεργασίες ισχύει η ακόλουθη σχέση:

$$C \leq D \leq T$$

όπου

- Τ είναι η περίοδος της διεργασίας

Εργασίες των οποίων η πρόοδος δεν εξαρτάται από την πρόοδο των άλλων εργασιών ονομάζονται *ανεξάρτητες (independent)*. Αυτός ο ορισμός δεν λαμβάνει υπόψη ως εξάρτηση τον συναγωνισμό μεταξύ των εργασιών, για την απόκτηση του επεξεργαστή.

2.2.1. Χαρακτηριστικά των Χρονοπρογραμματιστών

Ένας *χρονοπρογραμματιστής (scheduler)* παρέχει έναν αλγόριθμο ή πολιτική για την ρύθμιση της εκτέλεσης των διεργασιών στον επεξεργαστή σύμφωνα με κάποια προκαθορισμένα κριτήρια. Σε ένα τυπικό πολυδιεργασιακό λειτουργικό σύστημα, οι διεργασίες εναλλάσσουν την εκτέλεση τους και οι διεργασίες με την μεγαλύτερη *προτεραιότητα (priority)* εκτελούνται πρώτες. Ελάχιστη ή καμία προσοχή δεν δίνεται στις προθεσμίες. Αυτό είναι σαφώς ανεπαρκές για τα συστήματα πραγματικού χρόνου, καθώς αυτά τα συστήματα απαιτούν πολιτικές χρονοπρογραμματισμού που λαμβάνουν υπόψη τους χρονικούς περιορισμούς των διεργασιών πραγματικού χρόνου.

Οι χρονοπρογραμματιστές παράγουν ένα *χρονοπρόγραμμα (schedule)* για ένα δεδομένο σύνολο διεργασιών. Εάν ένα σύνολο διεργασιών μπορεί να δρομολογηθεί έτσι ώστε να ικανοποιεί συγκεκριμένες προϋποθέσεις και συνθήκες, το σύνολο των διεργασιών ονομάζεται *εφικτό (feasible)*. Μια χαρακτηριστική προϋπόθεση για τις αυστηρές περιοδικές διεργασίες είναι ότι πρέπει πάντα να τηρούν τις προθεσμίες τους.

Ένας *βέλτιστος χρονοπρογραμματιστής (optimal scheduler)* είναι σε θέση να παράγει ένα εφικτό χρονοπρόγραμμα για όλα τα εφικτά σύνολα διεργασιών που υπακούουν σε μια δεδομένη προϋπόθεση. Για ένα συγκεκριμένο σύνολο διεργασιών ένα *βέλτιστο χρονοπρόγραμμα (optimal schedule)* είναι το καλύτερο δυνατό χρονοπρόγραμμα σύμφωνα με κάποια προκαθορισμένα κριτήρια. Γενικότερα, ένας χρονοπρογραμματιστής είναι βέλτιστος εάν παράγει πάντα ένα εφικτό χρονοπρόγραμμα, όταν βέβαια το σύνολο διεργασιών έχει εφικτά χρονοπρογράμματα. Αντιστρόφως, εάν ένας βέλτιστος χρονοπρογραμματιστής αποτύχει να βρει ένα εφικτό χρονοπρόγραμμα, μπορούμε να συμπεράνουμε ότι το δεδομένο σύνολο διεργασιών δεν μπορεί να δρομολογηθεί από κανέναν χρονοπρογραμματιστή [10].

2.2.2. Online και Offline Χρονοπρογραμματιστές

Οι αλγόριθμοι χρονοπρογραμματισμού μπορούν να κατηγοριοποιηθούν σε *offline* και σε *online*. Ένας χρονοπρογραμματιστής ορίζεται ως *offline*, εάν ο υπολογισμός του χρονοπρογράμματος γίνεται πριν από την εκτέλεση του συστήματος. Κάτι τέτοιο στηρίζεται εξ ολοκλήρου σε μια *εκ των προτέρων (a priori)* γνώση της συμπεριφοράς των διεργασιών. Για παράδειγμα πρέπει να είναι γνωστό εκ των προτέρων ποιες εργασίες θα καταφθάσουν στο σύστημα. Κάτι τέτοιο προκαλεί μικρή *επιβάρυνση στον χρόνο εκτέλεσης (run-time overhead)* του συστήματος. Αυτό το σχήμα είναι εφικτό μόνο όταν το σύστημα είναι ντετερμινιστικό, δηλ. ο αριθμός των εργασιών είναι σταθερός και οι χρόνοι εκτέλεσης και οι χρόνοι άφιξης είναι γνωστοί.

Ένας *online* χρονοπρογραμματιστής παίρνει τις αποφάσεις χρονοπρογραμματισμού κατά τη διάρκεια εκτέλεσης του συστήματος, δηλ. καθώς καταφθάνουν οι εργασίες. Συνεπώς δεν απαιτείται γνώση των χαρακτηριστικών των εργασιών που θα καταφθάσουν στο μέλλον. Οι χρονοπρογραμματιστές αυτής της κατηγορίας έχουν υψηλότερο κόστος χρόνου εκτέλεσης αλλά μπορούν να προσφέρουν μεγαλύτερη αξιοποίηση του επεξεργαστή.

Οι χρονοπρογραμματιστές επίσης μπορεί να είναι *προεκχωρητικοί (preemptive)* ή *μη-προεκχωρητικοί*. Οι προεκχωρητικοί μπορούν σε οποιαδήποτε στιγμή να αναστείλουν την εκτέλεση μιας διεργασίας και να την επανεκκινήσουν αργότερα χωρίς να επηρεάζουν την συμπεριφορά της.

2.2.3. Μοντέλο του Συστήματος

Ένα συνηθισμένο μοντέλο που αναφέρεται στην βιβλιογραφία, και αποτελεί την βάση για τα συστήματα πραγματικού χρόνου, παρουσιάζεται στη συνέχεια [9].

Ένα σύνολο διεργασιών **P** έχει τα στοιχεία $\{\tau_1,...\tau_i,...,\tau_n\}$, όπου *n* είναι η πληθικότητα του συνόλου **P**. Κάθε τ_i έχει τα ακόλουθα χαρακτηριστικά:

- D_i είναι η προθεσμία
- T_i είναι η περίοδος
- C_i είναι ο χρόνος εκτέλεσης

Για το σύνολο **P** ισχύουν οι ακόλουθοι περιορισμοί:

i. $C_i \leq D_i = T_i$ δηλαδή οι διεργασίες έχουν χρόνο εκτέλεσης μικρότερο από την προθεσμία τους και η προθεσμία είναι ίση με την περίοδο.
ii. Ο χρόνος εκτέλεσης για μια διεργασία είναι σταθερός.
iii. Όλες οι διεργασίες είναι περιοδικές.

iv. Δεν υπάρχουν *σχέσεις διάταξης (precedence relations)* ανάμεσα στις διεργασίες.

v. Δεν επιτρέπεται *διαδιεργασιακή επικοινωνία (inter-process communication)* ή συγχρονισμός.

vi. Δεν λαμβάνονται υπόψη οι απαιτήσεις για πόρους, που έχουν οι διεργασίες.

vii. Οι *εναλλαγές περιβάλλοντος (context switch)* έχουν μηδενικό κόστος.

viii. Όλες οι διεργασίες κατανέμονται σε έναν επεξεργαστή.

Παρολαυτά, ένα ρεαλιστικό μοντέλο απαιτεί την χαλάρωση των παραπάνω περιορισμών. Για παράδειγμα θα μπορούσαν να ισχύουν τα παρακάτω:

- Οι διεργασίες να είναι περιοδικές και απεριοδικές.
- Να υπάρχουν σχέσεις διάταξης ανάμεσα στις διεργασίες του συνόλου **P**, π.χ η εκτέλεση της διεργασίας τ_i να προηγείται της τ_j.
- Να επιτρέπεται η ύπαρξη κοινόχρηστων πόρων μεταξύ των διεργασιών, που προστατεύονται, π.χ με σημαφόρους.
- Οι χρόνοι εκτέλεσης των διεργασιών να είναι μεταβλητοί και να κυμαίνονται μεταξύ μιας ελάχιστης και μιας μέγιστης τιμής.

2.2.4. Χρονοπρογραμματισμός σε Συστήματα με Έναν Επεξεργαστή

Στα συστήματα αυτά υπάρχει ένας χρονοπρογραμματιστής και μόνο μία διεργασία εκτελείται την φορά. Επίσης οι διεργασίες είναι ανεξάρτητες μεταξύ τους. Στη συνέχεια θα μελετήσουμε αλγόριθμους χρονοπρογραμματισμού περιοδικών διεργασιών και θα εξετάσουμε και την περίπτωση που στο σύστημα εκτελούνται από κοινού περιοδικές και απεριοδικές διεργασίες.

2.2.4.1. Χρονοπρογραμματιστής RM (Rate Monotonic)

Στην απλή περίπτωση όπου οι διεργασίες του συστήματος είναι όλες περιοδικές, έχει αποδειχθεί ότι ο αλγόριθμος *RM (rate monotonic)* είναι ένας βέλτιστος αλγόριθμος στατικών προτεραιοτήτων [9]. Λέγοντας βέλτιστος εννοούμε ότι εάν μια διεργασία μπορεί να δρομολογηθεί από οποιοδήποτε αλγόριθμο στατικών προτεραιοτήτων τότε μπορεί επίσης να δρομολογηθεί και από τον RM.

Σύμφωνα με τον αλγόριθμο RM, σε όλες τις διεργασίες ανατίθεται μια προτεραιότητα σύμφωνα με την περίοδό τους. Η ανάθεση γίνεται ως εξής: όσο πιο σύντομη είναι η περίοδος της διεργασίας τόσο υψηλότερη θα είναι η προτεραιότητα της. Σε αυτό το απλό σχήμα οι προτεραιότητες παραμένουν στατικές και επομένως η υλοποίηση είναι απλή. Ο αλγόριθμος αυτός είναι ουσιαστικά offline και η απόδοση του, ως προς την αξιοποίηση του επεξεργαστή, είναι ικανοποιητική. Ο αλγόριθμος αποτελείται από δύο βήματα. Στο πρώτο βήμα αποφασίζε-

ται εάν ένα δεδομένο σύνολο διεργασιών μπορεί να δρομολογηθεί μέσω του RM και στο δεύτερο ορίζεται μια διάταξη των διεργασιών βάσει της στατικής τους προτεραιότητας. Η συνθήκη που ελέγχεται στο πρώτο βήμα είναι [9]:

$$n(2^{\frac{1}{n}}-1)\geqslant U \tag{1}$$

όπου

- n είναι ο αριθμός των διεργασιών
- U είναι η συνολική *αξιοποίηση (utilization)* του επεξεργαστή από τις διεργασίες

Το U δίνεται από την παρακάτω σχέση:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{2}$$

Με βάση την συνθήκη (1), έχει αποδειχθεί ότι το U συγκλίνει στο 69% όταν το n είναι μεγάλο [9]. Εάν ένα σύνολο διεργασιών έχει αξιοποίηση επεξεργαστή μικρότερη από 69% τότε είναι εγγυημένο ότι θα δρομολογηθεί από τον αλγόριθμο RM. Εντούτοις, αυτή η συνθήκη είναι ικανή αλλά όχι και αναγκαία. Αυτό σημαίνει ότι μπορεί να υπάρχουν σύνολα διεργασιών που μπορούν να δρομολογηθούν χρησιμοποιώντας τον RM, αλλά που ξεπερνούν το προαναφερθέν *όριο αξιοποίησης (utilization bound)*.

Στο [7] προτείνεται μια μέθοδος η οποία μπορεί να δείξει εάν ένα συγκεκριμένο σύνολο διεργασιών μπορεί να χρονοπρογραμματιστεί ακόμη και αν ξεπερνά το παραπάνω όριο αξιοποίησης. Αυτή η μέθοδος είναι πιο πολύπλοκη αλλά μπορεί να επιφέρει την μέγιστη αξιοποίηση του επεξεργαστή, δηλαδή 100%.

2.2.4.2. Χρονοπρογραμματιστής EDF (Earliest Deadline First)

Όπως και στον αλγόριθμο RM έτσι και στον αλγόριθμο EDF θεωρούμε ότι υπάρχει ένας προεκχωρητικός χρονοπρογραμματιστής που βασίζεται σε προτεραιότητες. Ο αλγόριθμος αυτός είναι online και *βέλτιστος (optimum)*. Χρησιμοποιώντας αυτόν τον αλγόριθμο οι προτεραιότητες ανατίθενται στις εργασίες σύμφωνα με τις προθεσμίες τους: στην εργασία με την τρέχουσα πιο σύντομη προθεσμία θα ανατεθεί η μεγαλύτερη προτεραιότητα. Η εργασία με την μεγαλύτερη προτεραιότητα θα εκτελείται στον επεξεργαστή. Χρησιμοποιώντας τον EDF, ένα σύνολο εργασιών είναι εφικτό όταν ισχύει η ακόλουθη συνθήκη

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

Η παραπάνω συνθήκη είναι ικανή και αναγκαία και σύμφωνα με αυτήν, η αξιοποίηση του επεξεργαστή μπορεί να φτάσει το 100%. Στην περίπτωση που οι προθεσμίες των εργασιών δεν είναι ίσες με τις περιόδους τότε η συνθήκη αυτή είναι ικανή αλλά όχι αναγκαία [1].

Όπως αναφέρεται στο [9] ο αλγόριθμος EDF είναι βέλτιστος υπό την έννοια ότι εάν ένα σύνολο εργασιών μπορεί να χρονοδρομολογηθεί από οποιονδήποτε αλγόριθμο, τότε μπορεί να χρονοδρομολογηθεί και από τον EDF.

2.2.4.3. Ανεξάρτητες Απεριοδικές Διεργασίες

Στα περισσότερα συστήματα πραγματικού χρόνου εκτελούνται μαζί απεριοδικές και περιοδικές διεργασίες. Ο αλγόριθμος RM μπορεί να προσαρμοστεί για να χειρίζεται απεριοδικές και περιοδικές διεργασίες. Στο [8] προτείνονται διάφοροι μηχανισμοί εξυπηρετητών (Deferrable Server και Priority Exchange) για να την βελτίωση της *απόκρισης (responsiveness)* των απεριοδικών διεργασιών. Η βασική ιδέα είναι να υπάρχει μια περιοδική διεργασία της οποίας λειτουργία θα είναι να εξυπηρετεί μια ή περισσότερες απεριοδικές διεργασίες. Αυτή η περιοδική διεργασία θα λειτουργεί σαν εξυπηρετητής και στην οποία μπορεί να διατεθεί ο μέγιστος δυνατός χρόνος εκτέλεσης έτσι ώστε να τηρούνται και οι προθεσμίες των άλλων περιοδικών διεργασιών. Στο [17] περιγράφεται ένας καλύτερος μηχανισμός εξυπηρέτησης που ονομάζεται Sporadic Server. Ένας βέλτιστος μηχανισμός εξυπηρέτησης, που ονομάζεται Slack Stealer, έχει προταθεί στο [6] και βασίζεται στην ακόλουθη ιδέα: ο μηχανισμός εξυπηρέτησης "κλέβει" όσο περισσότερο χρόνο εκτέλεσης επιτρέπεται από τις περιοδικές διεργασίες χωρίς να προκαλέσει το χάσιμο των προθεσμιών τους. Παρόλο που αυτός ο μηχανισμός δεν θεωρείται πρακτικός, λόγω του υψηλού επιπλέον φόρτου που προκαλεί, ο αλγόριθμος αυτού του μηχανισμού παρέχει ένα κάτω φράγμα στον χρόνο απόκρισης των απεριοδικών διεργασιών.

Οι παραπάνω μηχανισμοί υποθέτουν ότι ο χρονοπρογραμματισμός των περιοδικών εργασιών γίνεται χρησιμοποιώντας τον αλγόριθμο RM. Παρόλο που ο RM είναι ένας βέλτιστος αλγόριθμος, στην γενική περίπτωση δεν μπορεί να επιτύχει πλήρη αξιοποίηση του επεξεργαστή. Στη χειρότερη περίπτωση, η μέγιστη αξιοποίηση επεξεργαστή που μπορεί να επιτύχει είναι περίπου 69%, ενώ γενικά για κάποιο τυχαίο σύνολο εργασιών μπορεί να φτάσει περίπου στο 88% [19]. Για πολλές εφαρμογές κάτι τέτοιο αποτελεί έναν πολύ σημαντικό περιορισμό που μπορεί να μην είναι αποδεκτός. Η αξιοποίηση του επεξεργαστή μπορεί να αυξηθεί χρησιμοποιώντας αλγορίθμους χρονοπρογραμματισμού, όπως είναι ο EDF. Ο EDF, όπως έχει αναφερθεί, είναι βέλτιστος και επιτυγχάνει πλήρη αξιοποίηση του επεξεργαστή. Έχει αποδειχτεί ότι ο EDF παραμένει βέλτιστος όσον αφορά την από κοινού χρονοδρομολόγηση περιοδικών και απεριοδικών διεργασιών [2]. Έχουν προταθεί διάφοροι μηχανισμοί και αλγόριθμοι εξυπη-

ρέτησης απεριοδικών εργασιών που βασίζονται στον αλγόριθμο EDF. Ανάμεσα σε αυτούς είναι ο αλγόριθμος *Dynamic Priority Exchange* και ο αλγόριθμος *Total Bandwidth Server* που ανήκει στην κατηγορία αλγορίθμων *"διατήρησης εύρους-ζώνης" (bandwidth preserving)*. Σύμφωνα με αυτήν την κατηγορία αλγορίθμων ένα ποσοστό της αξιοποίησης του επεξεργαστή δεσμεύεται για την εξυπηρέτηση των απεριοδικών εργασιών.

2.2.4.4. Total Bandwidth Server (TBS)

Η βασική ιδέα πίσω από τον TBS είναι ότι κάθε φορά που καταφτάνει μια απεριοδική εργασία στο σύστημα, της αποδίδεται το *συνολικό εύρος ζώνης (total bandwidth)* ενός εξυπηρετητή. Η διεργασία-εξυπηρετητής προσθέτει την κάθε απεριοδική εργασία στο χρονοπρόγραμμα κατά την άφιξη της, χωρίς όμως να παραβιάσει τις προθεσμίες των άλλων εργασιών που ήδη βρίσκονται στο χρονοπρόγραμμα.

Η χωρητικότητα του εξυπηρετητή εκφράζεται σαν κλάσμα της συνολικής αξιοποίησης της CPU. Για τις απεριοδικές εργασίες, ο εξυπηρετητής δεν θα πρέπει να ορίζει προθεσμίες που θα προκαλούσαν την απαίτηση περισσότερου χρόνου από όσον διαθέτει ο εξυπηρετητής. Π.χ εάν εμφανιστεί μια απεριοδική εργασία που θα χρειαζόταν ένα χιλιοστό του δευτερολέπτου (millisecond) του χρόνου της CPU και η χωρητικότητα του εξυπηρετητή ήταν το ένα δέκατο του χρόνου του επεξεργαστή, τότε η προθεσμία της εργασίας θα ήταν δέκα χιλιοστά του δευτερολέπτου στο μέλλον. Η εξυπηρέτηση των απεριοδικών εργασιών δεν μπορεί να επικαλύπτεται, έτσι τα δέκα χιλιοστά του δευτερολέπτου μετριούνται από την άφιξη της εργασίας ή από την πιο τελευταία προθεσμία που έχει οριστεί από τον εξυπηρετητή.

Πιο συγκεκριμένα όταν καταφτάσει στο σύστημα η κ-οστή εργασία τη χρονική στιγμή r_k, θα της ανατεθεί μια προθεσμία βάσει του παρακάτω τύπου:

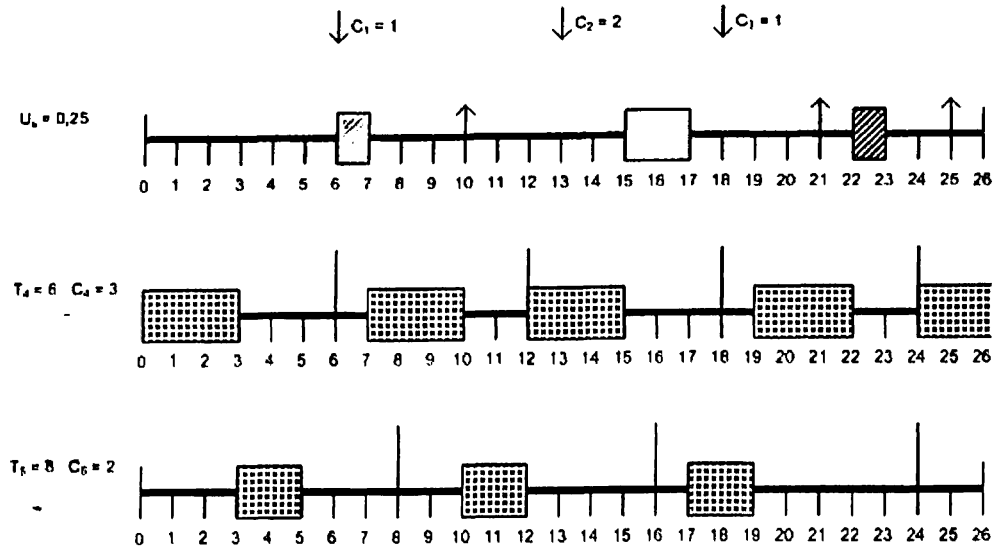
$$d_k = max(r_k, d_{k-1}) + \frac{C_k}{U_s}$$

όπου

- C_k είναι ο χρόνος εκτέλεσης της εργασίας.
- U_S είναι ο παράγοντας αξιοποίησης του εξυπηρετητή, δηλαδή το εύρος ζώνης του.

Εξ' ορισμού $d_0 = 0$.

Στη συνέχεια η εργασία προστίθεται στην ουρά του συστήματος και χρονοδρομολογείται σύμφωνα με τον αλγόριθμο EDF, όπως κάθε άλλη περιοδική εργασία.



Σχήμα 2.4 Εξυπηρετητής TB.

Ο όρος $max(r_k, d_{k-1})$ στον παραπάνω τύπο ουσιαστικά "θυμάται" το εύρος ζώνης που έχει ήδη ανατεθεί σε άλλες εργασίες. Η ανάθεση των προθεσμιών γίνεται με τετοιο τρόπο έτσι ώστε σε κάθε χρονική στιγμή ο χρόνος που διατίθεται για τις απεριοδικές εργασίες δεν ξεπερνά τον παράγοντα αξιοποίησης U_S του εξυπηρετητή.

Στο Σχήμα 2.4 απεικονίζεται η συμπεριφορά του εξυπηρετητή TB. Στην πρώτη απεριοδική εργασία, η οποία φτάνει τη χρονική στιγμή 6, ανατίθεται μια προθεσμία $d_1 = \max(0, r_1) + C_1/U_S = 6 + 1/0,25 = 10$. Λόγω του ότι η προθεσμία d_1 είναι η πιο σύντομη στο σύστημα, η απεριοδική εργασία θα εκτελεστεί αμέσως. Η δεύτερη απεριοδική εργασία καταφθάνει τη στιγμή 13 και της ανατίθεται προθεσμία $d_2 = \max(d_1, r_2) + C_2/U_S = 13 + 2/0,25 = 21$. Η δεύτερη απεριοδική εργασία δεν εκτελείται αμέσως διότι τη χρονική στιγμή 13 εκτελείται μια περιοδική εργασία με μικρότερη προθεσμία. Τέλος, η τρίτη απεριοδική εργασία καταφθάνει τη χρονική στιγμή 18 και λαμβάνει προθεσμία $d_3 = \max(d_2, r_3) + C_3/U_S = 21 + 1/0,25 = 25$ και εκτελείται τη χρονική στιγμή 22.

Στην περίπτωση που ο παράγοντας U_S είναι μεγαλύτερος, αποδίδονται μικρότερες προθεσμίες στις απεριοδικές εργασίες, με συνέπεια να έχουν μεγαλύτερη προτεραιότητα. Συνεπώς, σε αυτήν την περίπτωση οι χρόνοι απόκρισης των απεριοδικών εργασιών είναι μικρότεροι.

Στο [19] αποδεικνύεται το παρακάτω θεώρημα που ουσιαστικά αποδεικνύει την εφικτότητα χρονοπρογραμματισμού ενός συνόλου εργασιών, χρησιμοποιώντας τον TBS:
"Δοθέντος ενός συνόλου *n* περιοδικών εργασιών με παράγοντα αξιοποίησης U_P και ενός εξυπηρετητή TB με παράγοντα αξιοποίησης U_S, το σύνολο είναι χρονοπρογραμματίσιμο εάν και μόνο εάν $U_P + U_S \leq 1$".

Η υλοποίηση του TBS είναι από τις πιο απλές, συγκριτικά με άλλες μεθόδους εξυπηρέτη-

σης απεριοδικών εργασιών [19]. Για την απόδοση μια προθεσμίας σε μια νεοεισερχόμενη εργασία, το μόνο που απαιτείται είναι να κρατάμε αποθηκευμένη την προθεσμία που αποδόθηκε στην τελευταία απεριοδική εργασία. Στην συνέχεια η εργασία θα προστεθεί στην ουρά και θα χρονοδρομολογηθεί μέσω του αλγορίθμου EDF όπως κάθε άλλη περιοδική εργασία. Συνεπώς η επιβάρυνση είναι πρακτικά αμελητέα.

2.2.4.5. Dynamic Priority Exchange Server (DPE)

Στον αλγόριθμο *δυναμικής ανταλλαγής προτεραιοτήτων (dynamic priority exchange)* [19], μια περιοδική διεργασία ορίζεται σαν εξυπηρετητής για τις μη-περιοδικές διεργασίες. Η βασική ιδέα του αλγορίθμου είναι να επιτρέπει στον εξυπηρετητή να ανταλλάσσει τον χρόνο εκτέλεσής του (run-time), με τον χρόνο εκτέλεσης άλλων περιοδικών διεργασιών μικρότερης προτεραιότητας σε περίπτωση που δεν υπάρχουν εκκρεμούν απεριοδικές αιτήσεις. Με αυτόν τον τρόπο, ο χρόνος εκτέλεσης του εξυπηρετητή ανταλλάσσεται μόνο με αυτόν των περιοδικών διεργασιών.

Στο [8] παρουσιάζεται ο αρχικός αλγόριθμος *ανταλλαγής προτεραιοτήτων (priority exchange (PE))*, δηλαδή ένας εξυπηρετητής απεριοδικών αιτήσεων που χρησιμοποιεί τον αλγόριθμο RM. Ο εξυπηρετητής DPE αποτελεί επέκταση του PE, έτσι ώστε να χρησιμοποιεί τον αλγόριθμο EDF. Ο αλγόριθμος ορίζεται ως εξής [19]:

- Ο εξυπηρετητής έχει περίοδο T_S και χωρητικότητα C_S.
- Στην αρχή κάθε περιόδου, η *απεριοδική* χωρητικότητα του εξυπηρετητή παίρνει την τιμή C_S^d, όπου d είναι η προθεσμία της τρέχουσας περιόδου του εξυπηρετητή.
- Κάθε προθεσμία d που σχετίζεται με τις εκτελέσεις της i-οστής περιοδικής διεργασίας, έχει απεριοδική χωρητικότητα $C_{S_i}^d$ ίση με μηδέν.
- Στις απεριοδικές χωρητικότητες που έχουν τιμή μεγαλύτερη του μηδενός, ανατίθενται προτεραιότητες σύμφωνα με τις προθεσμίες τους, βάσει του αλγορίθμου EDF, όπως και σε όλες τις περιοδικές διεργασίες.
- Οποτεδήποτε η αίτηση με την μεγαλύτερη προτεραιότητα στο σύστημα είναι μια απεριοδική χωρητικότητα C χρονικών μονάδων, συμβαίνουν τα παρακάτω:
  - → Εάν υπάρχουν απεριοδικές αιτήσεις στο σύστημα, αυτές εξυπηρετούνται εώς ότου ολοκληρωθούν ή μέχρι να εξαντληθεί η χωρητικότητα (κάθε αίτηση καταναλώνει χωρητικότητα ίση με τον χρόνο εκτέλεσής της).
  - → Εάν δεν υπάρχουν απεριοδικές αιτήσεις στο σύστημα τότε εκτελείται η περιοδική διεργασία με την μικρότερη προθεσμία. Μια χωρητικότητα ίση με το μήκος της εκτέλεσης προστίθεται στην απεριοδική χωρητικότητα της προθεσμίας της διεργα-

σίας και αφαιρείται από το C (δηλ. γίνεται ανταλλαγή της προθεσμίας της υψηλής προτεραιότητας χωρητικότητας και της περιοδικής διεργασίας).

→ Εάν δεν υπάρχουν ούτε απεριοδικές αλλά ούτε και περιοδικές αιτήσεις τότε έχουμε αδρανή χρόνο και η χωρητικότητα C καταναλώνεται μέχρι, το πολύ, να εξαντληθεί.

Αν και η επιβάρυνση που επιφέρει ο εξυπηρετητής DPE στο σύστημα είναι στην τάξη μεγέθους του EDF, η υλοποίησή του δεν θεωρείται τετριμμένη.

2.2.4.6. Χρόνοι Εκτέλεσης

Για να παρθούν αποτελεσματικές αποφάσεις χρονοπρογραμματισμού απαιτείται γνώση του χρόνου εκτέλεσης μιας διεργασίας. Στην πραγματικότητα αυτός ο χρόνος μπορεί να ποικίλει με κάθε εκτέλεση της διεργασίας. Συνεπώς υπάρχουν δύο εναλλακτικές λύσεις:

i. Χρησιμοποίηση του χρόνου εκτέλεσης της χειρότερης περίπτωσης.
ii. Χρησιμοποίηση χρόνου λιγότερου από τον χρόνο εκτέλεσης της χειρότερης περίπτωσης.

Εάν ο χρονοπρογραμματισμός γίνει σύμφωνα με τους χρόνους της χειρότερης περίπτωσης, και οι χρόνοι αυτοί είναι αισθητά μεγαλύτεροι από τους χρόνους της μέσης περίπτωσης, τότε θα έχουμε σαν αποτελέσματα πτώση της αξιοποίησης του επεξεργαστή. Παραδείγματος χάριν, ας θεωρήσουμε δύο διεργασίες που δρομολογούνται χρησιμοποιώντας τον αλγόριθμο RM. Η συνολική αξιοποίηση του επεξεργαστή, χρησιμοποιώντας τους χρόνους της χειρότερης περίπτωσης, έστω ότι είναι 82,8%, δηλαδή όσο είναι ακριβώς ο περιορισμός που επιβάλλει η σχέση (1). Σε αυτήν την περίπτωση καμία άλλη διεργασία δεν μπορεί να χρονοδρομολογηθεί. Εάν η μέση αξιοποίηση των διεργασιών είναι το μισό του χρόνου της χειρότερης περίπτωσης, τότε η αξιοποίηση του επεξεργαστή θα είναι 41,4%, χωρίς να μπορούν άλλες διεργασίες να χρονοδρομολογηθούν στον επεξεργαστή.

Από την άλλη εάν ο χρονοπρογραμματισμός γίνει χρησιμοποιώντας χρόνους εκτέλεσης που είναι μικρότεροι από τους χρόνους της χειρότερης περίπτωσης, τότε υπάρχει περίπτωση να έχουμε την εμφάνιση υπερφόρτωσης. Εάν κάτι τέτοιο προκύψει, τότε θα πρέπει τουλάχιστον να μην παραβιαστούν οι περιορισμοί εκτέλεσης των κρίσιμων διεργασιών. Για αυτό το θέμα έχουν προταθεί δύο προσεγγίσεις [1]:

i. Η εξασφάλιση ότι οι διεργασίες που χάνουν τις προθεσμίες τους δεν είναι κρίσιμες, δηλ. η σειρά κατά την οποία οι διεργασίες χάνουν τις προθεσμίες τους πρέπει να είναι προβλέψιμη. Κατά συνέπεια, οι διεργασίες που δεν είναι κρίσιμες μπορεί να υποχρεωθούν να χάσουν τις προθεσμίες τους.

ii. Σε ένα πολυεπεξεργαστικό ή κατανεμημένο σύστημα οι διεργασίες που θεωρούνται πιθανές να χάσουν τις προθεσμίες τους μεταναστεύονται σε άλλους επεξεργαστές ή συστήματα. Αυτή η προσέγγιση δεν είναι δυνατή σε ένα σύστημα που χρησιμοποιείται ένας offline χρονοπρογραμματιστής. Θα εξετάσουμε τον χρονοπρογραμματισμό σε πολυεπεξεργαστικά συστήματα παρακάτω.

2.2.4.7. Υπολογισμός των Χρόνων Εκτέλεσης

Για τον υπολογισμό του χρόνου εκτέλεσης μιας διεργασίας απαιτείται ανάλυση του κώδικα της διεργασίας. Ένα προτεινόμενο μοντέλο είναι το εξής: ο κώδικας χωρίζεται σε τμήματα, έτσι ώστε ένα τμήμα να αποτελείται είτε από ένα αίτημα για πόρους (resource request), είτε από κώδικα συγχρονισμού, είτε από απλά μπλοκ κοινού κώδικα. Σε κάθε τμήμα ανατίθεται και ένας χρόνος εκτέλεσης της χειρότερης περίπτωσης. Στην συνέχεια υπολογίζεται ο συνολικός χρόνος εκτέλεσης της διεργασίας βάσει αυτών των χρόνων.

Όταν ο κώδικας εκτελεί μια λειτουργία που μπορεί να προκαλέσει μπλοκάρισμα της διεργασίας, πρέπει να είναι γνωστό ένα άνω φράγμα (upper bound) του χρόνου μπλοκαρίσματος για να είναι εφικτός ο υπολογισμός του χρόνου της χειρότερης περίπτωσης. Ένας τρόπος για να ορίσουμε το άνω φράγμα είναι από το γνώρισμα του υλικού (hardware) ότι όλες οι αιτήσεις για τις συσκευές εξυπηρετούνται με τρόπο FIFO. Για παράδειγμα, μια διεργασία μπορεί να καθυστερήσει επειδή εκτελείται η κρίσιμη περιοχή κάθε άλλης διεργασίας. Η εκτέλεση κάθε κρίσιμης περιοχής γίνεται όταν απαιτείται κάποιος πόρος από μια διεργασία. Ένας άλλος τρόπος υπολογισμού του άνω φράγματος γίνεται με το να περιοριστούν τα τμήματα του κώδικα μόνο σε εκείνα που είναι *αναλύσιμα (analysable)* [1]. Ορισμένοι περιορισμοί που μπορούν να εφαρμοστούν είναι:

- Δεν επιτρέπεται καμία άμεση ή έμμεση αναδρομική κλήση.
- Δεν επιτρέπονται παράμετροι σε συναρτήσεις ή υποπρογράμματα.
- Φραγμένοι βρόχοι είτε από έναν μέγιστο αριθμό επαναλήψεων είτε από ένα χρονικό όριο (timeout).
- Μια άλλη υπόθεση είναι ότι όλοι οι πόροι είναι διαθέσιμοι όταν απαιτούνται και έτσι δεν γίνεται ποτέ μπλοκάρισμα.

Παρόλο που οι παραπάνω προσεγγίσεις για τον υπολογισμό του άνω φράγματος είναι εφικτές, δεν μπορούν να εφαρμοστούν σε όλες τις περιπτώσεις συστημάτων. Γενικά ο υπολογισμός των χρόνων εκτέλεσης παραμένει ένα από τα βασικά ζητήματα στο ερευνητικό πεδίο των χρονοπρογραμματιστών.

2.2.5. Χρονοπρογραμματισμός σε Συστήματα Πολλών Επεξεργαστών

Η ανάπτυξη κατάλληλων τεχνικών χρονοπρογραμματισμού σε πολυεπεξεργαστικά συστήματα είναι αρκετά διαφορετική σε σχέση με τα συστήματα ενός επεξεργαστή διότι δεν είναι άμεσα εφαρμόσιμοι οι αλγόριθμοι των μονοεπεξεργαστικών συστημάτων.

Έχει αποδειχθεί ότι οι αλγόριθμοι που είναι βέλτιστοι για τα μονοεπεξεργαστικά συστήματα δεν είναι βέλτιστοι για πολυεπεξεργαστικά συστήματα [1]. Για παράδειγμα, ας θεωρήσουμε 3 περιοδικές διεργασίες P_1, P_2 και P_3 που πρέπει να εκτελεσθούν σε 2 επεξεργαστές. Έστω ότι η P_1 και η P_2 έχουν προθεσμία και περίοδο 50 μονάδες. Ο απαιτούμενος χρόνος εκτέλεσης ανά περίοδο είναι 25 μονάδες. Έστω ότι η P_3 έχει περίοδο 100 μονάδες και χρόνο εκτέλεσης 80 μονάδες.

Εάν χρησιμοποιηθεί ο αλγόριθμος RM ή ο EDF, η P_1 και η P_2 θα έχουν την πιο υψηλή προτεραιότητα και θα τρέξουν στους δύο επεξεργαστές, παράλληλα, για 25 μονάδες. Στις υπόλοιπες 75 μονάδες που είναι διαθέσιμες, θα πρέπει να εκτελεστεί η P_3 για 80 μονάδες. Το γεγονός ότι η P_3 έχει δύο επεξεργαστές διαθέσιμους δεν παίζει ρόλο καθώς ένας από τους δύο θα παραμείνει *άεργος (idle)*. Συνεπώς η P_3 θα χάσει την προθεσμία της παρόλο που η μέση αξιοποίηση των επεξεργαστών είναι μόνο 65%. Εντούτοις μια κατανομή που τοποθετεί τις P_1 και P_2 στον ένα επεξεργαστή και την P_3 στον άλλο, τηρεί όλες τις προθεσμίες. Έχει αποδειχθεί ότι το να βρούμε ένα βέλτιστο χρονοπρόγραμμα σε ένα σύστημα πολλών επεξεργαστών είναι NP-hard.

Το παραπάνω παράδειγμα έδειξε ότι η συνετή κατανομή (allocation) των διεργασιών μπορεί να επηρεάσει σημαντικά το χρονοπρόγραμμα. Ας θεωρήσουμε ένα ακόμη παράδειγμα: έστω ότι έχουμε 4 διεργασίες και 2 επεξεργαστές και έστω ότι οι περίοδοι τους είναι 10, 10, 14 και 14 αντίστοιχα. Εάν οι δύο διεργασίες με περίοδο 10 εκτελεστούν στον ίδιο επεξεργαστή και οι άλλες δύο στον άλλο, τότε η αξιοποίηση των επεξεργαστών μπορεί να φτάσει στο 100%. Οι χρόνοι εκτέλεσης των διεργασιών είναι 5, 5, 10 και 4 αντίστοιχα. Εντούτοις, εάν μια διεργασία με περίοδο 10 και μία με 14 τοποθετηθούν μαζί στον ίδιο επεξεργαστή, τότε η αξιοποίηση του επεξεργαστή μειώνεται περίπου στο 83%.

Το παραπάνω παράδειγμα δείχνει ότι είναι καλύτερο να κατανεμηθούν στατικά οι περιοδικές διεργασίες, δηλ. όλα τα στιγμιότυπα τους να εκτελούνται στον ίδιο επεξεργαστή, παρά να αφεθούν να μεταναστεύσουν και, κατά συνέπεια, ενδεχομένως να υποβαθμίσουν την απόδοση του συστήματος. Ακόμη και σε ένα *ισχυρά συνδεμένο (tightly coupled)* σύστημα που τρέχει έναν ενιαίο *αποστολέα (dispatcher)* είναι καλύτερο να διατηρηθούν οι διεργασίες στον ίδιο επεξεργαστή παρά να γίνει προσπάθεια να εκτελεστούν σε έναν άεργο επεξεργαστή και να διακινδυνευτεί η ισορροπία της κατανομής.

2.3. Συσχετισμός με την Παρούσα Εργασία

Όπως θα δούμε και στα παρακάτω κεφάλαια, στην παρούσα εργασία ισχύουν τα παρακάτω σε σχέση με το βασικό υπόβαθρο που συζητήθηκε σε αυτό το κεφάλαιο:

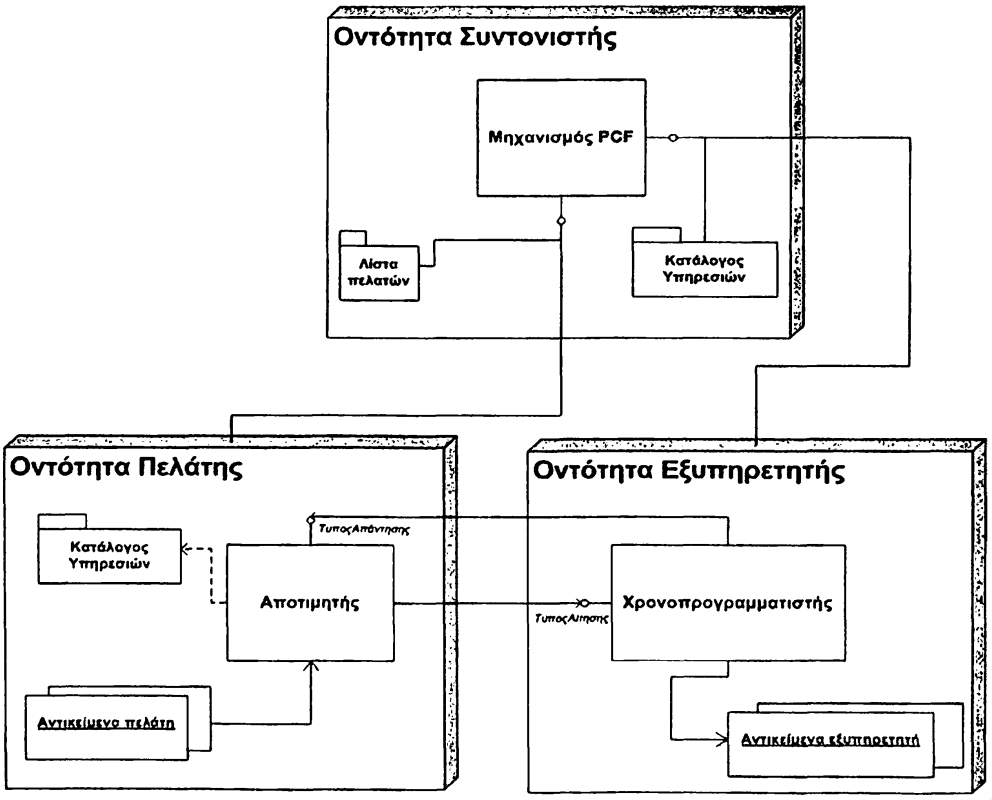
i. Το πρωτόκολλο που χρησιμοποιείται είναι το IEEE 802.11 για ασύρματα τοπικά δίκτυα (LAN). Οι οντότητες δεν επικοινωνούν με ad hoc τρόπο, αλλά υπάρχει, όπως θα δούμε και στο Κεφάλαιο 3, μια οντότητα που παίζει τον ρόλο του σημείου πρόσβασης. Οι οντότητες μπορούν να εισέρχονται στο τοπικό αυτό ασύρματο δίκτυο οποιαδήποτε στιγμή επιθυμούν, αλλά μπορούν να αποχωρούν μόνο μια προκαθορισμένη χρονική στιγμή την οποία έχουν κάνει γνωστή κατά την είσοδο τους στο δίκτυο. Αυτός ο περιορισμός μπορεί να φαίνεται αυστηρός για ένα ad hoc κινητό δίκτυο γενικής φύσης, αλλά από την πλευρά ενός συστήματος κόμβων που παρέχουν υπηρεσίες με απαιτήσεις πραγματικού χρόνου, κάτι τέτοιο φαίνεται λογικό.

ii. Κάθε κινητή οντότητα είναι εξοπλισμένη με έναν επεξεργαστή. Σε κάθε οντότητα που παρέχει κάποιες υπηρεσίες με απαιτήσεις πραγματικού χρόνου, στο σύστημα που προτείνουμε, εκτελείται ένας χρονοπρογραμματιστής. Όπως θα δούμε και στο Κεφάλαιο 4, ανάλογα με την πολιτική που ακολουθείται, χρησιμοποιούνται διαφορετικοί αλγόριθμοι χρονοπρογραμματισμού. Συγκεκριμένα, οι αλγόριθμοι σε όλες τις πολιτικές που προτείνονται είναι online, μιας και δεν είναι γνωστό εκ των προτέρων πότε και πόσες αιτήσεις θα καταφθάσουν σε κάθε κόμβο που παρέχει κάποιες υπηρεσίες. Όσον αφορά τις υπηρεσίες, είναι γνωστός ο χρόνος εκτέλεσης C της χειρότερης περίπτωσης. Στην περίπτωση που η πολιτική επιτρέπει την από κοινού χρονοδρομολόγηση περιοδικών και απεριοδικών εργασιών, τότε χρησιμοποιείται ο μηχανισμός TBS μαζί με τον αλγόριθμο EDF, συνεπώς η ανάθεση των προθεσμιών γίνεται δυναμικά. Ο λόγος που έχει επιλεγεί ο μηχανισμός TBS είναι λόγω της απλότητας στην υλοποίηση του και της πολύ καλής του απόδοσης [19].

ΚΕΦΑΛΑΙΟ 3. ΑΡΧΙΤΕΚΤΟΝΙΚΗ

3.1. Υπολογισμός Χρόνου Μετάδοσης Ενός Μηνύματος

Το θεμελιώδες στοιχείο της αρχιτεκτονικής της προτεινόμενης υπηρεσίας ενδιάμεσου λογισμικού, που παρουσιάζεται σε αυτήν την εργασία, είναι η κινητή οντότητα. Μια κινητή οντότητα μπορεί να επικοινωνεί με άλλες κινητές οντότητες αλλά και με σταθερές οντότητες. Οι κινητές οντότητες εκτελούνται σε φορητές συσκευές που οι δυνατότητες τους μπορεί να είναι περιορισμένες, όπως ένα PDA ή ένα κινητό τηλέφωνο νέας γενιάς, ή ισχυρές, όπως π.χ ένας φορητός υπολογιστής. Οι σταθερές οντότητες μπορούν να εκτελούνται σε προσωπικούς υπολογιστές ή σε εξυπηρετητές με μεγάλη υπολογιστική ισχύ.



Σχήμα 3.1 Σύνοψη της Αρχιτεκτονικής της Υπηρεσίας.

Στο Σχήμα 3.1 απεικονίζονται τα βασικά συστατικά που συνθέτουν την αρχιτεκτονική της προτεινόμενης υπηρεσίας. Το υπολογιστικό δικτυακό περιβάλλον αποτελείται από έναν αριθμό οντοτήτων. Κάθε κινητή οντότητα έχει διακριτό ρόλο, δηλ. είτε έχει τον ρόλο του πελάτη ή το ρόλο του εξυπηρετητή. Οι οντότητες επικοινωνούν μεταξύ τους με ανταλλαγή μηνυμάτων. Η ανταλλαγή των μηνυμάτων πραγματοποιείται από οντότητα σε οντότητα αλλά επιπρόσθετα υπάρχει και ένας ειδικός τύπος οντότητας που αναλαμβάνει τον συντονισμό της επικοινωνίας ανάμεσα στις άλλες οντότητες. Η ειδική αυτή οντότητα ονομάζεται *οντότητα-συντονιστής (coordinator-entity)* και αναλαμβάνει επίσης την διαχείριση ενός καταλόγου που περιέχει πληροφορίες σχετικά με τις υπηρεσίες που παρέχονται στο περιβάλλον από τις οντότητες-εξυπηρετητές. Η οντότητα-συντονιστής μπορεί να είναι είτε σταθερή ή κινητή, αλλά συνήθως είναι σταθερή μιας και έτσι μπορεί να υπολογιστεί πιο εύκολα η διάρκεια ζωής της. Στην περίπτωση που ο συντονιστής είναι κινητός τότε υποθέτουμε ότι αντικαθίσταται από μια άλλη οντότητα-συντονιστή σε καθορισμένα χρονικά διαστήματα.

- Συνεπώς υπάρχουν τρεις τύποι οντοτήτων:
  i. Οντότητα-πελάτης.
  ii. Οντότητα-εξυπηρετητής.
  iii. Οντότητα-συντονιστής.

Όπως αναφέραμε, ο συντονιστής διατηρεί έναν κατάλογο με πληροφορίες σχετικά με τις παρεχόμενες υπηρεσίες. Κάθε φορά που εισέρχεται μία νέα οντότητα-εξυπηρετητής ενημερώνει τον συντονιστή για την είσοδο της. Ο συντονιστής στη συνέχεια δημιουργεί μια εγγραφή στον κατάλογο στην οποία αποθηκεύονται πληροφορίες σχετικά με την οντότητα. Η δομή της εγγραφής παρουσιάζεται αναλυτικά στο Σχήμα 3.2. Όπως βλέπουμε, για κάθε οντότητα-εξυπηρετητή αποθηκεύεται η IP διεύθυνσή της, η *θύρα (port)* στην οποία δέχεται τις αιτήσεις, η

```
struct service
{
   enum type {periodic, aperiodic};
   string serviceName;
};

struct serverInfo
{
   string IPAddress;
   unsigned short port;
   unsigned long lifetime;
   sequence <service> services;
};
```

Σχήμα 3.2 Εγγραφή στον Κατάλογο του Συντονιστή.

διάρκεια ζωής της και μια λίστα με όλες τις υπηρεσίες που προσφέρει. Μια υπηρεσία χαρακτηρίζεται από το όνομα της και μπορεί να εκτελεστεί περιοδικά ή όχι.

Αντίστοιχα, όταν μια οντότητα-εξυπηρετητής αποχωρεί, λόγω λήξης της διάρκειας ζωής της, τότε όλες οι οντότητες διαγράφουν την αντίστοιχη εγγραφή από τον κατάλογο. Αυτό πραγματοποιείται ανεξάρτητα σε κάθε οντότητα, μιας και η κάθε οντότητα γνωρίζει την διάρκεια ζωής του κάθε εξυπηρετητή στο δίκτυο.

Από την άλλη πλευρά, κάθε φορά που εισέρχεται μια οντότητα-πελάτης, ο συντονιστής αναλαμβάνει να στείλει ένα αντίγραφο του καταλόγου στον πελάτη. Συνεπώς η κάθε οντότητα-πελάτης διατηρεί και από ένα αντίγραφο του καταλόγου. Εφόσον ο κατάλογος αυτός μπορεί να μεταβληθεί στην πλευρά του συντονιστή, με την είσοδο μιας οντότητας-εξυπηρετητή, θα πρέπει να ενημερωθούν κατάλληλα και οι τοπικοί κατάλογοι που διατηρούν οι οντότητες-πελάτες.

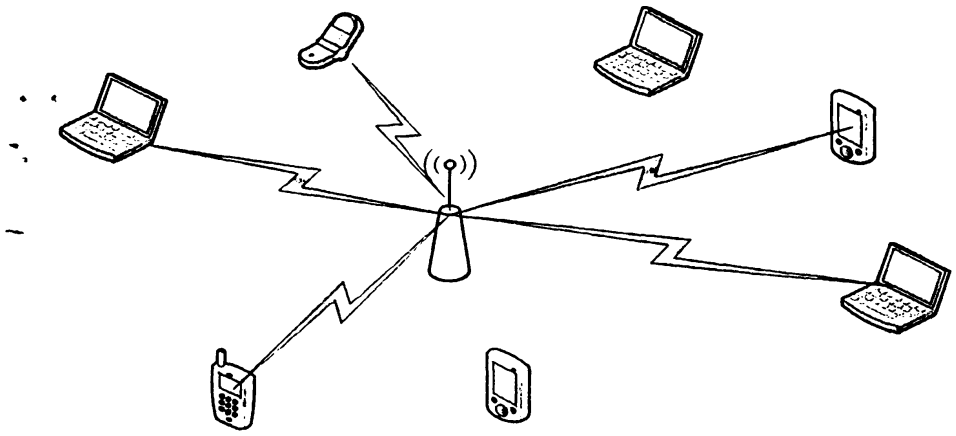
Κάτι τέτοιο μπορεί να γίνει ως εξής: κάθε φορά που γίνεται μια μεταβολή στον κατάλογο του συντονιστή, η μεταβολή αυτή αποστέλνεται και στους πελάτες που βρίσκονται στο περιβάλλον. Έτσι ο συντονιστής διατηρεί και μια λίστα με όλους τους πελάτες που βρίσκονται στο περιβάλλον.

Όπως φαίνεται και στο Σχήμα 3.1 σε κάθε οντότητα-πελάτη βρίσκονται ενθυλακωμένα αντικείμενα τα οποία επιθυμούν την κλήση μιας υπηρεσίας σε κάποια δεδομένη χρονική στιγμή. Τα αντικείμενα αυτά αποστέλλουν μια αίτηση, που αντιστοιχεί σε μια κλήση μιας υπηρεσίας, στον *αποτιμητή (assessor)* του πελάτη. Ο αποτιμητής συμβουλεύεται πρώτα τον τοπικό του κατάλογο για να αποφανθεί αν υπάρχει κάποια απομακρυσμένη οντότητα-εξυπηρετητής που παρέχει την αντίστοιχη υπηρεσία. Ο αποτιμητής επιλέγει τον πρώτο εξυπηρετητή που παρέχει την αντίστοιχη υπηρεσία, ξεκινώντας από την πρώτη εγγραφή του καταλόγου. Αν βρεθεί κάποια τέτοια οντότητα-εξυπηρετητής τότε αυτή επιλέγεται και ο αποτιμητής αναλαμβάνει να προωθήσει την αίτηση στην επιλεγμένη οντότητα-εξυπηρετητή. Η δομή του μηνύματος της αίτησης φαίνεται στο Σχήμα 3.3. Κάθε μήνυμα περιλαμβάνει την *ταυτότητα (id)* της αίτησης με την οποία σχετίζεται, την διάρκεια ζωής του πελάτη που αποστέλλει το μήνυμα και το αναγνωριστικό της υπηρεσίας την οποία θέλει να χρησιμοποιήσει ο πελάτης.

```
struct message
{
   long reqID;
   unsigned long clientLifetime;
   service s;
}
```

Σχήμα 3.3 Δομή του Μηνύματος Αίτησης.

Στην πλευρά του εξυπηρετητή, την αίτηση θα την λάβει ο χρονοπρογραμματιστής και θα εξετάσει αν θα πρέπει να συμπεριληφθεί στο χρονοπρόγραμμα ή όχι. Στην περίπτωση που ο χρονοπρογραμματιστής αποφανθεί ότι δεν μπορεί να εξυπηρετήσει την αίτηση, αν για παράδειγμα δεν έχει αρκετό διαθέσιμο χρόνο διάρκειας ζωής, τότε αποστέλλει πίσω στον αποτιμητή του πελάτη μια αρνητική απάντηση. Σε αυτήν την περίπτωση ο αποτιμητής θα πρέπει να επιλέξει έναν άλλον εξυπηρετητή για να αποστείλει την αίτηση.



Σχήμα 3.4 Συντονισμός μέσω Σημείου Πρόσβασης.

Κάθε αίτηση αντιστοιχεί σε μια υπηρεσία και αντίστοιχα κάθε υπηρεσία την υλοποιεί ένα αντικείμενο που βρίσκεται ενθυλακωμένο στον εξυπηρετητή. Ο χρονοπρογραμματιστής διατηρεί μια ουρά στην οποία αποθηκεύονται οι αιτήσεις. Ο χρονοπρογραμματισμός της εκτέλεσης των αιτήσεων εξαρτάται από την πολιτική που ακολουθείται από τον χρονοπρογραμματιστή. Οι εναλλακτικές πολιτικές που μπορεί να ακολουθήσει ο χρονοπρογραμματιστής αναλύονται λεπτομερώς στο Κεφάλαιο 4.

3.1. Υπολογισμός Χρόνου Μετάδοσης Ενός Μηνύματος

Ο απώτερος στόχος των πολιτικών που προτείνουμε, είναι να εγγυηθούν ότι μια οντότητα-πελάτης θα λάβει μια απάντηση, σε μια αίτηση που έχει στείλει σε μια οντότητα-εξυπηρετητή, προτού αποχωρήσει από το δικτυακό περιβάλλον. Γενικά στην αρχιτεκτονική μας υποθέτουμε ότι ο χρόνος επικοινωνίας της χειρότερης περίπτωσης για την αποστολή ενός μηνύματος από την μία οντότητα στην άλλη, με την βοήθεια της οντότητας-συντονιστή, μπορεί να υπολογιστεί. Ο υπολογισμός αυτός βασίζεται στο μέγεθος του μηνύματος που πρόκειται να αποσταλεί, στον μέγιστο αριθμό οντοτήτων, καθώς και στα χαρακτηριστικά του δικτυακού πρωτοκόλλου που χρησιμοποιείται. Το μέγεθος των μηνυμάτων που αποστέλλονται μεταξύ των οντοτήτων είναι γνωστά στον αποτιμητή και στον χρονοπρογραμματιστή. Η δικτυακή υποδο-

μή του υπολογιστικού περιβάλλοντος των οντοτήτων βασίζεται στο πρότυπο IEEE 802.11 για ασύρματα LAN [4]. Το πρωτόκολλο αυτό παρέχει δύο θεμελιώδεις μηχανισμούς για την προσπέλαση του *μέσου (medium)*. Ο πρώτος ονομάζεται *Distributed Coordination Function (DCF)* και χειρίζεται την επαναμετάδοση των συγκρουόμενων πακέτων. Η επαναμετάδοση βασίζεται σε *δυαδικά εκθετικούς κανόνες υπαναχώρησης (binary exponential back-off rules)*. Ο χειρισμός της σύγκρουσης των πακέτων μέσω του DCF, καθιστά πολύ δύσκολο τον υπολογισμό του χρόνου αναμονής για την απόκτηση πρόσβασης στο μέσο. Συνεπώς, γίνεται πολύπλοκος και ο υπολογισμός του χρόνου αποστολής ενός μηνύματος μεταξύ των οντοτήτων.

Ο δεύτερος μηχανισμός που παρέχει το πρότυπο IEEE 802.11 ονομάζεται *Point Coordination Function (PCF)*. Ο μηχανισμός αυτός εγγυάται ότι οι αναμεταδόσεις των πακέτων θα γίνονται χωρίς συγκρούσεις και μέσα σε ένα συγκεκριμένο χρονικό διάστημα. Ωστόσο, απαιτείται η ύπαρξη μιας οντότητας που ονομάζεται *Point Coordinator (PC)* που εδρεύει σε ένα *σημείο πρόσβασης (access point)*. Η οντότητα-συντονιστής της αρχιτεκτονικής μας θα παίζει τον ρόλο του PC. Σύμφωνα με το πρότυπο IEEE 802.11, ο PC περιοδικά δίνει το δικαίωμα για την αναμετάδοση μηνυμάτων σε κάθε μία από τις οντότητες του δικτύου, δηλαδή υπάρχουν προκαθορισμένα *χρονικά διαστήματα (slots)* που γίνεται η εκπομπή των μηνυμάτων. Κατά τη διάρκεια αυτών των διαστημάτων οι κινητές οντότητες μπορούν να αναμεταδώσουν τα μηνύματά τους ή μέρος αυτών ανάλογα με το μέγεθος των μηνυμάτων. Συνεπώς, τα γνωρίσματα που επηρεάζουν τον χρόνο μετάδοσης ενός μηνύματος είναι τα εξής: ο *ρυθμός μετάδοσης (transmission rate)* του μέσου, ο χρόνος εκπομπής σε κάθε slot και το μέγεθος της απάντησης. Υποθέτουμε ότι η *καθυστέρηση διάδοσης (propagation delay)* είναι μηδενική.

Συνεπώς το κόστος, c_{rep}, αποστολής ενός μηνύματος στην χειρότερη περίπτωση, υπολογίζεται ως εξής: Ας υποθέσουμε ότι στο δίκτυο υπάρχουν το πολύ N οντότητες που ανταλλάσσουν μηνύματα μεταξύ τους. Στην χειρότερη περίπτωση, μια οντότητα θα χρειαστεί να περιμένει χρόνο $(N-1)* t_{slot}$ για να πάρει δικαίωμα εκπομπής. Έστω ότι το μέγεθος ενός μηνύματος που θέλει να στείλει μια οντότητα είναι s_{rep}, ο ρυθμός μετάδοσης του μέσου είναι R και ο διαθέσιμος χρόνος του κάθε slot είναι t_{slot}. Όπως έχει αναφερθεί, είναι πιθανόν να μην έχει μεταδοθεί ολόκληρο το μήνυμα σε ένα slot αν το μέγεθος του είναι μεγάλο, δηλ. ένα μήνυμα μπορεί να απαιτεί αρκετά slot για να μεταδοθεί. Ο αριθμός των slot που απαιτούνται για ένα μήνυμα θα είναι $n_s = (s_{rep} / R) / t_{slot}$. Λαμβάνοντας υπόψη ότι για κάθε slot, στο οποίο εκπέμπει κάποιο τμήμα του μηνύματος, η οντότητα θα πρέπει να περιμένει χρόνο $(N-1)* t_{slot}$, ο χρόνος που απαιτείται για την μετάδοση ολόκληρου του μηνύματος, στην χειρότερη περίπτωση, θα είναι:

$$c_{rep} = n_s * ((N-1) * t_{slot} + t_{slot}) = n_s * N * t_{slot}$$

Η αποτίμηση του c_{rep} για κάθε υπηρεσία μπορεί να γίνει μια φορά κατά την αρχικοποίηση του εξυπηρετητή που την προσφέρει, αν υποθέσουμε ότι το μέγεθος των μηνυμάτων που παράγονται από αυτήν είναι σταθερό.

ΚΕΦΑΛΑΙΟ 4. ΠΟΛΙΤΙΚΕΣ

Ο θεμελιώδης στόχος των πολιτικών που προτείνονται, είναι να εγγυηθούν ότι μια οντότητα-πελάτης θα λάβει απάντηση σε μια αίτηση που έστειλε σε μια οντότητα-εξυπηρετητή, προτού αποχωρήσει από το δίκτυο. Κάθε πολιτική απαιτεί διαφορετικούς υπολογιστικούς πόρους και παρέχει διαφορετικό επίπεδο εγγυήσεων.

4.1. Πολιτική Lifetime

Η πρώτη από τις πολιτικές βασίζεται αποκλειστικά στις διάρκειες ζωής των κινητών οντοτήτων. Ουσιαστικά θα λέγαμε ότι πρόκειται για μια γρήγορη ευρετική μέθοδο μιας και όπως θα δούμε στην συνέχεια δεν παρέχει επαρκείς εγγυήσεις για την ολοκληρωμένη εξυπηρέτηση μιας αίτησης.

Σύμφωνα με αυτήν την πολιτική οι χρονοπρογραμματιστές των οντοτήτων-εξυπηρετητών διατηρούν μια ουρά στην οποία αποθηκεύονται οι αιτήσεις. Κάθε αίτηση αντιστοιχεί σε μια συγκεκριμένη υπηρεσία που παρέχει ο εξυπηρετητής. Ο αλγόριθμος που χρησιμοποιείται για την χρονοδρομολόγηση των αιτήσεων είναι ο κλασσικός αλγόριθμος Round Robin (RR).

Κάθε νέα αίτηση r προέρχεται από ένα αντικείμενο της κινητής οντότητας-πελάτη, την οποία θα ονομάζουμε m_{client}, και σύμφωνα με την αρχιτεκτονική η αίτηση παραδίδεται στον αποτιμητή της οντότητας. Ο αποτιμητής στη συνέχεια θα πρέπει να επιλέξει έναν εξυπηρετητή, από όλους τους διαθέσιμους που παρέχουν την υπηρεσία που αντιστοιχεί στην συγκεκριμένη αίτηση, και να προωθήσει την αίτηση r στον επιλεγμένο εξυπηρετητή. Θα συμβολίζουμε με m_{server} μία κινητή οντότητα-εξυπηρετητή. Έτσι αφού έχει προωθηθεί η αίτηση από τον

αποτιμητή του πελάτη στον επιλεγμένο m_{server}, στη συνέχεια η υπηρεσία χρονοπρογραμματιστή του εξυπηρετητή τοποθετεί την αίτηση στην ουρά.

Ανάμεσα στο σύνολο των διαθέσιμων εξυπηρετητών που μπορούν να εξυπηρετήσουν την αίτηση του m_{client}, μπορεί να υπάρχουν εξυπηρετητές:

i. Με διάρκεια ζωής μεγαλύτερη από την διάρκεια ζωής του πελάτη.

ii. Με διάρκεια ζωής μικρότερη ή ίση με του πελάτη.

Αν ο αποτιμητής επιλέξει κάποιον εξυπηρετητή από την πρώτη κατηγορία τότε η αίτηση μπορεί να εξυπηρετηθεί μετά το τέλος της διάρκειας ζωής του πελάτη. Κάτι τέτοιο θα γίνει στην περίπτωση που ο εξυπηρετητής ολοκληρώσει την εκτέλεση της εργασίας, που αντιστοιχεί στην αίτηση, μέσα στο χρονικό όριο που καθορίζει η διάρκεια ζωής του αλλά σε χρόνο που είναι μετά την λήξη της διάρκειας ζωής του πελάτη. Από την άλλη πλευρά, αν ο αποτιμητής επιλέξει κάποιον εξυπηρετητή από τη δεύτερη κατηγορία τότε η αίτηση μπορεί να εξυπηρετηθεί μέσα στο χρονικό πλαίσιο που ορίζει η διάρκεια ζωής του πελάτη.

Και στις δύο περιπτώσεις όμως η αίτηση μπορεί να μην εξυπηρετηθεί καθόλου, ανάλογα με το πόσο φόρτο έχει ο εξυπηρετητής κάτι το οποίο δεν γνωρίζει ο πελάτης. Έτσι λαμβάνοντας υπόψη τα παραπάνω, αν υπάρχει ένας ή περισσότεροι εξυπηρετητές που έχουν διάρκεια ζωής μικρότερη ή ίση με αυτή του πελάτη τότε ένας από αυτούς επιλέγεται με τυχαίο τρόπο από τον αποτιμητή. Στην περίπτωση που δεν υπάρχει κανένας εξυπηρετητής με μικρότερη ή ίση διάρκεια ζωής τότε ο αποτιμητής επιλέγει τυχαία έναν από τους εξυπηρετητές που έχουν μεγαλύτερη διάρκεια ζωής. Πιο συγκεκριμένα, έστω ότι c_{rep} είναι ο χρόνος αποστολής ενός μηνύματος απάντησης στην χειρότερη περίπτωση. Επίσης με S συμβολίζουμε το σύνολο όλων των εξυπηρετητών που βρίσκονται στο δίκτυο και με τους οποίους μπορεί να επικοινωνήσει ο m_{client}. Θεωρούμε δύο *ξένα (disjoint)* υποσύνολα του S, το S' και το S''. Για τα δύο αυτά υποσύνολα ισχύει ότι $S', S'' \subseteq S \mid S' \cup S'' = S$ και ορίζονται ως εξής:

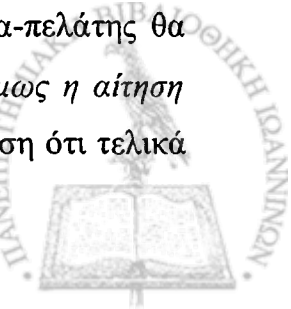
$$S' = \{ m_{server} \in S \mid m_{server}Lifetime \leq m_{client}Lifetime - c_{rep} \}$$

$$S'' = S - S'$$

Τότε για τον επιλεγμένο εξυπηρετητή m_{server} θα έχουμε:

$$\begin{cases} \text{Εάν } S' \neq \varnothing \text{ τότε } & m_{server} \in S' \\ \text{αλλιώς} & m_{server} \in S'' \end{cases}$$

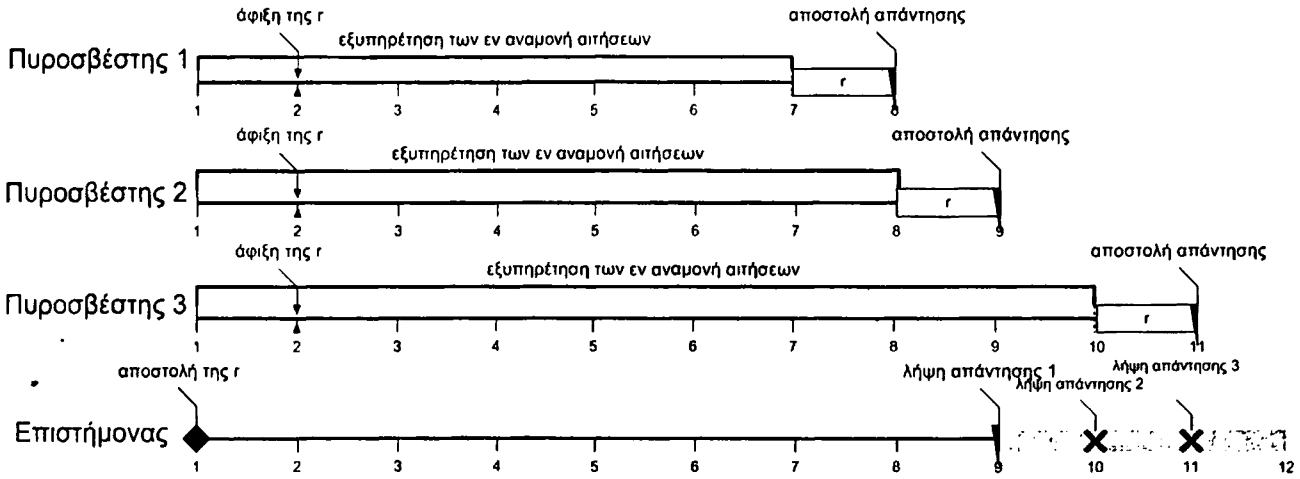
Συνεπώς εάν $m_{server} \in S'$ τότε η πολιτική Lifetime εγγυάται ότι η οντότητα-πελάτης θα λάβει απάντηση μέσα στο χρονικά πλαίσιο της διάρκειας ζωής του *εφόσον όμως η αίτηση εξυπηρετηθεί από τον εξυπηρετητή*. Ωστόσο δεν υπάρχει καμία απολύτως εγγύηση ότι τελικά



θα εξυπηρετηθεί η αίτηση.

Η πολιτική Lifetime είναι πολύ απλή καθώς το κόστος επικοινωνίας που επιφέρει είναι πολύ μικρό και περιλαμβάνει μόνο την αποστολή του μηνύματος της αίτησης από τον πελάτη και την λήψη του μηνύματος της απάντησης από τον εξυπηρετητή. Συνεπώς η πολιτική αυτή απαιτεί ελάχιστη πληροφορία και γνώση σχετικά με την συμπεριφορά των διαφορετικών κινητών οντοτήτων που αποτελούν το δίκτυο.

Θεωρώντας το παράδειγμα με το πυρηνικό εργοστάσιο, στο Κεφάλαιο 1, ας υποθέσουμε ότι ένας επιστήμονας εισέρχεται σε ένα εργαστήριο του εργοστασίου και θέλει να μάθει από κάποιον πυροσβέστη την τρέχουσα τιμή της ραδιενέργειας. Στο ίδιο εργαστήριο υπάρχουν 3 διαθέσιμοι πυροσβέστες και ο επιστήμονας θα πρέπει να επιλέξει έναν από αυτούς για να στείλει την αίτηση του. Ας υποθέσουμε ότι η διάρκεια ζωής του επιστήμονα είναι 9 χρονικές μονάδες και των πυροσβεστών 8, 9 και 11 μονάδες αντίστοιχα. Το κόστος για την αποστολή των μηνυμάτων είναι 1 χρονική μονάδα.



Σχήμα 4.1 Εφαρμόζοντας την Πολιτική Lifetime.

Στο Σχήμα 4.1 απεικονίζονται και τα τρία διαφορετικά σενάρια που αντιστοιχούν στις περιπτώσεις που ο επιστήμονας επιλέγει τον πρώτο, τον δεύτερο και τον τρίτο πυροσβέστη αντίστοιχα.

Σύμφωνα με την πολιτική Lifetime ο επιστήμονας θα επιλέξει τον πρώτο πυροσβέστη. Εάν ο πυροσβέστης καταφέρει να εξυπηρετήσει την αίτηση του επιστήμονα, αυτό θα γίνει στην χειρότερη περίπτωση, την χρονική στιγμή 8. Συνεπώς, επειδή το κόστος για την αποστολή της απάντησης είναι 1 μονάδα, ο επιστήμονας θα λάβει την απάντηση την χρονική στιγμή 9.

Από την άλλη πλευρά αν ο επιστήμονας επέλεγε τον δεύτερο πυροσβέστη τότε στην χειρότερη περίπτωση η αίτηση θα εξυπηρετηθεί την χρονική στιγμή 9 και έτσι θα λάβει απάντη-

ση τη χρονική στιγμή 10, κάτι που δεν είναι μέσα στα χρονικά πλαίσια της διάρκειας ζωής του επιστήμονα. Παρόμοια θα είναι και η κατάληξη εάν ο επιστήμονας επιλέξει τον τρίτο πυροσβέστη. Στην χειρότερη περίπτωση η αίτηση θα εξυπηρετηθεί τη χρονική στιγμή 10 και η απάντηση θα φτάσει τη χρονική στιγμή 11.

4.2. Πολιτική LifetimeLoad

Η δεύτερη πολιτική παρέχει ισχυρότερες εγγυήσεις σε σχέση με την πολιτική Lifetime. Όπως και στην πολιτική Lifetime, ο χρονοπρογραμματιστής του εξυπηρετητή διατηρεί μια ουρά και η χρονοδρομολόγιηση βασίζεται στον αλγόριθμο RR. Ωστόσο στην πολιτική LifetimeLoad λαμβάνονται υπόψη οι διάρκειες ζωής των οντοτήτων αλλά και ο τρέχων φόρτος του κάθε εξυπηρετητή.

Όπως και πριν, μια νέα αίτηση r προέρχεται από ένα αντικείμενο μιας κινητής οντότητας-πελάτη m_{client}, και η αίτηση παραδίδεται στον αποτιμητή της οντότητας. Ο αποτιμητής στη συνέχεια επιλέγει τυχαία έναν εξυπηρετητή, από όλους τους διαθέσιμους που παρέχουν την υπηρεσία που αντιστοιχεί στην συγκεκριμένη αίτηση, και προωθεί την αίτηση r στον επιλεγμένο εξυπηρετητή. Μόλις ο χρονοπρογραμματιστής του εξυπηρετητή λάβει την αίτηση r, ελέγχει εάν είναι εφικτή η χρονοδρομολόγησή της βάσει του φόρτου και της διάρκειας ζωής του m_{server}.

Πιο συγκεκριμένα, ας υποθέσουμε ότι ο συνολικός αριθμός των αιτήσεων εν αναμονή, για τον m_{server} είναι Nm_{server}, συμπεριλαμβανόμενης και της νέας αίτησης r. Με $Creq_i$ συμβολίζουμε τις απαιτούμενες μονάδες χρόνου για την εξυπηρέτηση της κάθε αίτησης req_i που βρίσκεται στην ουρά και που έχει αποσταλεί από έναν πελάτη m_{client_i}. Επιπρόσθετα, έστω ότι $crep_i$ είναι η χρόνος αποστολής ενός μηνύματος απάντησης στον m_{client_i}, στη χειρότερη περίπτωση. Η αίτηση r θα βρίσκεται στην θέση Nm_{server}. Κάθε νέα αίτηση που προστίθεται στην ουρά του χρονοπρογραμματιστή εισάγει μια επιπλέον καθυστέρηση στην ολοκλήρωση της εκτέλεσης όλων των άλλων αιτήσεων που βρίσκονται στην ουρά. Αυτό προκύπτει λόγω της φύσης του αλγορίθμου RR που εξυπηρετεί τις αιτήσεις εκ περιτροπής.

Για αυτό το λόγο, ο χρονοπρογραμματιστής θα πρέπει να εξασφαλίσει, πριν προσθέσει μια νέα αίτηση στην ουρά, πως η επιπλέον καθυστέρηση που εισάγεται δεν θα καθυστερήσει τις υπόλοιπες αιτήσεις τόσο ώστε να μην εξυπηρετηθούν έγκαιρα. Αν η επιπλέον καθυστέρηση είναι μεγάλη τότε οι αντίστοιχες απαντήσεις για τις εν αναμονή αιτήσεις της ουράς θα φτάσουν εκτός του χρονικού πλαισίου που καθορίζει η διάρκειας ζωής των πελατών που τις έστειλαν.

Επιπρόσθετα ο χρονοπρογραμματιστής θα πρέπει να ελέγξει εάν η νέα αίτηση θα εξυπηρετηθεί μέσα στο χρονικό πλαίσιο της διάρκειας ζωής του εξυπηρετητή και πως η απάντηση θα αποσταλεί στον πελάτη μέσα στα πλαίσια της διάρκειας ζωής του πελάτη. Έτσι για να επιτευχθούν οι παραπάνω στόχοι ο χρονοπρογραμματιστής εκτελεί τα παρακάτω βήματα:

i. Για κάθε αίτηση req_i, $i = 1,...,Nm_{server}$ (συμπεριλαμβανόμενης της νέας αίτησης r) ο χρονοπρογραμματιστής υπολογίζει τον συνολικό χρόνο $Dreq_i$ που απαιτείται για την ολοκλήρωση της εκτέλεσης της.

ii. Στη συνέχεια, για να αποδεχτεί ο χρονοπρογραμματιστής την νέα αίτηση πρέπει να ελέγξει ότι ισχύει η παρακάτω συνθήκη:

$$\forall req_i,\ i = 1,...,Nm_{server} \mid Dreq_i \le m_{client\,i}Lifetime - crep_i \text{ και}$$
$$Dreq_i \le m_{server}Lifetime$$

Μετά τον έλεγχο της παραπάνω συνθήκης ο χρονοπρογραμματιστής αποφασίζει εάν θα αποδεχθεί ή όχι τη νέα αίτηση. Εάν η συνθήκη ισχύει, τότε ο χρονοπρογραμματιστής αποδέχεται οριστικά την νέα αίτηση προσθέτοντας την στο χρονοπρόγραμμα. Όταν τελικά ολοκληρωθεί η εκτέλεση της νέας αίτησης, ο εξυπηρετητής θα αποστείλει πίσω στον πελάτη μια απάντηση με τα αποτελέσματα της εκτέλεσης. Στην αντίθετη περίπτωση, όταν δηλαδή δεν ισχύει η συνθήκη, ο εξυπηρετητής αποστέλλει μια αρνητική απάντηση στον πελάτη και ο χρονοπρογραμματιστής απορρίπτει την αίτηση r.

Και στις δύο περιπτώσεις ο αποτιμητής του m_{client} θα λάβει τελικά μια απάντηση από τον m_{server}. Στην περίπτωση που η απάντηση είναι αρνητική, ο αποτιμητής επιλέγει έναν άλλο εξυπηρετητή και ακολουθείται η ίδια διαδικασία. Εάν οι απαντήσεις όλων των διαθέσιμων εξυπηρετητών είναι αρνητικές τότε η αίτηση r απορρίπτεται από τον m_{client}, εφόσον δεν υπάρχει καμία διαθέσιμη οντότητα-εξυπηρετητής που να μπορεί να την εξυπηρετήσει.

Ο υπολογισμός του συνολικού χρόνου $Dreq_i$ που απαιτείται για την ολοκλήρωση της εκτέλεσης μιας αίτησης req_i γίνεται ως εξής: λόγω του ότι η αίτηση req_i βρίσκεται στην θέση i της ουράς και η χρονοδρομολόγησή της βασίζεται στον αλγόριθμο RR, απαιτούνται i χρονικές μονάδες για να τοποθετηθεί στο τέλος της ουράς. Αν η αίτηση req_i βρίσκεται ήδη στο τέλος της ουράς τότε δεν απαιτείται καμία χρονική μονάδα.

Θεωρούμε το $Q = \{A_{req_k} \mid k = 1,...,N'm_{server}\}$ να είναι οι εναπομένουσες μονάδες χρόνου που απαιτούνται για την εκτέλεση των εν αναμονή αιτήσεων τη στιγμή που η αίτηση req_i έχει τοποθετηθεί στο τέλος της ουράς. Ο αριθμός των στοιχείων του Q μπορεί να είναι μικρότερος από Nm_{server} λόγω του ότι κάποιες αιτήσεις της αρχικής ουράς μπορεί να έχουν ολοκληρωθεί

τη χρονική στιγμή που η req_i έχει τοποθετηθεί στο τέλος της ουράς. Για κάθε A_{reqk} ισχύει ότι $A_{reqk} \leq C_{reqk}$.

Ας υποθέσουμε πως θέλουμε να υπολογίσουμε τον συνολικό χρόνο που απαιτείται για την ολοκλήρωση της εκτέλεσης των A_{reqk} μονάδων. Έστω ότι $Q' = [B_j \mid j = 0,...,M]$ είναι μια ακολουθία της οποίας το $B_0 = 0$. Τα υπόλοιπα στοιχεία της Q' προκύπτουν από το Q, αφαιρώντας τις τυχόν πολλαπλές εμφανίσεις της ίδιας τιμής από το Q και ταξινομώντας τις εναπομένουσες τιμές A_i, για τις οποίες ισχύει $A_i \leq A_{reqk}$, σε αύξουσα σειρά. Για την Q' ισχύουν τα παρακάτω:

i. $B_0 = 0$

ii. $\forall B_i, B_j \in Q' \mid i < j \Rightarrow Bi < Bj$

iii. $\forall B_j > B_0$ υπάρχει τουλάχιστον ένα $A_{reqk} \in Q \mid A_{reqk} = B_j$

iv. $\forall A_i \leq A_{reqk} \in Q \mid (\exists B_j \in Q' \mid B_j = A_i)$

Για όλα τα $B_j \in Q'$ ορίζουμε την *πολλαπλότητα (multiplicity)* τους $mult_j$ ως εξής:

$$\begin{cases} mult_j = |\, Q_{Bj} \,|, j > 0 \\ mult_0 = 0, \quad j = 0 \end{cases}$$

όπου $Q_{Bj} \subseteq Q \wedge (\forall A_{reqk} \in Q_{Bj} \mid A_{reqk} = B_j)$.

Έστω ότι $B_l \in Q'$ είναι η τιμή που αντιστοιχεί στις μονάδες χρόνου που απαιτούνται για την ολοκλήρωση της αίτησης req_k, δηλ. $B_l = A_{reqk}$. Το B_l μπορεί να γραφτεί ως εξής:

$$B_l = B_1 + (B_2 - B_1) + (B_3 - B_2) + ... + (B_{l-1} - B_{l-2}) + (B_l - B_{l-1}) \qquad (1)$$

Οι B_1 μονάδες θα εκτελεστούν σε $|Q| * B_1$ μονάδες χρόνου. Μετά από $|Q| * B_1$ μονάδες χρόνου, όλες οι αιτήσεις που απαιτούσαν B_1 μονάδες για να ολοκληρωθούν θα αφαιρεθούν από την ουρά. Έτσι το μήκος της ουράς του χρονοπρογραμματιστή θα γίνει $|Q| - mult_1$. Επιπρόσθετα, όλες οι αιτήσεις που απαιτούσαν B_2 μονάδες χρόνου για να ολοκληρώσουν την εκτέλεση τους θα απαιτούν τώρα $B_2 - B_1$ μονάδες. Συνεπώς, οι επόμενες $B_2 - B_1$ μονάδες του αθροίσματος (1) θα εκτελεστούν σε $(|Q| - mult_1) * (B_2 - B_1)$ μονάδες χρόνου. Στη συνέχεια, το μήκος της ουράς θα γίνει $(|Q| - mult_1 - mult_2)$ και οι επόμενες $B_3 - B_2$ μονάδες του αθροίσματος (1) θα εκτελεστούν σε $(|Q| - mult_1 - mult_2) * (B_3 - B_2)$. Συνεπώς η εκτέλεση των τελευταίων μονάδων του αθροίσματος (1), θα απαιτεί τις παρακάτω μονάδες χρόνου:

$$(|Q| - \sum_{m=0}^{l-1} mult_m) \cdot (B_l - B_{l-1})$$

Συνεπώς, αθροίζοντας όλες τις απαιτούμενες χρονικές μονάδες του αθροίσματος (1), μπορούμε να καταλήξουμε στο ότι ο χρόνος $Dreq_i$ μπορεί να υπολογιστεί χρησιμοποιώντας τον παρακάτω τύπο:

$$D_{req_i} = i + \sum_{k=1}^{l} [\ (|Q| - \sum_{m=0}^{k-1} mult_m) \cdot (B_k - B_{k-1})]$$

Για παράδειγμα, έστω ότι μία νέα αίτηση r με $C_r = 4$ καταφθάνει στον εξυπηρετητή. Εκείνη τη χρονική στιγμή η ουρά του χρονοπρογραμματιστή είναι [2, 8, 3], δηλαδή πρώτη είναι μια αίτηση που απαιτεί χρόνο 3. Με την προσθήκη της r η ουρά θα γίνει [4, 2, 8, 3] και έτσι θα πρέπει να υπολογίσουμε τον συνολικό χρόνο που απαιτείται για την ολοκλήρωση της εκτέλεσης κάθε αίτησης. Ας υπολογίσουμε τον χρόνο για την νέα αίτηση r: Η αίτηση r βρίσκεται στην τέταρτη θέση, η οποία είναι και η τελευταία στην ουρά, συνεπώς δεν θα απαιτηθεί καμία χρονική μονάδα για να τοποθετηθεί στο τέλος της ουράς. Άρα

$$Q = [\underset{A_4}{4}, \underset{A_3}{2}, \underset{A_2}{8}, \underset{A_1}{3}] \quad και \quad |Q| = 4$$

Στην συνέχεια ταξινομούμε την Q και αφαιρούμε τις πολλαπλές εμφανίσεις τιμών οπότε προκύπτει η

$$Q' = [\underset{B_4}{8}, \underset{B_3}{4}, \underset{B_2}{3}, \underset{B_1}{2}] \quad με \quad B_0 = 0$$

Η τιμή B_3 αντιστοιχεί στην A_4 έτσι έχουμε $l = 3$. Για καθένα από τα παραπάνω B_j η πολλαπλότητα είναι 1 μιας και δεν υπήρχε καμία πολλαπλή εμφάνιση τιμής στην Q. Συνεπώς εφαρμόζοντας τον τύπο που υπολογίζει το $Dreq_4$ έχουμε

$$D_{req_4} = 0 + \sum_{k=1}^{3} [\ (|Q| - \sum_{m=0}^{k-1} mult_m) \cdot (B_k - B_{k-1})] =$$

$$0 + [(4-0)(B_1 - B_0) + (4-1)(B_2 - B_1) + (4-2)(B_3 - B_2)] = 4 \cdot 2 + 3 \cdot 1 + 2 \cdot 1 = 13$$

Έτσι η αίτηση r θα ολοκληρώσει την εκτέλεση της σε 13 μονάδες από την άφιξη της στον εξυπηρετητή.

Ας υπολογίσουμε τώρα τον χρόνο για την πρώτη αίτηση ($i = 1$) στην ουρά, την req_1: Για την τοποθέτηση της στην τελευταία θέση στην ουρά, θα απαιτηθεί 1 χρονική μονάδα. Άρα

$$Q = [\underset{A_4}{2}, \underset{A_3}{4}, \underset{A_2}{2}, \underset{A_1}{8}] \quad και \quad |Q| = 4$$

Στην συνέχεια ταξινομούμε την Q και αφαιρούμε τις πολλαπλές εμφανίσεις τιμών οπότε προκύπτει η

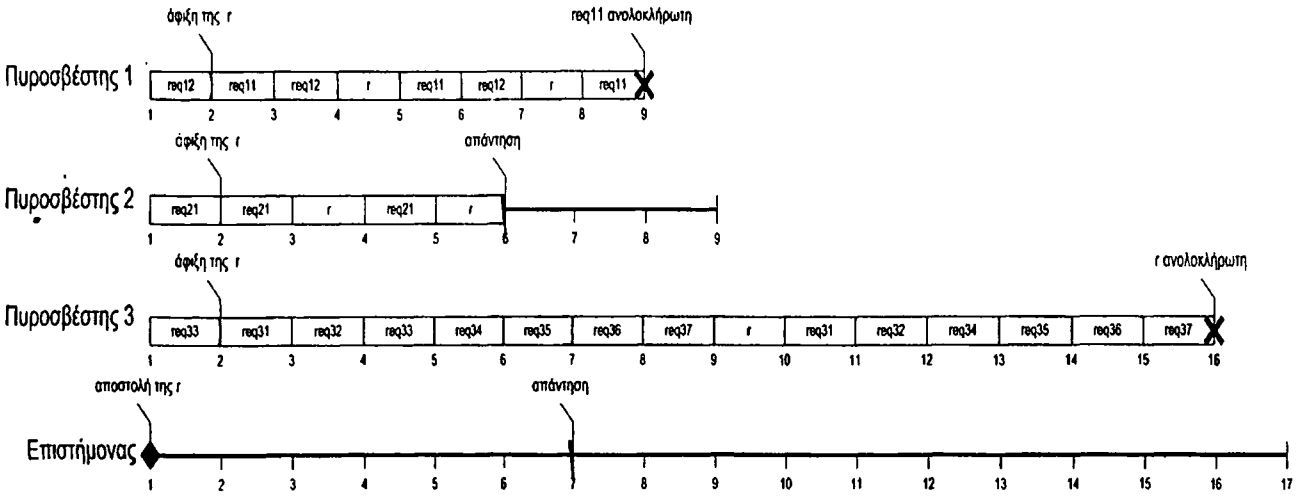
$$Q' = \underset{B_3\ B_2\ B_1}{[8, 4, 2]} \quad με \quad B_0 = 0$$

Η τιμή B_1 αντιστοιχεί στην A_4 έτσι έχουμε $l = 1$. Για το B_1 η πολλαπλότητα είναι 2, ενώ για καθένα από τα υπόλοιπα B_j η πολλαπλότητα είναι 1. Συνεπώς εφαρμόζοντας τον τύπο που υπολογίζει το $Dreq_1$ έχουμε

$$D_{req_1} = 1+\sum_{k=1}^{1}[\,(|Q| - \sum_{m=0}^{k-1} mult_m)\cdot(B_k - B_{k-1})] =$$

$$1+[(4-0)(B_1-B_0)] = 1+4\cdot 2 = 9$$

Έτσι η αίτηση req_1 θα ολοκληρώσει την εκτέλεση της σε 9 μονάδες αν προστεθεί τελικώς η νέα αίτηση r. Αν δεν είχε έρθει η αίτηση r στον εξυπηρετητή, η αίτηση req_1 θα ολοκλήρωνε την εκτέλεση της σε 7 μονάδες. Συνεπώς η προσθήκη της αίτησης r θα προκαλέσει καθυστέρηση 2 χρονικών μονάδων στην req_1.



Σχήμα 4.2 Εφαρμογή της Πολιτικής LifetimeLoad.

Στο Σχήμα 4.2, απεικονίζεται η εφαρμογή της πολιτικής LifetimeLoad στο παράδειγμα με το πυρηνικό εργοστάσιο. Όπως και προηγουμένως, υποθέτουμε ότι ο επιστήμονας θέλει να επιλέξει έναν πυροσβέστη, από τους τρεις διαθέσιμους, που μπορεί να εξυπηρετήσει την αίτηση του μέσα στο χρονικό πλαίσιο που καθορίζει η διάρκεια ζωής του επιστήμονα. Αυτή τη φορά οι διάρκειες ζωής των πυροσβεστών είναι 9, 9 και 16 μονάδες, αντίστοιχα. Έστω ότι ο χρόνος εκτέλεσης της χειρότερης περίπτωσης για την r είναι 2 μονάδες και το κόστος επικοινωνίας κατά τη χειρότερη περίπτωση είναι 1 μονάδα, δηλαδή έχουμε ότι $C_r = 2$ και $c_{rep} = 1$.

Ας υποθέσουμε ότι ο αποτιμητής του πελάτη επιλέγει πρώτα τον Πυροσβέστη 1. Ο χρονοπρογραμματιστής του πυροσβέστη έχει δύο αιτήσεις στην ουρά του. Την στιγμή που η αίτηση r καταφθάνει, η req_{11} απαιτεί 4 ακόμα μονάδες χρόνου για να ολοκληρώσει ενώ η req_{12} απαιτεί 2 μονάδες. Με την προσθήκη της r στην ουρά, ο συνολικός χρόνος για την ολοκλήρωσή της θα είναι 6 μονάδες. Λόγω του ότι η αίτηση r έφτασε τη χρονική στιγμή 2, τελικά η εκτέλεση της θα ολοκληρωθεί τη χρονική στιγμή 8. Συνεπώς, ο επιστήμονας θα λάβει απάντηση τη χρονική στιγμή 9, δηλαδή μέσα στον χρόνο της διάρκειας ζωής του. Ωστόσο, με την προσθήκη της r, η συνολική καθυστέρηση για την ολοκλήρωση της αίτησης req_{11} θα είναι 8 μονάδες χρόνου. Έτσι η req_{11} θα ολοκληρώσει την εκτέλεση της τελικά τη χρονική στιγμή 10, που όμως είναι μεγαλύτερη από τον χρόνο διάρκειας ζωής του πρώτου πυροσβέστη. Συνεπώς, η αποδοχή της νέας αίτησης r θα προκαλέσει το χάσιμο της προθεσμίας για την req_{11}. Κάτι τέτοιο προφανώς δεν είναι αποδεκτό και για αυτό η πολιτική LifetimeLoad δεν θα επιτρέψει στον χρονοπρογραμματιστή του πρώτου πυροσβέστη να αποδεχτεί την αίτηση r και να την προσθέσει στο χρονοπρόγραμμα.

Ας υποθέσουμε τώρα ότι ο επιστήμονας επιλέγει τον Πυροσβέστη 2 πρώτα. Η ουρά του χρονοπρογραμματιστή του περιέχει μόνο μια αίτηση που απαιτεί 2 μονάδες χρόνου για να ολοκληρωθεί. Η καθυστέρηση για την ολοκλήρωση των αιτήσεων r και req_{21} θα είναι 4 και 3, αντίστοιχα. Έτσι η εκτέλεση των αιτήσεων r και req_{21} θα ολοκληρωθεί τις χρονικές στιγμές 6 και 5, αντίστοιχα. Αυτές οι χρονικές στιγμές ανήκουν μέσα στον χρόνο διάρκειας ζωής του δεύτερου πυροσβέστη αλλά και του επιστήμονα. Συνεπώς, η αίτηση r θα γίνει αποδεκτή και ο επιστήμονας θα λάβει απάντηση την χρονική στιγμή 7.

Τέλος στο Σχήμα 4.2 απεικονίζεται και η περίπτωση που ο επιστήμονας επιλέγει τον τρίτο πυροσβέστη. Όπως φαίνεται ο τρίτος πυροσβέστης έχει μεγάλο φόρτο και κατά συνέπεια ο διαθέσιμος χρόνος του δεν επαρκεί για να εξυπηρετηθεί η νέα αίτηση r. Οπότε αυτή η περίπτωση δεν είναι αποδεκτή.

4.3. Πολιτική FIFO

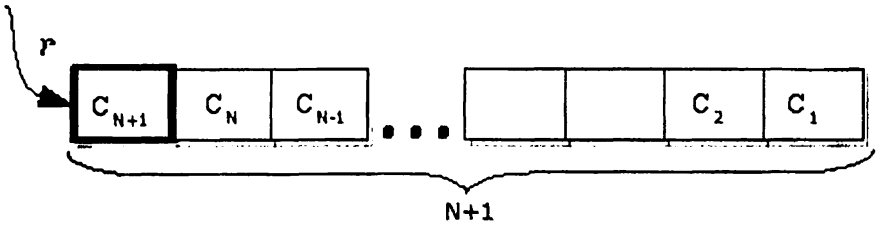
Η πολιτική FIFO είναι παρόμοια με την πολιτική LifetimeLoad υπό την έννοια ότι λαμβάνει υπόψη της το φόρτο του εξυπηρετητή και έτσι παρέχει ισχυρές εγγυήσεις. Όπως και στην πολιτική LifetimeLoad, ο χρονοπρογραμματιστής διατηρεί μια ουρά για τις αιτήσεις. Η διαφορά όμως σε αυτήν την πολιτική είναι ότι ο αλγόριθμος που χρησιμοποιείται είναι FIFO και όχι Round Robin όπως είναι στην πολιτική LifetimeLoad.

Όπως και με την προηγούμενη πολιτική, μια νέα αίτηση r προέρχεται από ένα αντικείμενο

μιας κινητής οντότητας-πελάτη m_{client} και η αίτηση παραδίδεται στον αποτιμητή της οντότητας. Ο αποτιμητής στη συνέχεια επιλέγει τυχαία έναν εξυπηρετητή, από όλους τους διαθέσιμους που παρέχουν την υπηρεσία που αντιστοιχεί στην συγκεκριμένη αίτηση, και προωθεί την αίτηση r στον επιλεγμένο εξυπηρετητή. Μόλις ο χρονοπρογραμματιστής του εξυπηρετητή λάβει την αίτηση r, ελέγχει εάν είναι εφικτή η χρονοδρομολόγησή της βάσει του φόρτου και της διάρκειας ζωής του m_{server}.

Επειδή ο αλγόριθμος χρονοπρογραμματισμού είναι FIFO, κάθε φορά θα εκτελείται η αίτηση που βρίσκεται στην πρώτη θέση της ουράς. Η εκτέλεση της πρώτης αίτησης στην ουρά θα πραγματοποιείται αδιάλειπτα μέχρι την ολοκλήρωσή της. Συνεπώς, ο αλγόριθμος χρονοπρογραμματισμού είναι μη-προεκχωρητικός.

Ας υποθέσουμε ότι ο συνολικός αριθμός των αιτήσεων στην ουρά του χρονοπρογραμματιστή, πριν την άφιξη της νέας αίτησης r, είναι N. Η νέα αίτηση θα προστεθεί στο τέλος της ου-



Σχήμα 4.3 Προσθήκη της Νέας Αίτησης στην Ουρά.

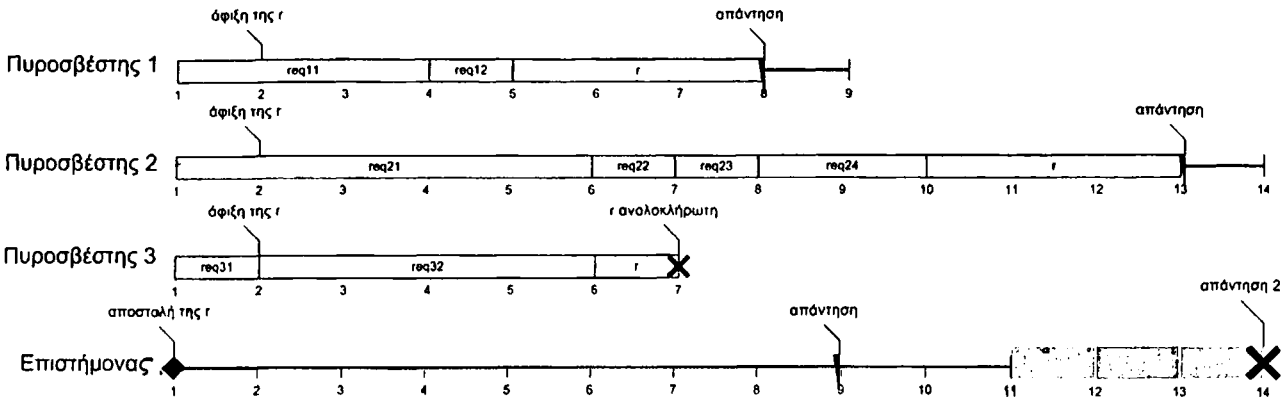
ράς, όπως φαίνεται και στο Σχήμα 4.3. Επειδή ο αλγόριθμος είναι FIFO και μη-προεκχωρητικός, η αίτηση r θα χρειαστεί να περιμένει να ολοκληρωθούν οι N αιτήσεις που προηγούνται, έτσι ώστε να φτάσει αυτή στην πρώτη θέση της ουράς και να εκτελεστεί. Συνεπώς, ο χρόνος αναμονής D_r για την r θα είναι:

$$D_r = \sum_{i=1}^{N} C_i$$

Συνολικά, η αίτηση r θα χρειαστεί χρόνο $L_r = D_r + C_r$ για την ολοκλήρωση της εκτέλεσης της. Ο χρονοπρογραμματιστής θα πρέπει να ελέγξει εάν η νέα αίτηση θα εξυπηρετηθεί μέσα στο χρονικό πλαίσιο της διάρκειας ζωής του εξυπηρετητή και ότι η απάντηση θα αποσταλεί στον πελάτη μέσα στη διάρκεια ζωής του πελάτη. Έτσι, για να επιτευχθούν οι παραπάνω στόχοι ο χρονοπρογραμματιστής θα πρέπει να ελέγξει την παρακάτω συνθήκη:

$$L_r \leq m_{server}Lifetime \text{ και } L_r \leq m_{client}Lifetime - c_{rep}$$

Εάν η παραπάνω συνθήκη ισχύει τότε ο χρονοπρογραμματιστής θα προσθέσει την νέα αίτηση στο χρονοπρόγραμμα, διαφορετικά θα αποστείλει μια αρνητική απάντηση στον πελάτη.



Σχήμα 4.4 Εφαρμογή της Πολιτικής FIFO.

Στο Σχήμα 4.4 απεικονίζεται η εφαρμογή της πολιτικής FIFO στο παράδειγμα μας με το πυρηνικό εργοστάσιο. Ο επιστήμονας θέλει να επιλέξει έναν πυροσβέστη, από τους τρεις διαθέσιμους, που μπορεί να εξυπηρετήσει την αίτηση του μέσα στο χρονικό πλαίσιο που καθορίζει η διάρκεια ζωής του. Αυτή τη φορά οι διάρκειες ζωής των πυροσβεστών είναι 9, 15 και 7 χρονικές μονάδες, αντίστοιχα. Έστω ότι ο χρόνος εκτέλεσης της χειρότερης περίπτωσης για την r είναι 3 μονάδες και το κόστος επικοινωνίας κατά τη χειρότερη περίπτωση είναι 1 μονάδα, δηλαδή έχουμε ότι $C_r = 3$ και $c_{rep} = 1$.

Ας υποθέσουμε ότι ο αποτιμητής του πελάτη επιλέγει πρώτα τον Πυροσβέστη 1. Ο χρονοπρογραμματιστής του πυροσβέστη έχει δύο αιτήσεις στην ουρά του. Η αίτηση r καταφθάνει τη χρονική στιγμή 2 και στην ουρά βρίσκονται πρώτη η req_{11}, που απαιτεί 2 ακόμα μονάδες χρόνου για να ολοκληρώσει, και δεύτερη η req_{12}, που απαιτεί 1 μονάδα. Η αίτηση r θα χρειαστεί να περιμένει 3 χρονικές μονάδες έως ότου ξεκινήσει η εκτέλεση της. Τη χρονική στιγμή 8 θα έχει ολοκληρωθεί η εκτέλεση της και ο επιστήμονας θα λάβει απάντηση τη χρονική στιγμή 9, δηλαδή μέσα στον χρόνο της διάρκειας ζωής του.

Στην περίπτωση που ο επιστήμονας επέλεγε τον Πυροσβέστη 2 τότε, όπως φαίνεται στο σχήμα, η αίτηση r θα ολοκληρώνονταν τη χρονική στιγμή 13, δηλ. μέσα στην διάρκεια ζωής του πυροσβέστη, αλλά εκτός της διάρκειας ζωής του επιστήμονα. Από την άλλη, αν ο επιστήμονας επέλεγε τον Πυροσβέστη 2, τότε η διάρκεια ζωής του πυροσβέστη δεν θα ήταν επαρκής για να ολοκληρωθεί η αίτηση r. Συνεπώς, και οι δύο αυτές περιπτώσεις δεν είναι αποδεκτές από την πολιτική FIFO.

4.4. Πολιτική EDFTB

Όπως είδαμε προηγουμένως οι πολιτικές LifetimeLoad και FIFO παρέχουν ισχυρές εγγυήσεις. Ωστόσο, αυτές οι εγγυήσεις μπορούν να παρέχονται μόνο στην περίπτωση που οι αιτήσεις αντιστοιχούν σε απεριοδικές εργασίες, δηλαδή εργασίες που εκτελούνται μία φορά κατά τη διάρκεια ζωής του εξυπηρετητή. Στην πράξη όμως οι οντότητες-πελάτες μπορεί να απαιτούν την εκτέλεση μιας εργασίας, από τον εξυπηρετητή, παραπάνω από μία φορά. Δηλαδή οι πελάτες απαιτούν την εκτέλεση μιας περιοδικής εργασίας. Στο παράδειγμα μας, οι επιστήμονες μπορεί να απαιτούν από τους πυροσβέστες την μέτρηση της ραδιενέργειας με περιοδικό τρόπο. Κάτι τέτοιο μπορεί να είναι χρήσιμο στους επιστήμονες έτσι ώστε να παραχθεί μια σειρά μετρήσεων, που θα επιστραφούν πίσω σε αυτούς για να μπορούν να πραγματοποιήσουν πιο λεπτομερή και ακριβή ανάλυση. Για αυτού του είδους τις αιτήσεις, που προκαλούν την εκτέλεση περιοδικών εργασιών, οι πολιτικές LifetimeLoad και FIFO δεν μπορούν να παρέχουν καμία εγγύηση.

Η πολιτική EDFTB είναι κατάλληλη και για τους δύο τύπους αιτήσεων, δηλαδή και για τις αιτήσεις που αντιστοιχούν σε περιοδικές εργασίες αλλά και για τις τυπικές αιτήσεις που εξυπηρετούνται μία φορά και αντιστοιχούν σε απεριοδικές εργασίες. Από εδώ και στο εξής θα αποκαλούμε τις πρώτες *περιοδικές αιτήσεις* και τις τελευταίες *απεριοδικές αιτήσεις*. Όπως υποδηλώνεται και από το όνομα της πολιτικής, βασίζεται στους αλγορίθμους EDF και TB που μελετήσαμε εκτενώς στο δεύτερο κεφάλαιο. Οι παραπάνω αλγόριθμοι έχουν τροποποιηθεί για να προσαρμοστούν στις ανάγκες ενός περιβάλλοντος με κινητούς κόμβους.

Όπως έχουμε αναφέρει και στο δεύτερο κεφάλαιο, στο [9] εξετάζεται ένα τυπικό σύστημα πραγματικού χρόνου στο οποίο εκτελούνται μόνο περιοδικές εργασίες που φτάνουν δυναμικά και προστίθενται σε μια ουρά. Η εκτέλεση των εργασιών γίνεται με προεκχωρητικό τρόπο. Κάθε εργασία τ_i χαρακτηρίζεται από μία περίοδο T_i και κάθε στιγμιότυπο της πρέπει να ολοκληρώνει την εκτέλεση του μέσα στην περίοδο T_i. Συνεπώς η προθεσμία της εργασίας ισούται με την περίοδο. Επίσης κάθε εργασία τ_i χαρακτηρίζεται και από τον χρόνο εκτέλεσής της στην χειρότερη περίπτωση $C\tau_i$. Σύμφωνα με τον αλγόριθμο EDF, η εργασία που εκτελείται κάθε χρονική στιγμή είναι αυτή με την πιο κοντινή προθεσμία από όλες τις περιοδικές εργασίες που βρίσκονται στην ουρά του χρονοπρογραμματιστή. Στο [9] έχει αποδειχθεί ότι πρέπει να ισχύει η ακόλουθη ικανή και αναγκαία συνθήκη για να μπορεί να είναι εφικτό ένα σύνολο εργασιών:

$$U_P = \sum_{i=1...N} (C_{\tau_i}/T_i) \leq 1$$

Στο [18] εξετάζεται η περίπτωση του από κοινού χρονοπρογραμματισμού περιοδικών και απεριοδικών εργασιών σε ένα σύστημα πραγματικού χρόνου και προτείνεται ο αλγόριθμος

TB. Ο παράγοντας αξιοποίησης U του συστήματος χωρίζεται σε δύο παράγοντες, στον U_P και στον U_S. Ο παράγοντας U_P σχετίζεται με την εκτέλεση των περιοδικών εργασιών, ενώ ο U_S σχετίζεται με την εκτέλεση των απεριοδικών. Βάσει του U_S, αποδίδεται στις απεριοδικές εργασίες τa_i μια προθεσμία $d\tau a_i$. Η προθεσμία αυτή είναι η συντομότερη δυνατή και δεν θέτει σε κίνδυνο την ομαλή και έγκαιρη εκτέλεση των περιοδικών εργασιών. Συγκεκριμένα, η προθεσμία δίνεται από τον παρακάτω τύπο:

$$d_{\tau a_i} = max(r_k, d_{\tau a_i - 1}) + \frac{C_{\tau a_i}}{U_S}$$

Στον παραπάνω τύπο το r_k συμβολίζει τη χρονική στιγμή που καταφθάνει η εργασία τa_i. Το $d\tau a_{i-1}$ συμβολίζει την προθεσμία της απεριοδικής αίτησης η οποία είχε φτάσει πριν ακριβώς από την τa_i. Βάσει της προθεσμίας $d\tau a_i$, η εργασία τa_i δρομολογείται με τον αλγόριθμο EDF όπως κάθε περιοδική εργασία. Έτσι, λαμβάνοντας υπόψη όλα τα παραπάνω, ένα σύνολο από περιοδικές και απεριοδικές εργασίες μπορεί να χρονοπρογραμματιστεί εάν και μόνο εάν:

$$U_P + U_S \leq 1$$

Στη συνέχεια παρουσιάζονται αναλυτικά οι επεκτάσεις που έγιναν στους αλγορίθμους EDF και TB για την εφαρμογή τους σε ένα περιβάλλον κινητών κόμβων. Σύμφωνα λοιπόν με την πολιτική EDFTB, ο χρονοπρογραμματιστής των οντοτήτων-εξυπηρετητών χρησιμοποιεί τον αλγόριθμο EDF. Ωστόσο, η εφικτότητα χρονοπρογραμματισμού των περιοδικών και των απεριοδικών αιτήσεων περιλαμβάνει και επιπρόσθετους περιορισμούς που ελέγχονται από τον χρονοπρογραμματιστή κατά την άφιξη μιας νέας αίτησης r που προέρχεται από τον αποτιμητή μιας οντότητας-πελάτη.

Εάν η αίτηση r είναι περιοδική θα προκαλέσει την εκτέλεση μιας περιοδικής εργασίας στην πλευρά του εξυπηρετητή. Λόγω του ότι η διάρκεια ζωής τόσο του εξυπηρετητή όσο και του πελάτη είναι περιορισμένες, ο πελάτης είναι υποχρεωμένος να συσχετίσει την αίτηση r με μια περίοδο T_r και με έναν απαιτούμενο αριθμό εκτελέσεων n_r της περιοδικής εργασίας. Για παράδειγμα, ένας επιστήμονας θα ζητούσε από έναν πυροσβέστη να μετρήσει τα επίπεδα ραδιενέργειας 3 φορές με περίοδο 2 μονάδες χρόνου.

Η αίτηση r τελικά θα καταφτάσει στον χρονοπρογραμματιστή του m_{server}, ο οποίος υποθέτει ότι η συνολική αξιοποίηση U του εξυπηρετητή χωρίζεται σε U_P και U_S. Το U_P σχετίζεται με την συνολική αξιοποίηση των περιοδικών εργασιών και το U_S με των απεριοδικών. Ο στόχος του χρονοπρογραμματιστή είναι να επαληθεύσει την ισχύ των παρακάτω περιορισμών:

- Για τις περιοδικές αιτήσεις:
    i. Η r θα εξυπηρετηθεί n_r φορές κατά τη διάρκεια του χρόνου ζωής του εξυπηρε-

τητή.

ii. Οι απαντήσεις στην *r* θα αποσταλούν πίσω στον πελάτη μέσα στα χρονικά πλαίσια της διάρκειας ζωής του.

- Για τις απεριοδικές αιτήσεις:
  i. Η *r* θα εξυπηρετηθεί μέσα στα χρονικά πλαίσια της διάρκειας ζωής του εξυπηρετητή.
  ii. Η απάντηση στην *r* θα αποσταλεί πίσω στον πελάτη μέσα στα χρονικά πλαίσια της διάρκειας ζωής του.

Αν κάποιος από τους παραπάνω περιορισμούς δεν ισχύει, ο χρονοπρογραμματιστής αποστέλλει μια αρνητική απάντηση στον αποτιμητή του πελάτη. Σε διαφορετική περίπτωση ο χρονοπρογραμματιστής αποδέχεται οριστικά την νέα αίτηση και αποστέλλει την απάντηση (ή τις απαντήσεις, αν είναι περιοδική) με τα αποτελέσματα μόλις ολοκληρωθεί η εκτέλεση της αίτησης. Ανάλογα με την απάντηση ο αποτιμητής συμπεριφέρεται όπως στην περίπτωση των πολιτικών LifetimeLoad και FIFO, δηλαδή αν είναι αρνητική ξαναστέλνει την αίτηση σε έναν άλλον εξυπηρετητή.

4.4.1. Περιοδικές Αιτήσεις

Στην περίπτωση που η *r* είναι περιοδική, ο χρονοπρογραμματιστής του m_{server} εκτελεί τα παρακάτω βήματα:

. Πρώτα ελέγχει εάν η αξιοποίηση του m_{server} παραμένει χαμηλότερη από 1, στην περίπτωση που η νέα αίτηση γίνει αποδεκτή από τον χρονοπρογραμματιστή. Αν υπάρχουν Nm_{server} αιτήσεις στην ουρά του εξυπηρετητή, ο χρονοπρογραμματιστής επαληθεύει την ακόλουθη συνθήκη:

$$U_P = \sum_{i=1}^{N_{m_{server}}} \frac{C_{r_i}}{T_{r_i}} + \left(\frac{C_r}{T_r}\right) \leq 1 - U_S \qquad (1)$$

Στον παραπάνω τύπο τα Cr_i και Tr_i δηλώνουν αντίστοιχα τον χρόνο εκτέλεσης της χειρότερης περίπτωσης και την περίοδο κάθε αίτησης που βρίσκεται στην ουρά. Εάν αυτός ο τύπος ισχύει τότε η εκτέλεση της αίτησης *r* δεν θα διακινδυνεύσει την έγκαιρη εκτέλεση όλων των υπόλοιπων περιοδικών αιτήσεων που βρίσκονται στην ουρά του χρονοπρογραμματιστή.

Ωστόσο ο χρονοπρογραμματιστής θα πρέπει να ελέγξει επιπρόσθετα πως η διάρκεια ζωής του εξυπηρετητή είναι επαρκής για να εκτελεστεί η αίτηση *r* n_r φορές. Αυτός ο έλεγχος επιτυγχάνεται στο δεύτερο βήμα με την αποτίμηση της παρακάτω συνθήκης:

$$\left\lfloor \frac{m_{server} Lifetime}{T_r} \right\rfloor \geq n_r \qquad (2)$$

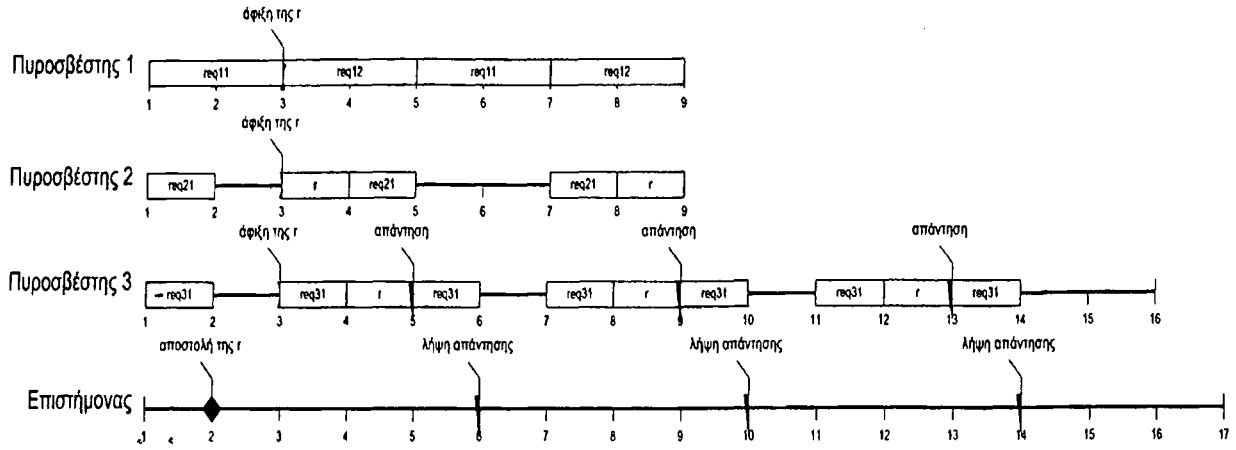
Από την παραπάνω διαίρεση κρατάμε μόνο το ακέραιο μέρος, μιας και ο αριθμός n_r είναι πάντα ακέραιος.

Έτσι, ο χρονοπρογραμματιστής του m_{server} στα δύο πρώτα βήματα ελέγχει εάν ο εξυπηρετητής μπορεί να εγγυηθεί τον πρώτο από τους δύο περιορισμούς για τις περιοδικές αιτήσεις, δηλαδή ότι η r θα εξυπηρετηθεί n_r φορές κατά τη διάρκεια του χρόνου ζωής του εξυπηρετητή. Το τρίτο βήμα που εκτελεί ο χρονοπρογραμματιστής ελέγχει τον δεύτερο περιορισμό που επιτάσσει πως οι απαντήσεις στην r θα αποσταλούν πίσω στον πελάτη μέσα στα χρονικά πλαίσια της διάρκειας ζωής του. Λόγω του ότι η εκτέλεση των εργασιών που αντιστοιχούν στις περιοδικές αιτήσεις γίνεται προεκχωρητικά, ο μόνος τρόπος για να διασφαλίσουμε ότι ο πελάτης θα λάβει την απάντηση για την r μέσα στον χρόνο της διάρκειας ζωής του είναι να ελέγξουμε εάν ο m_{client} έχει μεγαλύτερη διάρκεια ζωής από τον m_{server}. Δηλαδή θα πρέπει να ισχύει η παρακάτω συνθήκη:

$$m_{server} Lifetime \leq m_{client} Lifetime - c_{rep} \qquad (3)$$

Εάν ισχύουν όλα τα παραπάνω τότε ο χρονοπρογραμματιστής του εξυπηρετητή αποδέχεται οριστικά την νέα αίτηση. Στην αντίθετη περίπτωση ο εξυπηρετητής αποστέλλει μια αρνητική απάντηση και ο πελάτης επιλέγει έναν άλλον διαθέσιμο εξυπηρετητή για να στείλει την αίτηση. Εάν οι απαντήσεις από όλους τους διαθέσιμους εξυπηρετητές είναι αρνητικές τότε η αίτηση απορρίπτεται από τον πελάτη.

Στο παράδειγμα μας με το πυρηνικό εργοστάσιο, ας υποθέσουμε ότι τη χρονική στιγμή 2 ο επιστήμονας θέλει να στείλει μια περιοδική αίτηση με χρόνο εκτέλεσης $C_r = 1$ και περίοδο $T_r = 4$ σε έναν από τους τρεις πυροσβέστες όπως φαίνεται στο Σχήμα 4.5. Ο επιστήμονας, επιπρόσθετα, απαιτεί να εκτελεστεί η αίτηση r τρεις φορές, δηλαδή έχουμε $n_r = 3$. Στο συγκεκριμένο παράδειγμα υποθέτουμε ότι $U_S = 0$, δηλαδή δεν διατίθεται καθόλου χρόνος για την εξυπηρέτηση απεριοδικών αιτήσεων.



Σχήμα 4.5 Εφαρμογή της Πολιτικής EDFTB για Περιοδικές Αιτήσεις.

Ας υποθέσουμε ότι ο επιστήμονας επιλέγει αρχικά τον Πυροσβέστη 1. Ο Πυροσβέστης 1 έχει ήδη στην ουρά του δύο περιοδικές αιτήσεις, τις req_{11} και req_{12}, με $Creq_{11} = 2$, $Treq_{11} = 4$ και $Creq_{12} = 2$, $Treq_{12} = 4$, αντίστοιχα. Με την προσθήκη της r στην ουρά του Πυροσβέστη 1, ο παράγοντας αξιοποίησης U_P θα γίνει: $U_P = \left(\frac{2}{4}+\frac{2}{4}\right)+\frac{1}{4} = 1.25$. Έτσι, ο Πυροσβέστης 1 δεν μπορεί να προσθέσει στο χρονοπρόγραμμά του την νέα αίτηση r γιατί κάτι τέτοιο θα διακινδύνευε την έγκαιρη εκτέλεση των αιτήσεων.

Ας λάβουμε υπόψη την περίπτωση που ο επιστήμονας επιλέγει τον δεύτερο πυροσβέστη. Ο δεύτερος πυροσβέστης έχει ήδη στην ουρά του την περιοδική αίτηση req_{21} με χρόνο εκτέλεσης $Creq_{21} = 1$ και περίοδο $Treq_{21} = 3$. Με την προσθήκη της αίτησης r ο παράγοντας αξιοποίησης U_P θα γίνει 0,583 και συνεπώς η συνθήκη (1) ισχύει. Ωστόσο, η υπόλοιπη διάρκεια ζωής του πυροσβέστη τη χρονική στιγμή που λαμβάνει την αίτηση r είναι 6. Η αίτηση μπορεί να εκτελεστεί $\lfloor 6/3 \rfloor = 2$ φορές. Κατά συνέπεια, η συνθήκη (2) δεν ισχύει μιας και ο επιστήμονας απαιτεί να εκτελεστεί η r τρεις φορές.

Τέλος, ας θεωρήσουμε ότι ο επιστήμονας επιλέγει τον τρίτο πυροσβέστη για να στείλει την αίτηση του. Ο Πυροσβέστης 3 έχει στην ουρά του την αίτηση req_{31} με $Creq_{31} = 1$ και περίοδο $Treq_{31} = 2$. Με την άφιξη της r ο παράγοντας αξιοποίησης U_P γίνεται 0,75 και η συνθήκη (1) ισχύει. Επιπλέον, η διάρκεια ζωής του πυροσβέστη εκείνη τη στιγμή είναι 13 και συνεπώς μπορεί να εκτελέσει την r τρεις φορές μιας και ισχύει η συνθήκη (2). Υποθέτοντας ότι $c_{rep} = 1$, τότε ισχύει και η τρίτη συνθήκη και έτσι ο πυροσβέστης θα αποστείλει πίσω τις απαντήσεις μέσα στην διάρκεια ζωής του επιστήμονα.

4.4.2. Απεριοδικές Αιτήσεις

Εάν η αίτηση είναι απεριοδική, ο χρονοπρογραμματιστής ακολουθεί την προσέγγιση TB [18]. Βάσει του παράγοντα U_S, ο χρονοπρογραμματιστής αναθέτει μια προθεσμία στην r και ελέγχει εάν αυτή η προθεσμία είναι συνεπής με τις διάρκειες ζωής του πελάτη και του εξυπηρετητή. Η προθεσμία υπολογίζεται με βάση τον παρακάτω τύπο:

$$d_r = max(t_r, d_{r'}) + \frac{C_r}{U_S}$$

όπου

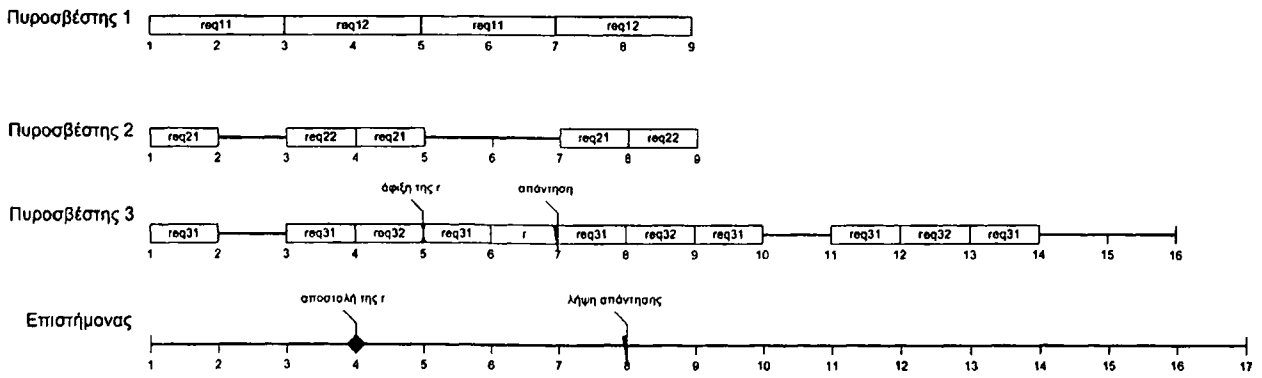
- t_r είναι ο χρόνος άφιξης της αίτησης r
- d_r' είναι η προθεσμία της απεριοδικής αίτησης r' η οποία είχε φτάσει πριν ακριβώς από την r.

Επιπρόσθετα θα πρέπει να ισχύει η παρακάτω σχέση έτσι ώστε η προθεσμία που θα υπολογιστεί να είναι συνεπής με τις διάρκειες ζωής του εξυπηρετητή και του πελάτη:

$$d_r \leq min(m_{server}\,Lifetime, (m_{client}\,Lifetime - c_{rep}))$$

Πιο συγκεκριμένα, η παραπάνω σχέση καθορίζει ότι ο εξυπηρετητής θα ολοκληρώσει την εκτέλεση της αίτησης μέσα στα χρονικά πλαίσια που καθορίζει η διάρκεια ζωής του, αλλά και ότι ο πελάτης θα λάβει πίσω την απάντηση πριν τελειώσει η διάρκεια ζωής του.

Στο Σχήμα 4.6, επανεξετάζουμε το σενάριο του σχήματος 4.5. Αυτή τη φορά όμως ο παράγοντας U_S είναι 0,25. Υποθέτουμε ότι ο Πυροσβέστης 3 έχει προσθέσει στο χρονοπρόγραμμα την προηγούμενη περιοδική αίτηση του επιστήμονα, την οποία συμβολίζουμε πλέον σαν req_{32} στο σχήμα.



Σχήμα 4.6 Εφαρμογή της Πολιτικής EDFTB για Περιοδικές και Απεριοδικές Αιτήσεις.

Ας υποθέσουμε τώρα ότι ο επιστήμονας στέλνει μια απεριοδική αίτηση r στον ίδιο πυροσβέστη. Για τον Πυροσβέστη 3 έχουμε ότι U_P = 0,75 και U_S = 0,25. Η αίτηση καταφτάνει

στον πυροσβέστη τη χρονική στιγμή 5 και ο χρόνος εκτέλεσης της είναι $C_r = 1$. Επειδή είναι η πρώτη απεριοδική αίτηση, της ανατίθεται προθεσμία ίση με 9, δηλ. $d_r = \max(5,0) + 1/0,25 = 9$. Εάν ο χρόνος αποστολής της απάντησης είναι 1, στην χειρότερη περίπτωση, τότε η αίτηση r θα γίνει αποδεκτή από τον τρίτο πυροσβέστη και ο επιστήμονας θα λάβει την απάντηση τη χρονική στιγμή 8.

ΚΕΦΑΛΑΙΟ 5. ΠΕΙΡΑΜΑΤΙΚΗ ΜΕΛΕΤΗ

Σε αυτό το κεφάλαιο παρουσιάζονται πειραματικά αποτελέσματα της επίδοσης των πολιτικών που παρουσιάστηκαν στο προηγούμενο κεφάλαιο. Οι πολιτικές υλοποιήθηκαν σε ένα περιβάλλον προσομοίωσης, χρησιμοποιώντας την βιβλιοθήκη προσομοίωσης PARASOL [13]. Τα **κριτήρια** που χρησιμοποιήθηκαν για την μέτρηση της επίδοσης των πολιτικών, είναι δύο:

i. Το πρώτο είναι το *κλάσμα του αριθμού των αιτήσεων που ολοκληρώθηκαν έγκαιρα, σύμφωνα δηλαδή με την διάρκεια ζωής τόσο του εξυπηρετητή όσο και του πελάτη, προς τον αριθμό των αιτήσεων που έγιναν αποδεκτές και προστέθηκαν στο χρονοπρόγραμμα του χρονοπρογραμματιστή.*
ii. Το δεύτερο κριτήριο είναι το *κλάσμα του αριθμού των αιτήσεων που ολοκληρώθηκαν έγκαιρα, προς τον συνολικό αριθμό των αιτήσεων που πραγματοποίησαν οι πελάτες σε όλους τους εξυπηρετητές.*

Στόχος όλων των συστημάτων πραγματικού χρόνου είναι να επιτυγχάνουν επίδοση 100% στο πρώτο κριτήριο, συνεπώς αυτός είναι και ο στόχος των πολιτικών μας. Το δεύτερο κριτήριο επιβεβαιώνει εάν οι πολιτικές μας είναι "λειτουργικές". Για παράδειγμα, θα μπορούσαμε να έχουμε 100% επίδοση στο πρώτο κριτήριο ακόμα και αν ο χρονοπρογραμματιστής αποδεχόταν 1 μόνο αίτηση, κάτι που δεν είναι λειτουργικό.

Για την αποτίμηση της επίδοσης των πολιτικών, πραγματοποιήθηκαν πέντε κατηγορίες πειραμάτων. Στις πρώτες τέσσερις κρατάμε σταθερό των αριθμό των εξυπηρετητών και εξετάζουμε το πώς επιδρά ο χρόνος εκτέλεσης των υπηρεσιών τα δύο κριτήρια. Συγκεκριμένα, χρησιμοποιήθηκαν 3 οντότητες εξυπηρετητών οι οποίες δέχονταν αιτήσεις από οντότητες-πελάτες. Στην πέμπτη κατηγορία, κρατάμε σταθερό τον χρόνο εκτέλεσης των υπηρεσιών και

μεταβάλλουμε τον αριθμό των εξυπηρετητών. Και στις πέντε κατηγορίες, οι εξυπηρετητές παρέχουν σημασιολογικά ισοδύναμες υπηρεσίες και έτσι οι πελάτες μπορούσαν να στείλουν τις αιτήσεις τους σε οποιονδήποτε εξυπηρετητή. Για να υπάρχει ομοιόμορφη κατανομή του φόρτου στους εξυπηρετητές, οι πελάτες επέλεγαν με τυχαίο τρόπο τον εξυπηρετητή στον οποίο θα στέλνανε μια αίτηση για εξυπηρέτηση. Η τυχαία αυτή επιλογή των εξυπηρετητών ακολουθούσε μια ομοιόμορφη κατανομή πιθανότητας. Η διάρκεια ζωής των κινητών οντοτήτων, οι χρόνοι εκτέλεσης των αιτήσεων και οι περίοδοι των περιοδικών αιτήσεων παράγονταν με τυχαίο τρόπο ακολουθώντας μια ομοιόμορφη κατανομή.

Στις παρακάτω ενότητες γίνεται σύγκριση της κάθε πολιτικής, με και χωρίς τις επιπρόσθετες συνθήκες που έχουν προταθεί στο Κεφάλαιο 4. Συγκεκριμένα, στην Ενότητα 5.1 γίνεται σύγκριση της πολιτικής Lifetime με τον αλγόριθμο RoundRobin, στην Ενότητα 5.2 γίνεται σύγκριση της πολιτικής LifetimeLoad με τον αλγόριθμο RoundRobin, στην Ενότητα 5.3 γίνεται σύγκριση της πολιτικής FIFO με την FIFO χωρίς τις επιπρόσθετες συνθήκες και τέλος στην Ενότητα 5.4 γίνεται σύγκριση της πολιτικής EDFTB με την πολιτική EDFTB χωρίς τις επιπρόσθετες συνθήκες.

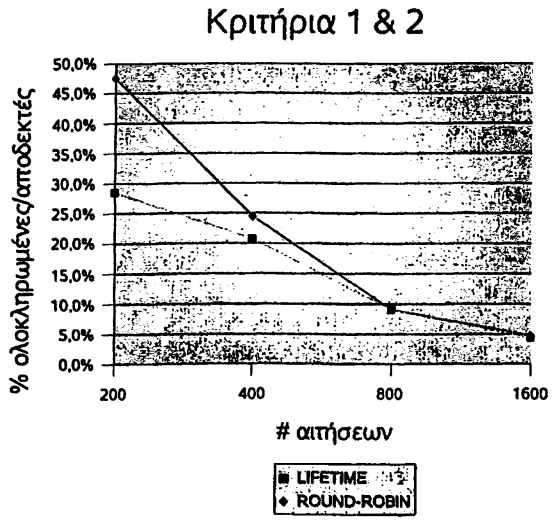
5.1. Σύγκριση της Πολιτικής Lifetime με τον Αλγόριθμο RR

Σε αυτήν την ενότητα γίνεται σύγκριση της πολιτικής Lifetime με τον αλγόριθμο Round Robin. Όπως αναφέρθηκε, οι παράμετροι των κινητών οντοτήτων παράγονταν με τυχαίο τρόπο ακολουθώντας μια ομοιόμορφη κατανομή. Πιο συγκεκριμένα, ο χρόνος εκτέλεσης της προσομοίωσης, *simRuntime*, ήταν 15000 μονάδες, το κβάντο των χρονοπρογραμματιστών ήταν 1 χρονική μονάδα και οι διάρκειες ζωής του εξυπηρετητή και του πελάτη έπαιρναν τιμές στο διάστημα (*simRuntime/10, simRuntime*) ακολουθώντας ομοιόμορφη κατανομή. Οι μετρήσεις πραγματοποιήθηκαν χρησιμοποιώντας τέσσερα διαφορετικά σύνολα: 200, 400, 800 και 1600 αιτήσεων. Για τις πρώτες τέσσερις από τις πέντε κατηγορίες μετρήσεων, τα διαστήματα από τα οποία έπαιρνε τιμές το C κάθε υπηρεσίας ήταν αντίστοιχα:

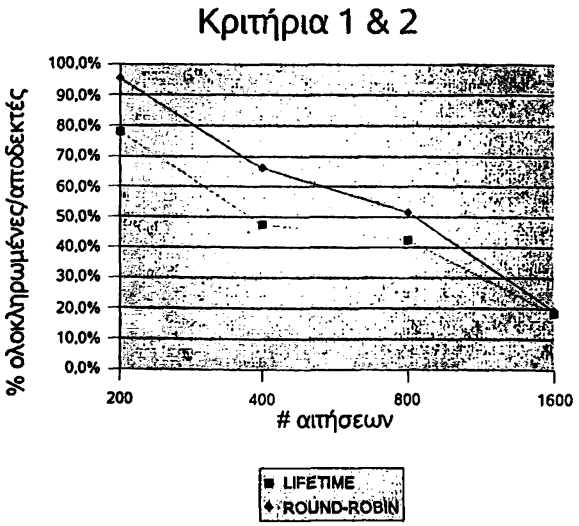
- (*1, clientLifetime / 40*) για την πρώτη κατηγορία.
- (*1, clientLifetime / 160*) για την δεύτερη.
- (*1, clientLifetime / 320*) για την τρίτη, και
- (*1, clientLifetime / 640*) για την τέταρτη.

Στα παρακάτω σχήματα γίνεται σύγκριση των δύο πολιτικών βάσει των δύο κριτηρίων που αναφέρθηκαν στην αρχή του κεφαλαίου. Λόγω του ότι οι δύο συγκεκριμένες πολιτικές κάνουν αποδεκτές όλες τις συνολικές αιτήσεις, τα δύο κριτήρια ταυτίζονται, δηλαδή ο αριθ-

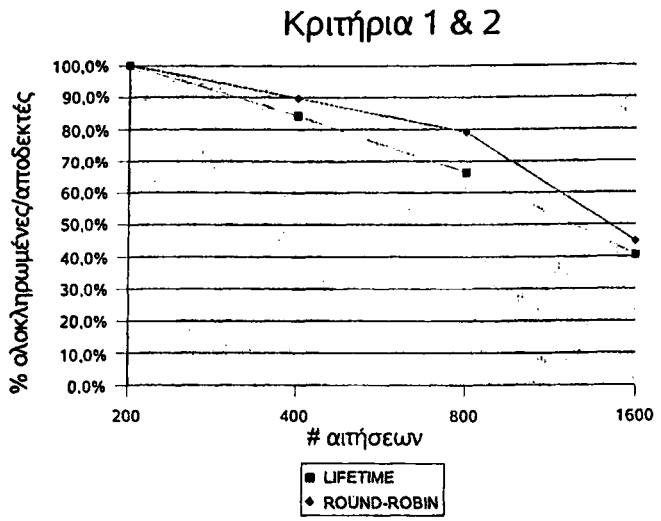
μός των αποδεκτών αιτήσεων είναι ίσος με τον αριθμό των συνολικών αιτήσεων.



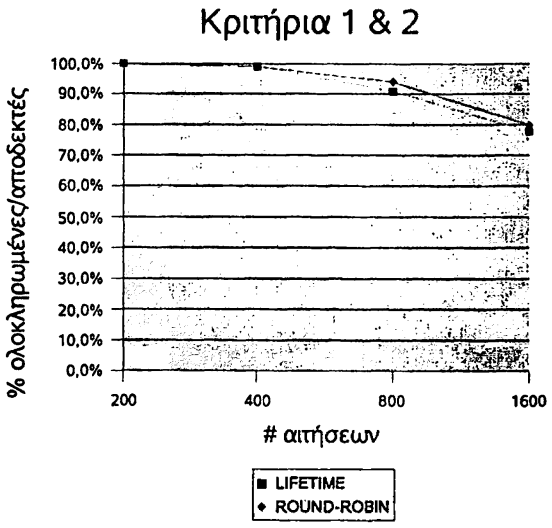
Σχήμα 5.1 Πρώτη Κατηγορία Μετρήσεων. $C \in (1, \frac{clientLifetime}{40})$



Σχήμα 5.2 Δεύτερη Κατηγορία Μετρήσεων. $C \in (1, \frac{clientLifetime}{160})$

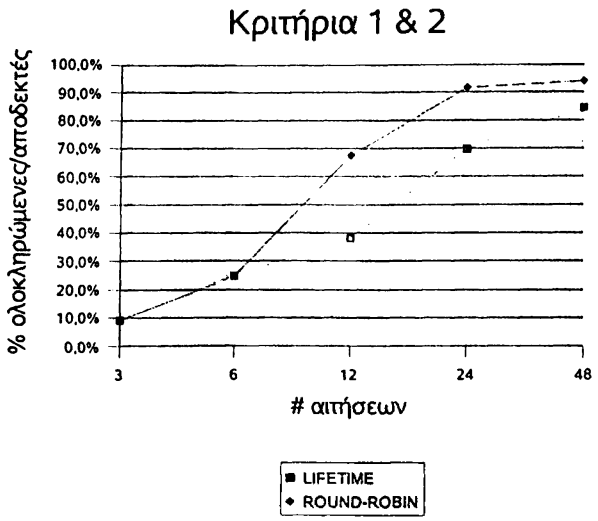


Σχήμα 5.3 Τρίτη Κατηγορία Μετρήσεων. $C \in (1, \frac{clientLifetime}{320})$



Σχήμα 5.4 Τέταρτη Κατηγορία Μετρήσεων. $C \in (1, \frac{clientLifetime}{640})$

Όπως παρατηρούμε στα παραπάνω σχήματα, η επίδοση των δύο κριτηρίων μειώνεται καθώς αυξάνεται ο αριθμός των αιτήσεων. Όσο μειώνεται ο χρόνος εκτέλεσης C, αυξάνεται το ποσοστό επίδοσης των δύο κριτηρίων. Αυτό προκύπτει διότι καθώς μειώνονται οι χρόνοι εκτέλεσης των υπηρεσιών, μειώνεται και ο φόρτος του κάθε εξυπηρετητή με αποτέλεσμα να μπορεί ο κάθε ένας να εξυπηρετεί επιτυχημένα περισσότερες αιτήσεις μέσα στο ίδιο χρονικό διάστημα.



Σχήμα 5.5 Σύγκριση με Αυξανόμενο Αριθμό Εξυπηρετητών.

Στο Σχήμα 5.5 φαίνεται η πέμπτη κατηγορία μετρήσεων. Σε αυτήν την περίπτωση ο αριθμός των εξυπηρετητών αυξάνεται: αντίστοιχα, έχουμε 3, 6, 12, 24 και 48 εξυπηρετητές στο σύστημα. Ο αριθμός των αιτήσεων που πραγματοποιούνται είναι 800 και το διάστημα από το οποίο παίρνει τιμές το C κάθε υπηρεσίας είναι (*1, clientLifetime / 40*). Όπως παρατηρούμε, η επίδοση των κριτηρίων αυξάνεται σχεδόν γραμμικά καθώς αυξάνεται ο αριθμός των εξυπηρετητών στο σύστημα. Αυτό συμβαίνει διότι καθώς αυξάνεται ο αριθμός των εξυπηρετητών μοιράζεται ο φόρτος του συστήματος. Συνεπώς, ο κάθε εξυπηρετητής εκτελεί λιγότερες υπηρεσίες στον ίδιο διαθέσιμο χρόνο, με αποτέλεσμα να ολοκληρώνονται έγκαιρα οι περισσότερες από αυτές.

Και στις πέντε κατηγορίες, οι επιδόσεις του αλγορίθμου Round Robin είναι καλύτερες από τις αντίστοιχες της πολιτικής Lifetime. Αυτό συμβαίνει διότι στην περίπτωση του Round Robin η αποστολή των αιτήσεων στους εξυπηρετητές γίνεται με τυχαίο τρόπο ακολουθώντας μια ομοιόμορφη κατανομή. Αυτό έχει σαν αποτέλεσμα να εξισορροπείται ο φόρτος στους εξυπηρετητές. Από την άλλη πλευρά, στην περίπτωση της πολιτικής Lifetime, η αποστολή των αιτήσεων γίνεται λαμβάνοντας υπόψη μόνο τη διάρκεια ζωής των εξυπηρετητών και καθόλου τον φόρτο των εξυπηρετητών. Αυτό έχει ως αποτέλεσμα, να απορρίπτονται εξυπηρετητές που έχουν μεγαλύτερη διάρκεια ζωής από την διάρκεια ζωής του πελάτη που αποστέλλει την αίτηση. Στην περίπτωση που ο φόρτος αυτών των εξυπηρετητών είναι μικρός, θα μπορούσαν να εξυπηρετήσουν έγκαιρα μια αίτηση παρόλο που η διάρκεια ζωής τους είναι μεγαλύτερη από του πελάτη.

Το πείραμα αυτό κάνει εμφανές το γεγονός ότι είναι σημαντικό να συμπεριληφθεί, στις πολιτικές μας, τόσο το κόστος όσο και η διάρκεια ζωής των κινητών οντοτήτων. Η παρατήρηση αυτή οδηγεί στην ουσία, στην πρόταση των πολιτικών που υλοποιούνται σε αυτήν την εργασία.

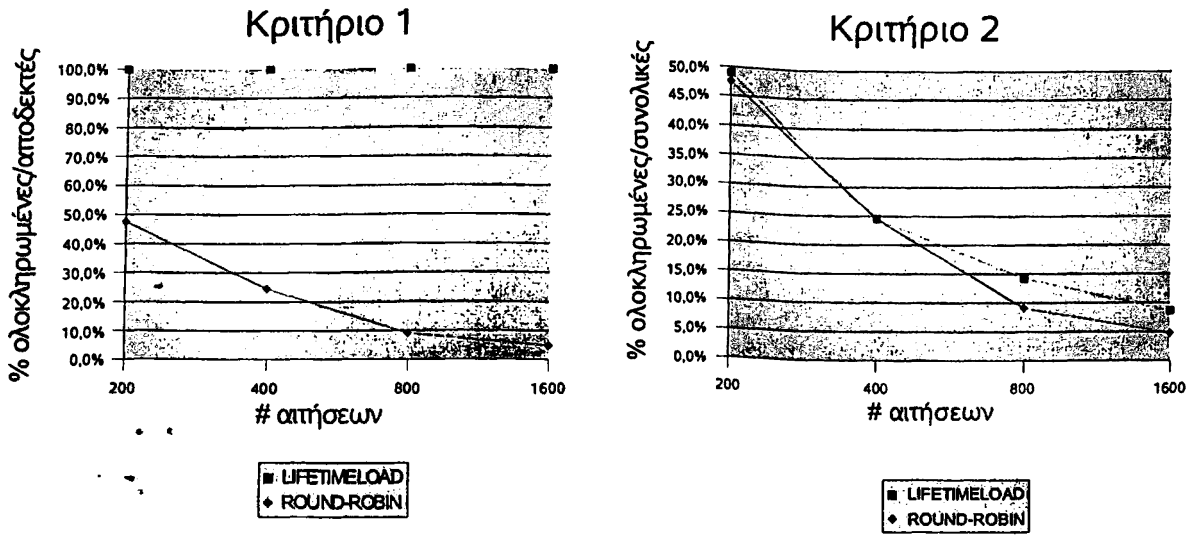
5.2. Σύγκριση της Πολιτικής LifetimeLoad με τον Αλγόριθμο RR

Σε αυτήν την ενότητα γίνεται σύγκριση της πολιτικής LifetimeLoad με τον αλγόριθμο Round Robin. Ουσιαστικά κάνουμε αποτίμηση της επίδοσης της πολιτικής LifetimeLoad, με και χωρίς τις επιπρόσθετες συνθήκες που έχουμε προτείνει.

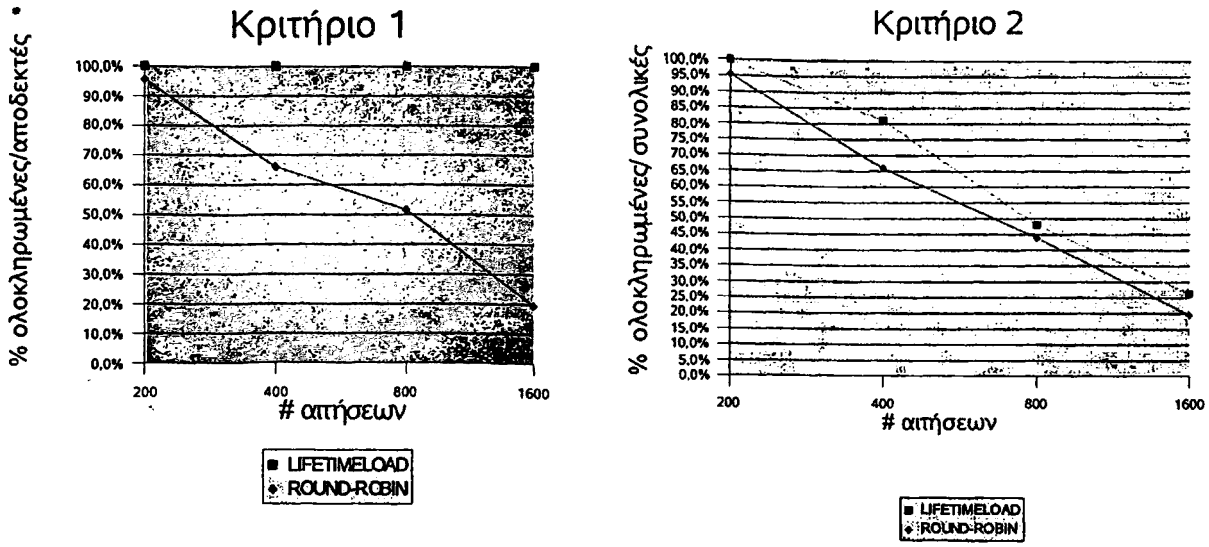
Όπως αναφέρθηκε, οι παράμετροι των κινητών οντοτήτων παράγονταν με τυχαίο τρόπο ακολουθώντας μια ομοιόμορφη κατανομή. Πιο συγκεκριμένα, ο χρόνος εκτέλεσης της προσομοίωσης, *simRuntime*, ήταν 15000 μονάδες. Το κβάντο των χρονοπρογραμματιστών ήταν 1 χρονική μονάδα. Οι διάρκειες ζωής του εξυπηρετητή και του πελάτη έπαιρναν τιμές στο διάστημα (*simRuntime/10*, *simRuntime*) ακολουθώντας ομοιόμορφη κατανομή. Οι μετρήσεις πραγματοποιήθηκαν χρησιμοποιώντας τέσσερα διαφορετικά σύνολα: 200, 400, 800 και 1600 αιτήσεων. Για τις πρώτες τέσσερις από τις πέντε κατηγορίες μετρήσεων, τα διαστήματα από τα οποία έπαιρνε τιμές το C κάθε υπηρεσίας ήταν:

- (*1, clientLifetime / 40*) για την πρώτη κατηγορία.
- (*1, clientLifetime / 160*) για την δεύτερη.
- (*1, clientLifetime / 320*) για την τρίτη, και
- (*1, clientLifetime / 640*) για την τέταρτη.

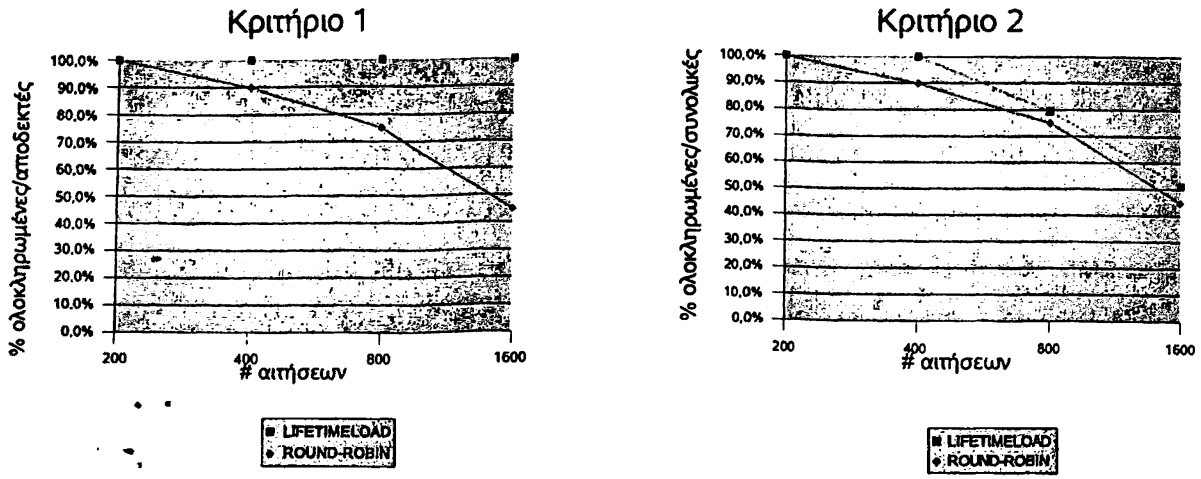
Στα παρακάτω σχήματα γίνεται σύγκριση των δύο πολιτικών βάσει των δύο κριτηρίων που αναφέρθηκαν στην αρχή του κεφαλαίου.



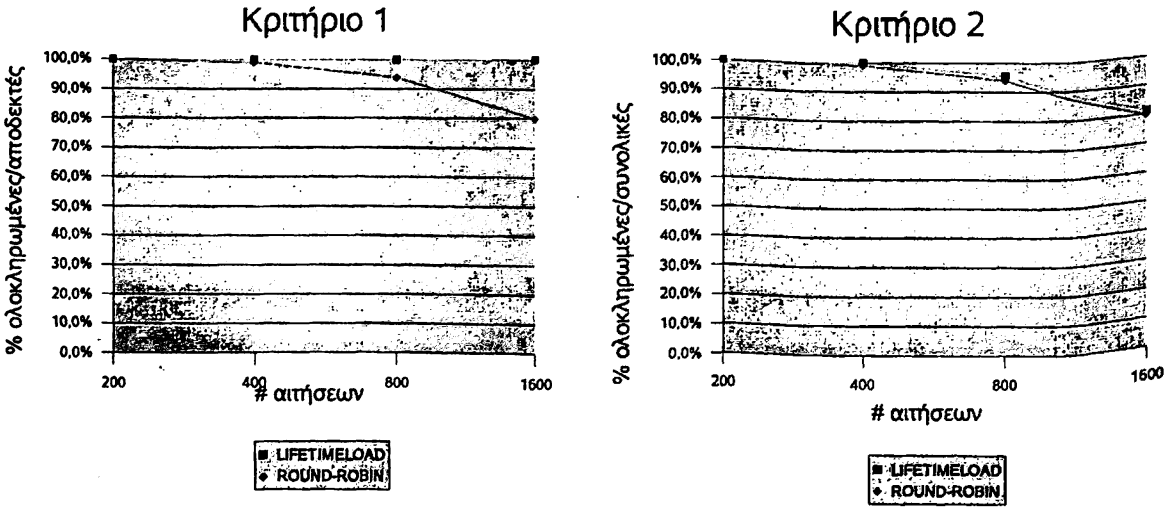
Σχήμα 5.6 Πρώτη Κατηγορία Μετρήσεων. $C \in (1, \frac{clientLifetime}{40})$



Σχήμα 5.7 Δεύτερη Κατηγορία Μετρήσεων. $C \in (1, \frac{clientLifetime}{160})$

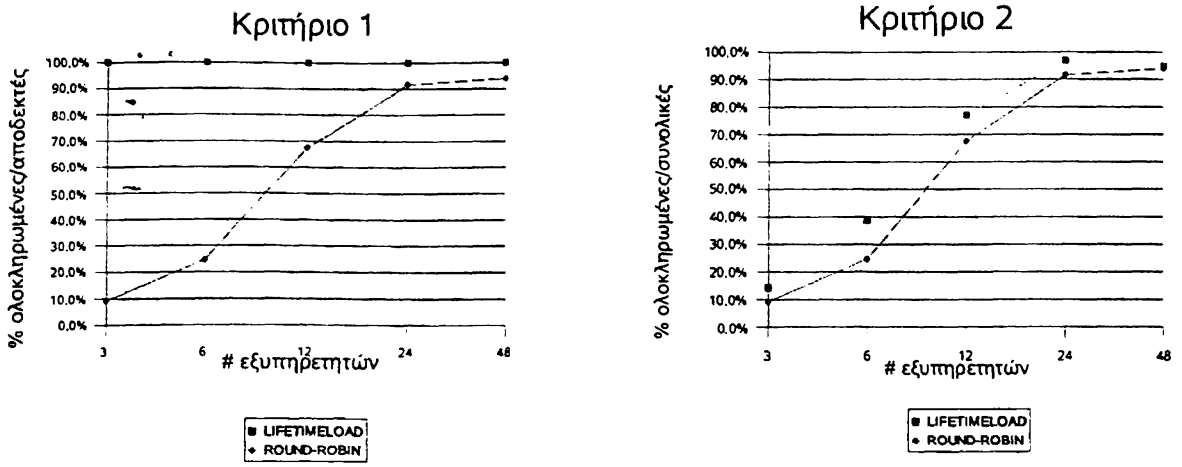


Σχήμα 5.8 Τρίτη Κατηγορία Μετρήσεων. $C \in (1, \frac{clientLifetime}{320})$



Σχήμα 5.9 Τέταρτη Κατηγορία Μετρήσεων. $C \in (1, \frac{clientLifetime}{640})$

Όπως παρατηρούμε στα παραπάνω σχήματα, το ποσοστό των αιτήσεων που ολοκληρώνονται έγκαιρα στην περίπτωση του RoundRobin, μειώνεται σχεδόν γραμμικά καθώς αυξάνεται ο αριθμός των αιτήσεων. Από την άλλη πλευρά, η πολιτική LifetimeLoad εγγυάται ότι όλες οι αιτήσεις που έγιναν αποδεκτές εκτελούνται έγκαιρα. Επίσης παρατηρούμε ότι το ποσοστό του δεύτερου κριτηρίου, δηλ. το κλάσμα των αιτήσεων που εξυπηρετήθηκαν με επιτυχία προς τις συνολικές αιτήσεις που παρήγαγαν οι πελάτες, μειώνεται γραμμικά και στις δύο περι-



Σχήμα 5.10 Σύγκριση με Αυξανόμενο Αριθμό Εξυπηρετητών.

πτώσεις .Όσο μειώνεται ο χρόνος εκτέλεσης C, αυξάνεται το ποσοστό επίδοσης των δύο κριτηρίων. Αυτό προκύπτει διότι καθώς μειώνονται οι χρόνοι εκτέλεσης των υπηρεσιών, μειώνεται και ο φόρτος του κάθε εξυπηρετητή με αποτέλεσμα να μπορεί ο κάθε ένας να εξυπηρετεί επιτυχημένα περισσότερες αιτήσεις μέσα στο ίδιο χρονικό διάστημα.

Στο Σχήμα 5.10 φαίνεται η πέμπτη κατηγορία μετρήσεων. Σε αυτήν την περίπτωση ο αριθμός των εξυπηρετητών αυξάνεται: αντίστοιχα, έχουμε 3, 6, 12, 24 και 48 εξυπηρετητές στο σύστημα. Ο αριθμός των αιτήσεων που πραγματοποιούνται είναι 800 και το διάστημα από το οποίο παίρνει τιμές το C κάθε υπηρεσίας είναι (*1, clientLifetime / 40*).

Όπως παρατηρούμε, η απόδοση του πρώτου κριτηρίου στην περίπτωση της πολιτικής LifetimeLoad παραμένει σταθερή στο 100%, καθώς αυξάνεται ο αριθμός των εξυπηρετητών. Αντίθετα, στην περίπτωση του RoundRobin το ποσοστό του πρώτου κριτηρίου αυξάνεται σχεδόν γραμμικά καθώς αυξάνεται ο αριθμός των εξυπηρετητών στο σύστημα. Όσον αφορά το δεύτερο κριτήριο, οι δύο πολιτικές συμπεριφέρονται όμοια καθώς η αύξηση είναι σχεδόν γραμμική και στις δύο περιπτώσεις. Αυτό προκύπτει διότι με την αύξηση των εξυπηρετητών μοιράζεται ο φόρτος του συστήματος, που είναι σταθερός, σε περισσότερους εξυπηρετητές.

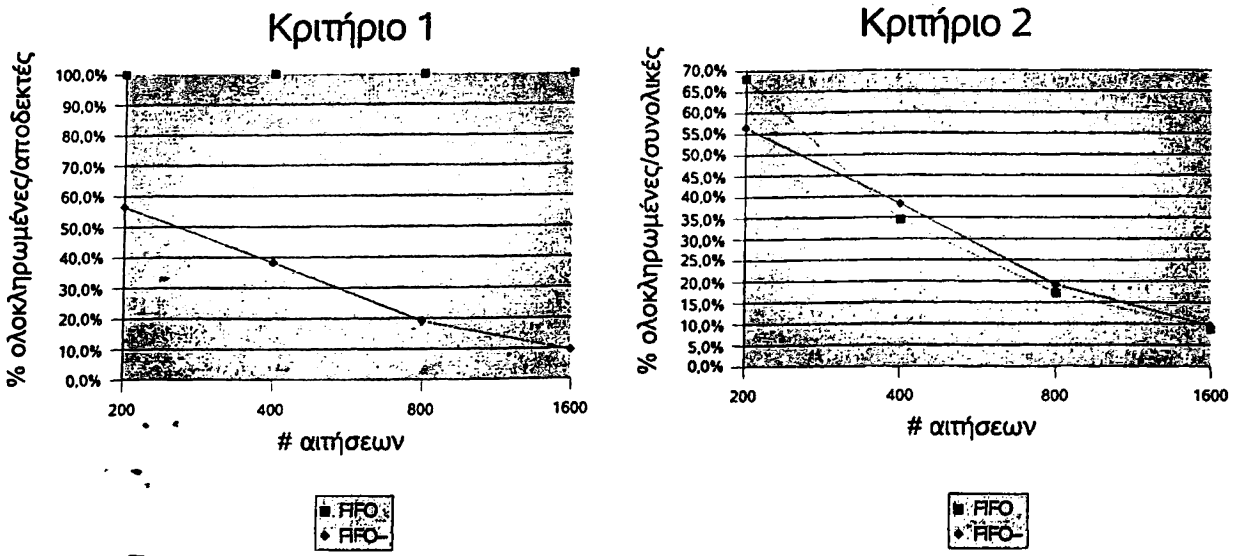
Παρατηρούμε λοιπόν, ότι και στις πέντε κατηγορίες των πειραματικών μετρήσεων, όσες αιτήσεις γίνονται αποδεκτές από την πολιτική LifetimeLoad, ολοκληρώνονται έγκαιρα. Όσον

αφορά το δεύτερο κριτήριο, οι επιδόσεις της είναι ελαφρώς καλύτερες από τον απλό Round Robin. Κάτι τέτοιο προκύπτει από την φύση του αλγορίθμου Round Robin, σύμφωνα με τον οποίο κάθε υπηρεσία εκτελείται για ένα κβάντο του χρονοπρογραμματιστή δίνοντας έτσι την ευκαιρία σε κάθε υπηρεσία που βρίσκεται στην ουρά να εκτελεστεί για ένα μικρό χρονικό διάστημα χωρίς να χρειαστεί να αναμένει να ολοκληρώσουν την εκτέλεση τους όλες οι υπηρεσίες που προηγούνται στην ουρά. Το γεγονός αυτό, σε συνδυασμό με το ότι στην πολιτική LifetimeLoad λαμβάνεται υπόψη και ο φόρτος του εξυπηρετητή και ότι ο πελάτης μπορεί να επιλέξει κάποιον άλλον εξυπηρετητή, που προσφέρει ισοδύναμες υπηρεσίες, για την εξυπηρέτηση της αίτησης, έχει σαν συνέπεια την καλύτερη διαχείριση του χρόνου των εξυπηρετητών.

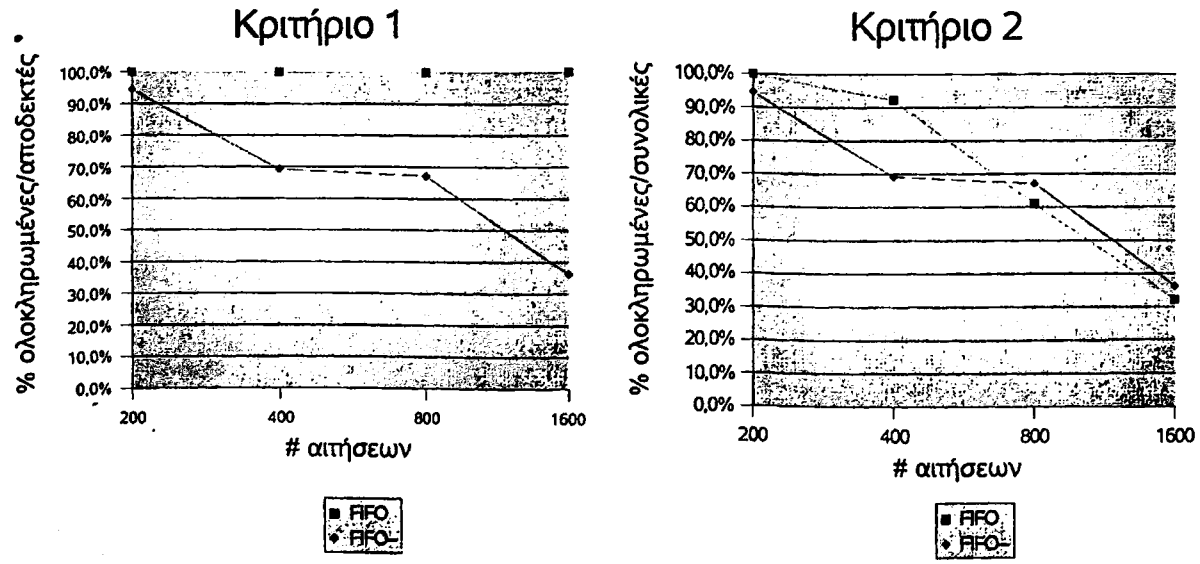
5.3. Σύγκριση της Πολιτικής FIFO με την FIFO Χωρίς τις Συνθήκες

Σε αυτήν την ενότητα γίνεται σύγκριση της πολιτικής FIFO, με και χωρίς τις συνθήκες που έχουν προταθεί στο Κεφάλαιο 4. Ονομάζουμε FIFO--, την πολιτική FIFO χωρίς τις συνθήκες.

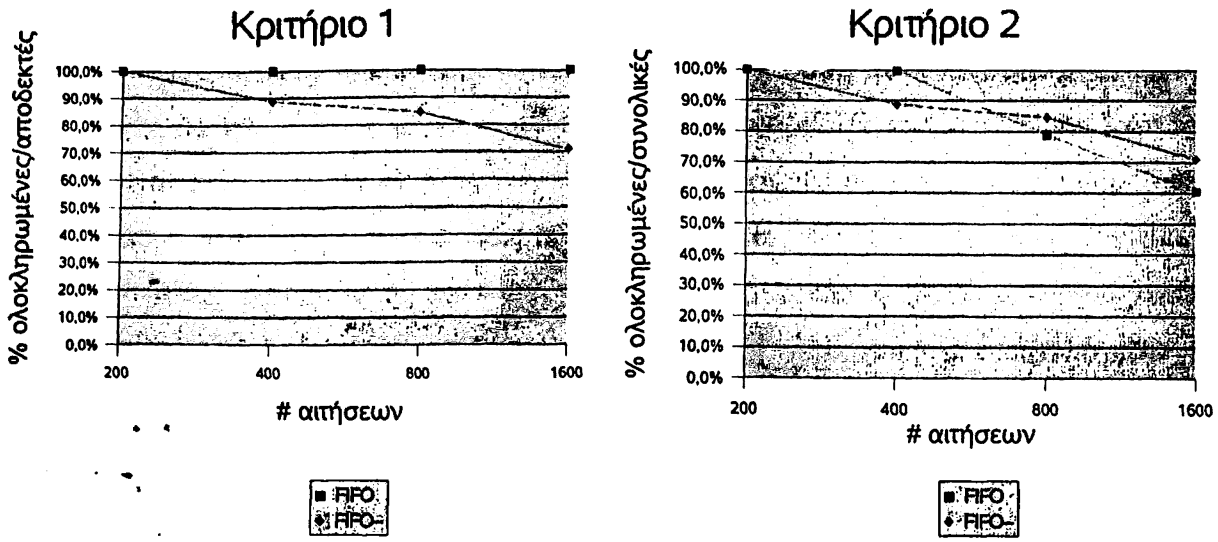
Όπως και στην προηγούμενη ενότητα, οι παράμετροι των κινητών οντοτήτων παράγονται με τυχαίο τρόπο ακολουθώντας μια ομοιόμορφη κατανομή. Συγκεκριμένα, ο χρόνος εκτέλεσης της προσομοίωσης, *simRuntime*, είναι 15000 μονάδες και οι διάρκειες ζωής του εξυπηρετητή και του πελάτη παίρνουν τιμές στο διάστημα (*simRuntime/10*, *simRuntime*) ακολουθώντας ομοιόμορφη κατανομή. Οι μετρήσεις πραγματοποιήθηκαν χρησιμοποιώντας τέσσερα διαφορετικά σύνολα: 200, 400, 800 και 1600 αιτήσεων. Για τις πρώτες τέσσερις από τις πέντε κατηγορίες μετρήσεων, τα διαστήματα από τα οποία έπαιρνε τιμές το C κάθε υπηρεσίας ήταν (*1, clientLifetime / 40*) για την πρώτη κατηγορία, (*1, clientLifetime / 160*) για την δεύτερη, (*1, clientLifetime / 320*) για την τρίτη και (*1, clientLifetime / 640*) για την τέταρτη. Στα παρακάτω σχήματα γίνεται σύγκριση των δύο πολιτικών βάσει των δύο κριτηρίων που αναφέρθηκαν στην αρχή του κεφαλαίου.



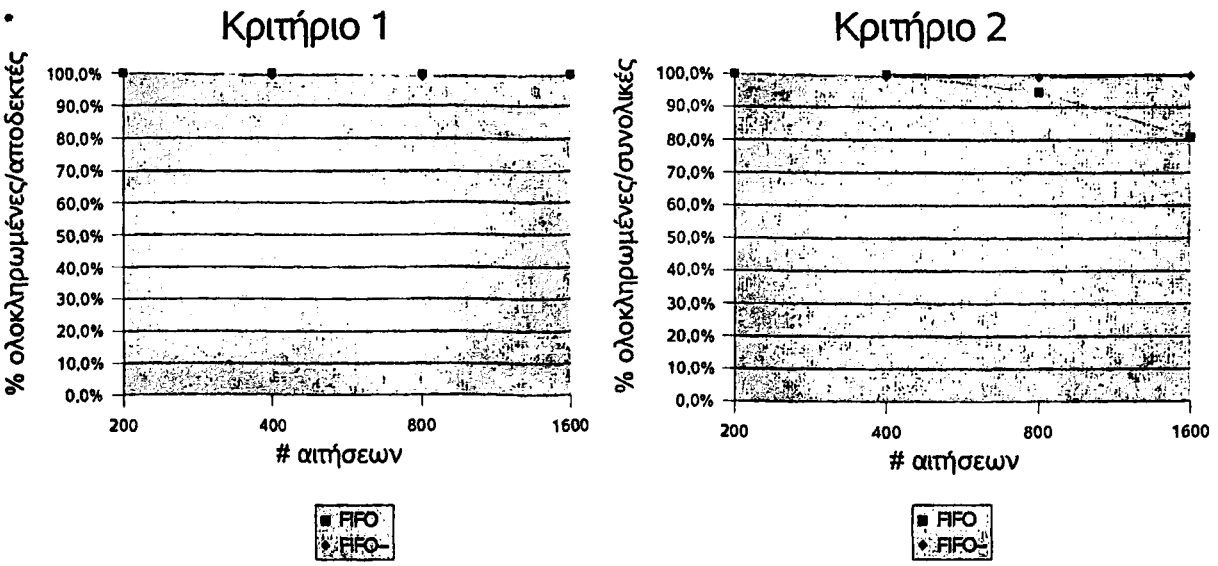
Σχήμα 5.11 Πρώτη Κατηγορία Μετρήσεων. $C \in (1, \frac{clientLifetime}{40})$



Σχήμα 5.12 Δεύτερη Κατηγορία Μετρήσεων. $C \in (1, \frac{clientLifetime}{160})$



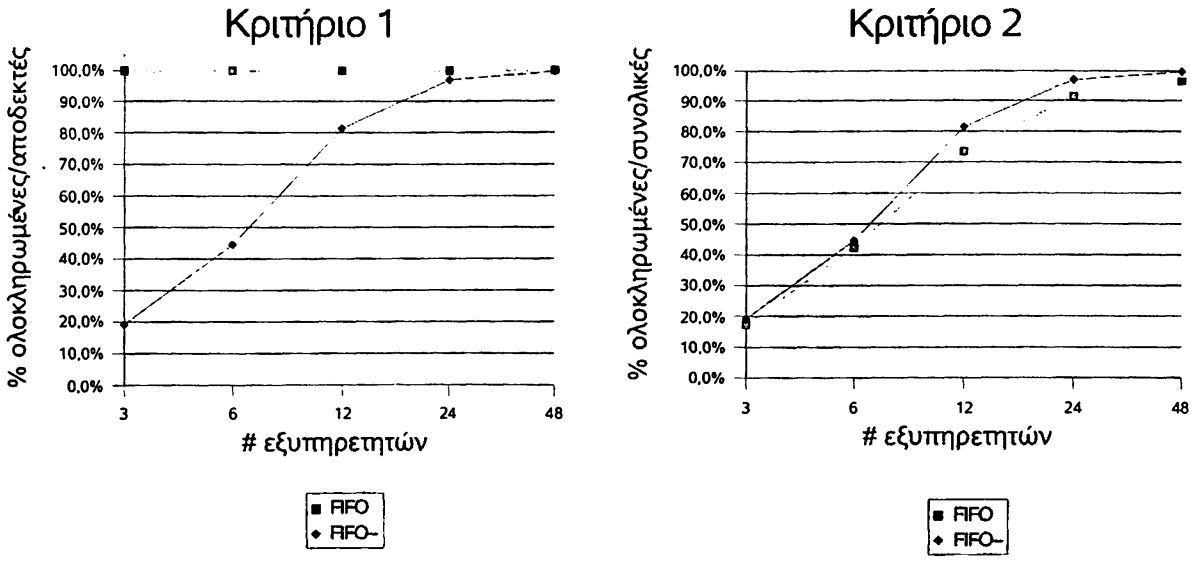
Σχήμα 5.13 Τρίτη Κατηγορία Μετρήσεων. $C \in (1, \frac{clientLifetime}{320})$



Σχήμα 5.14 Τέταρτη Κατηγορία Μετρήσεων. $C \in (1, \frac{clientLifetime}{640})$

Όπως παρατηρούμε στα παραπάνω σχήματα, η επίδοση του πρώτου κριτηρίου παραμένει σταθερή στο 100%, όσον αφορά την πολιτική FIFO. Η επίδοση αυτή παραμένει σταθερή και στις τέσσερις κατηγορίες, ακόμη και καθώς αυξάνεται ο αριθμός των αιτήσεων. Κάτι τέτοιο δικαιολογείται διότι η πολιτική FIFO λαμβάνει υπόψη της το φόρτο του κάθε εξυπηρετητή. Έτσι η πολιτική FIFO εγγυάται ότι όλες οι αιτήσεις που έγιναν αποδεκτές εκτελούνται έγκαιρα. Από την άλλη πλευρά, στην πολιτική FIFO-- η επίδοση του πρώτου κριτηρίου μειώνεται σχεδόν γραμμικά καθώς αυξάνονται οι αιτήσεις. Αυτή η συμπεριφορά δικαιολογείται λόγω αύξησης του φόρτου. Όσον αφορά το ποσοστό του δεύτερου κριτηρίου, δηλ. το κλάσμα των αιτήσεων που εξυπηρετήθηκαν με επιτυχία προς τις συνολικές αιτήσεις που παρήγαγαν οι πελάτες, αυτό μειώνεται γραμμικά και στις δύο περιπτώσεις.

Όσο μειώνεται ο χρόνος εκτέλεσης C, παρατηρούμε ότι αυξάνεται το ποσοστό επίδοσης του δεύτερου κριτηρίου της πολιτικής FIFO και των δύο κριτηρίων της πολιτικής FIFO--. Αυτό προκύπτει διότι καθώς μειώνονται οι χρόνοι εκτέλεσης των υπηρεσιών, μειώνεται και ο φόρτος του κάθε εξυπηρετητή με αποτέλεσμα να μπορεί ο κάθε ένας να εξυπηρετεί επιτυχημένα περισσότερες αιτήσεις μέσα στο ίδιο χρονικό διάστημα.



Σχήμα 5.15 Σύγκριση με Αυξανόμενο Αριθμό Εξυπηρετητών.

Στο Σχήμα 5.15 φαίνεται η πέμπτη κατηγορία μετρήσεων. Σε αυτήν την περίπτωση ο αριθμός των εξυπηρετητών αυξάνεται: αντίστοιχα, έχουμε 3, 6, 12, 24 και 48 εξυπηρετητές στο σύστημα. Ο αριθμός των αιτήσεων που πραγματοποιούνται είναι 800 και το διάστημα από το οποίο παίρνει τιμές το C κάθε υπηρεσίας είναι (*1, clientLifetime / 40*). Όπως παρατηρούμε, η απόδοση του πρώτου κριτηρίου στην περίπτωση της πολιτικής FIFO παραμένει στα-

θερή στο 100%, καθώς αυξάνεται ο αριθμός των εξυπηρετητών. Αντίθετα, στην περίπτωση της FIFO-- το ποσοστό του πρώτου κριτηρίου αυξάνεται σχεδόν γραμμικά καθώς αυξάνεται ο αριθμός των εξυπηρετητών στο σύστημα, συγκλίνοντας στο 100%. Όσον αφορά το δεύτερο κριτήριο, οι δύο πολιτικές συμπεριφέρονται όμοια καθώς η αύξηση είναι σχεδόν γραμμική και στις δύο περιπτώσεις. Αυτό προκύπτει διότι με την αύξηση των εξυπηρετητών μοιράζεται ο φόρτος του συστήματος σε περισσότερους εξυπηρετητές.

Παρατηρούμε λοιπόν, ότι και στις πέντε κατηγορίες των πειραματικών μετρήσεων, όσες αιτήσεις γίνονται αποδεκτές από την πολιτική FIFO, ολοκληρώνονται έγκαιρα. Όσον αφορά το δεύτερο κριτήριο, οι επιδόσεις της είναι ελαφρώς χειρότερες σε σχέση με την FIFO--. Αντίθετα, κάτι τέτοιο δεν παρατηρήθηκε στην σύγκριση των LifetimeLoad και RoundRobin, που αφορούσε στο δεύτερο κριτήριο. Η διαφορά έγκειται στον τρόπο εξυπηρέτησης των εργασιών. Στην περίπτωση της πολιτικής LifetimeLoad χρησιμοποιείται ο αλγόριθμος Round Robin, ενώ στην πολιτική FIFO χρησιμοποιείται ο αλγόριθμος FIFO. Σύμφωνα με τον αλγόριθμο FIFO, κάθε νέα εργασία που προστίθεται στην ουρά, χρειάζεται να αναμένει να ολοκληρώσουν την εκτέλεση τους όλες οι εργασίες που προηγούνται στην ουρά. Συνεπώς, μια εργασία με μικρό χρόνο εκτέλεσης μπορεί να απορριφθεί επειδή θα χρειαστεί να περιμένει όλες τις άλλες εργασίες που προηγούνται στην ουρά, οι οποίες πιθανώς να απαιτούν συνολικά αρκετό χρόνο για να ολοκληρώσουν την εκτέλεση τους. Η εργασία αυτή θα μπορούσε πιθανώς να εξυπηρετηθεί από την πολιτική LifetimeLoad διότι λόγω του μικρού χρόνου εκτέλεσης που απαιτεί, δεν θα προκαλούσε μεγάλη επιβάρυνση σε όλες τις υπόλοιπες μιας και μετά από κάποια κβάντα χρόνου θα ολοκλήρωνε την εκτέλεση της και θα αφαιρούνταν από την ουρά. Άρα, στην πολιτική FIFO γενικά απορρίπτεται μεγαλύτερο ποσοστό αιτήσεων στην περίπτωση μικρών C, σε σχέση με την LifetimeLoad. Αυτό έχει ως συνέπεια να είναι χειρότερη από την FIFO--, όσον αφορά στο δεύτερο κριτήριο.

Μια δεύτερη παράμετρος είναι η "ζημιά" που μπορεί να προκαλέσει μια προσθήκη μια νέας εργασίας. Πιο συγκεκριμένα, στην περίπτωση της FIFO--, η νέα εργασία που προστίθεται στην ουρά δεν προκαλεί καμία επιβάρυνση σε όσες προηγούνται. Αντίθετα, στον αλγόριθμο RoundRobin η προσθήκη μιας νέας αίτησης για την εκτέλεση μιας εργασίας προκαλεί επιβάρυνση σε όλες τις εργασίες της ουράς. Έτσι, η προσθήκη μίας νέας εργασίας στην περίπτωση του RoundRobin, μπορεί να προκαλέσει τη εκπρόθεσμη εκτέλεση και της νέας εργασίας αλλά και πολλών άλλων. Συνεπώς, η απόρριψη της νέας αυτής εργασίας εξασφαλίσει την έγκαιρη εκτέλεση όλων των άλλων εργασιών στην ουρά και το κέρδος είναι μεγάλο. Κάτι τέτοιο όμως, δεν ισχύει στην περίπτωση της πολιτικής FIFO--, μιας και οι υπόλοιπες εργασίες στην ουρά δεν επηρεάζονται από αυτήν. Συνεπώς, σε αυτήν την περίπτωση η απόρριψη της

νέας εργασίας δεν αποφέρει μεγάλο κέρδος.

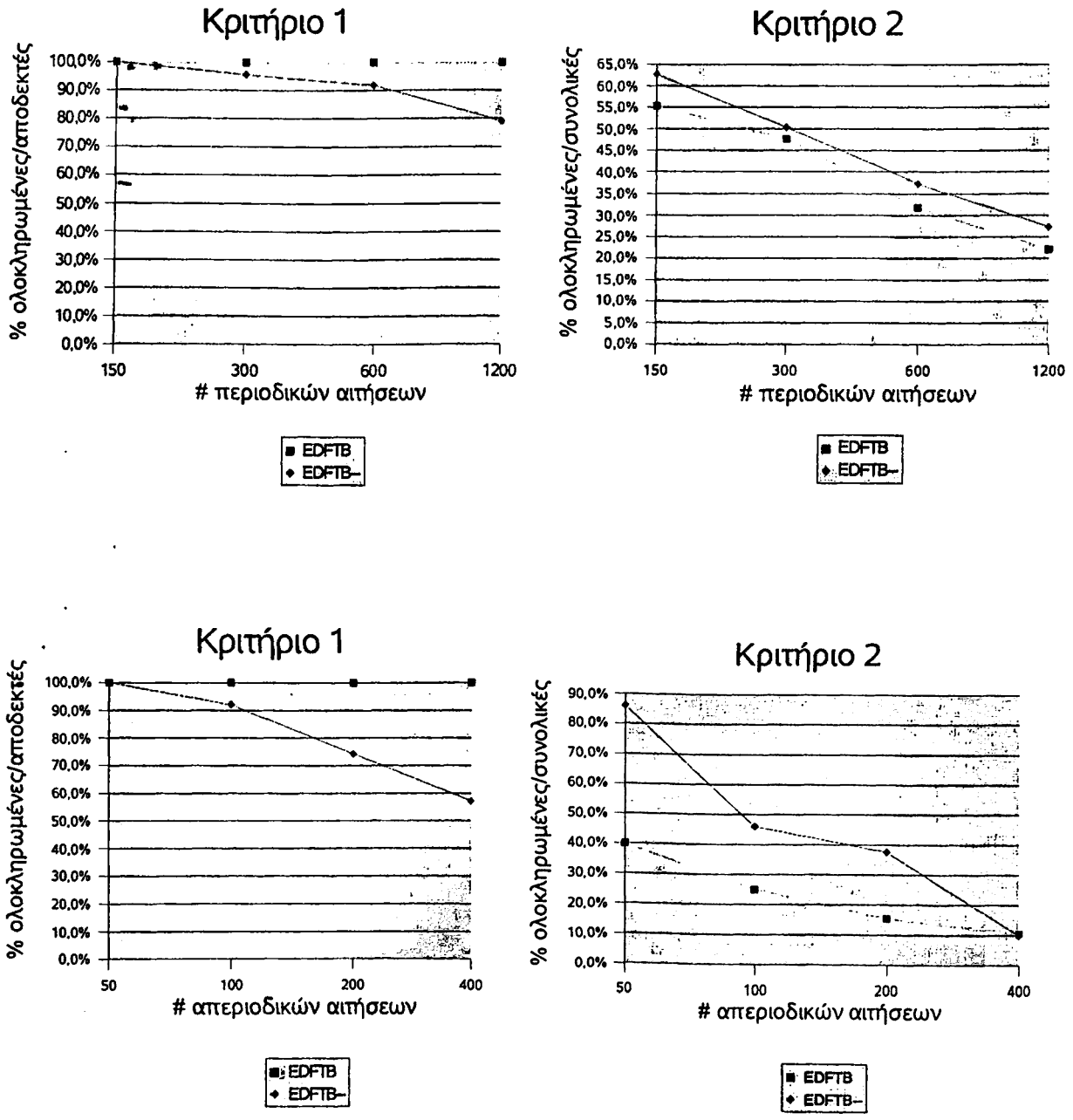
Στις ενότητες 5.1, 5.2 και 5.3 παρουσιάστηκαν οι πειραματικές μετρήσεις των πολιτικών Lifetime, LifetimeLoad και FIFO. Σε αυτές τις τρεις πολιτικές, όλες οι αιτήσεις σχετίζονταν με την εκτέλεση απεριοδικών εργασιών. Όπως παρατηρήσαμε από τις πειραματικές μετρήσεις των ανωτέρω πολιτικών, η πολιτική με τις καλύτερες επιδόσεις και στα δύο κριτήρια, είναι η LifetimeLoad. Στην επόμενη ενότητα θα μελετήσουμε τις πειραματικές μετρήσεις της πολιτικής EDFTB, η οποία υποστηρίζει τον από κοινού χρονοπρογραμματισμό περιοδικών και απεριοδικών εργασιών.

5.4. Σύγκριση της Πολιτικής EDFTB με την EDFTB Χωρίς τις Συνθήκες

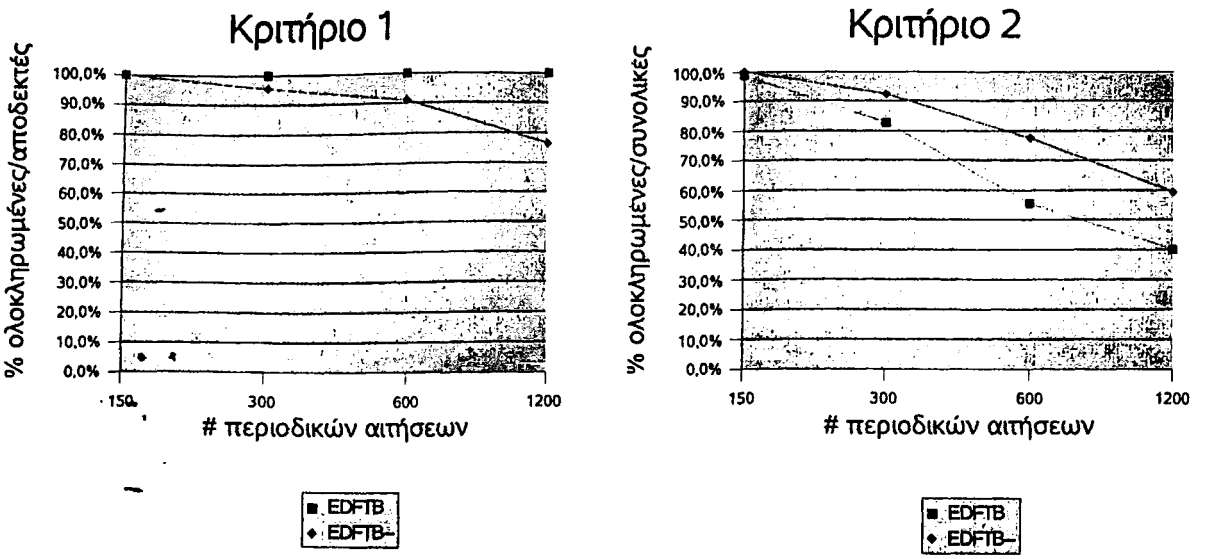
Σε αυτήν την ενότητα γίνεται σύγκριση της πολιτικής EDFTB, με και χωρίς τις συνθήκες που έχουν προταθεί στο Κεφάλαιο 4. Όπως και πριν, ονομάζουμε EDFTB--, την πολιτική EDFTB χωρίς τις επιπρόσθετες συνθήκες. Σύμφωνα με την πολιτική EDFTB, ο χρονοπρογραμματιστής εξυπηρετεί περιοδικές και απεριοδικές εργασίες στο ίδιο χρονοπρόγραμμα. Συνεπώς, οι μετρήσεις πραγματοποιήθηκαν με τον από κοινού χρονοπρογραμματισμό περιοδικών και απεριοδικών εργασιών.

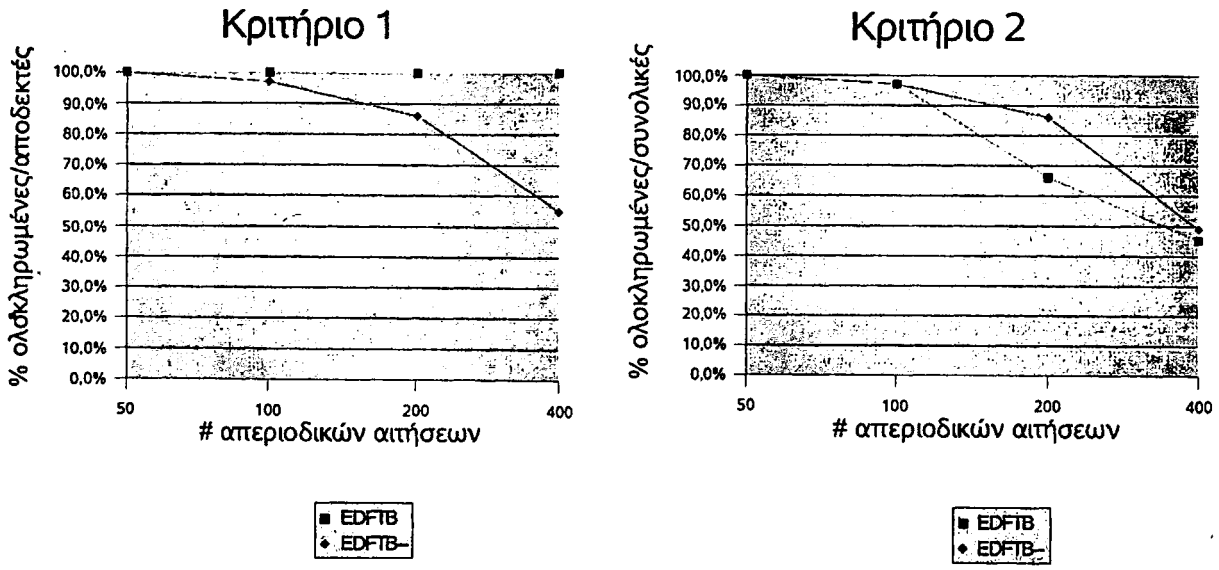
Όπως και προηγουμένως, οι τιμές των παραμέτρων των κινητών οντοτήτων ακολουθούν μια ομοιόμορφη κατανομή. Συγκεκριμένα, ο χρόνος εκτέλεσης της προσομοίωσης, *simRuntime*, είναι 15000 μονάδες και οι διάρκειες ζωής του εξυπηρετητή και του πελάτη παίρνουν τιμές στο διάστημα (*simRuntime/10*, *simRuntime*) ακολουθώντας ομοιόμορφη κατανομή. Η τιμή του παράγοντα αξιοποίησης *Us* ορίστηκε στην τιμή 0,25. Η περίοδος των περιοδικών εργασιών παίρνει τιμές στο διάστημα (*clientLifetime / 500, clientLifetime / 50*), η προθεσμία της κάθε περιοδικής εργασίας είναι ίση με την περίοδό της και ο αριθμός επαναλήψεων, *n*, παίρνει τιμές στο διάστημα [1, 10]. Οι μετρήσεις πραγματοποιήθηκαν χρησιμοποιώντας τέσσερα διαφορετικά σύνολα: 200, 400, 800 και 1600 αιτήσεων. Για κάθε ένα από τα προηγούμενα σύνολα χρησιμοποιήθηκε το 25% του συνόλου για την δημιουργία απεριοδικών αιτήσεων και το υπόλοιπο 75% για την δημιουργία περιοδικών. Το 25% αντιστοιχεί στην τιμή του παράγοντα *Us*. Για παράδειγμα, στην περίπτωση του τέταρτου συνόλου δημιουργήθηκαν 400 απεριοδικές και 1200 περιοδικές αιτήσεις. Για τις πρώτες τέσσερις από τις πέντε κατηγορίες μετρήσεων, τα διαστήματα από τα οποία έπαιρνε τιμές το C κάθε υπηρεσίας ήταν τα εξής: (*1, clientLifetime / 40*), (*1, clientLifetime / 160*), (*1, clientLifetime / 320*) και (*1, clientLifetime / 640*) αντίστοιχα. Στην πέμπτη κατηγορία μετρήσεων ο αριθμός των εξυπηρετητών αυξάνεται: αντίστοιχα, έχουμε 3, 6, 12, 24 και 48 εξυπηρετητές στο σύστημα. Ο αριθ-

μός των συνολικών αιτήσεων που πραγματοποιούνται είναι 800, δηλ. 600 περιοδικές και 200 απεριοδικές, και το διάστημα από το οποίο παίρνει τιμές το C κάθε υπηρεσίας είναι (*1, clientLifetime / 40*). Στα παρακάτω σχήματα γίνεται σύγκριση των δύο πολιτικών βάσει των δύο κριτηρίων που αναφέρθηκαν στην αρχή του κεφαλαίου.

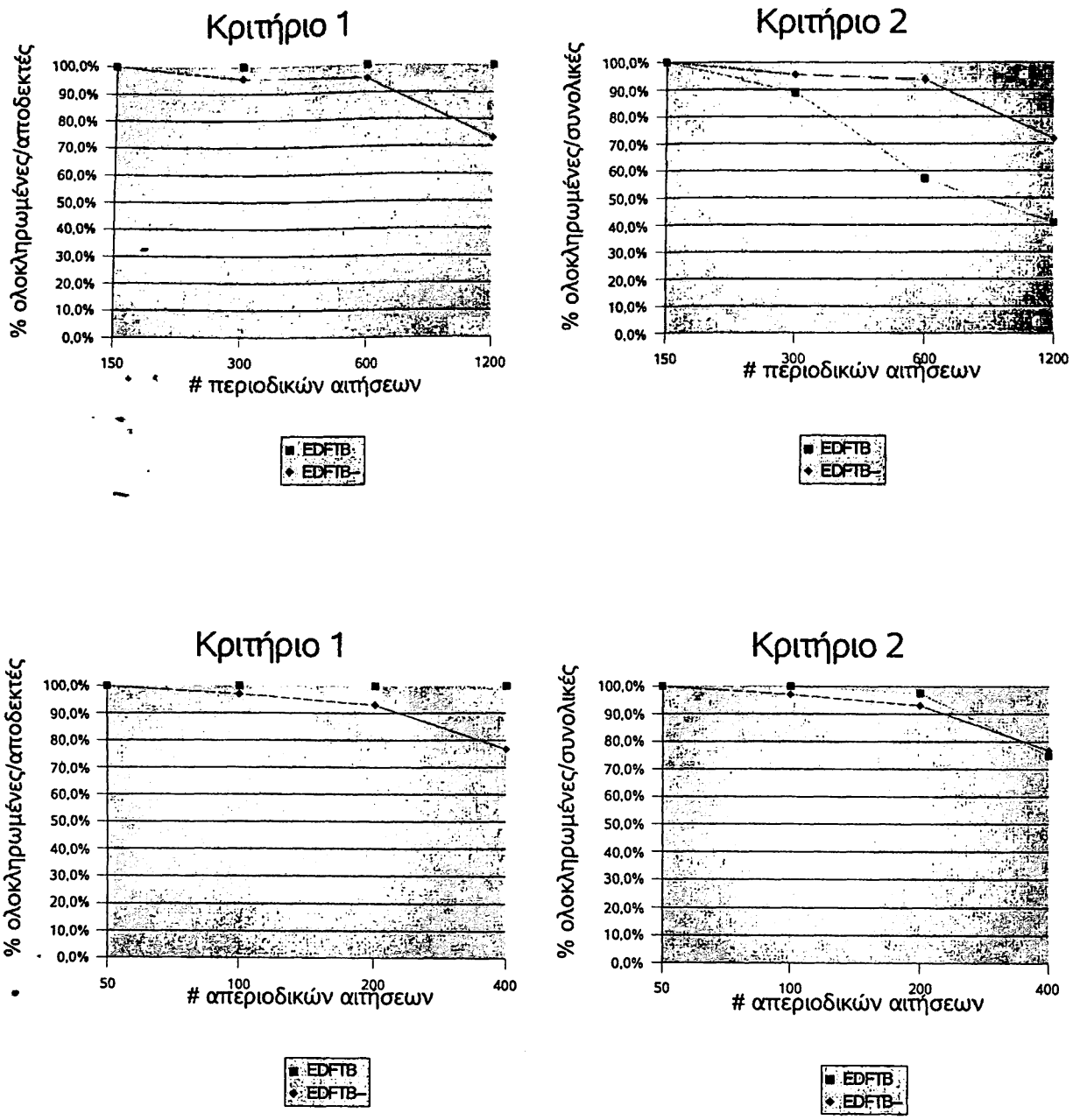


Σχήμα 5.16 Πρώτη Κατηγορία Μετρήσεων. $C \in (1, \frac{clientLifetime}{40})$

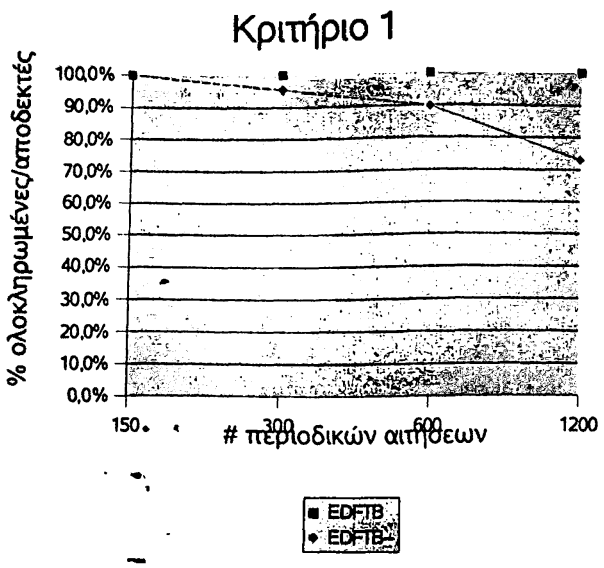


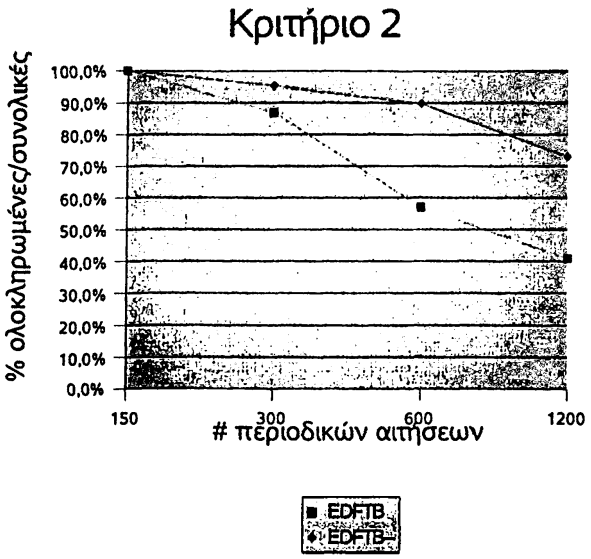


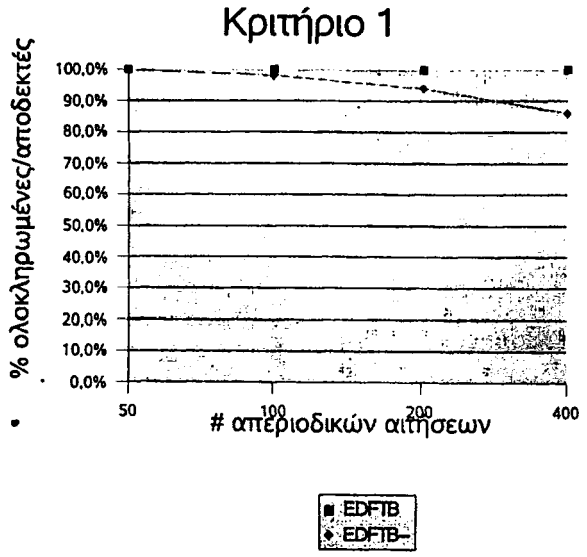
Σχήμα 5.17 Δεύτερη Κατηγορία Μετρήσεων. $C \in (1, \frac{clientLifetime}{160})$

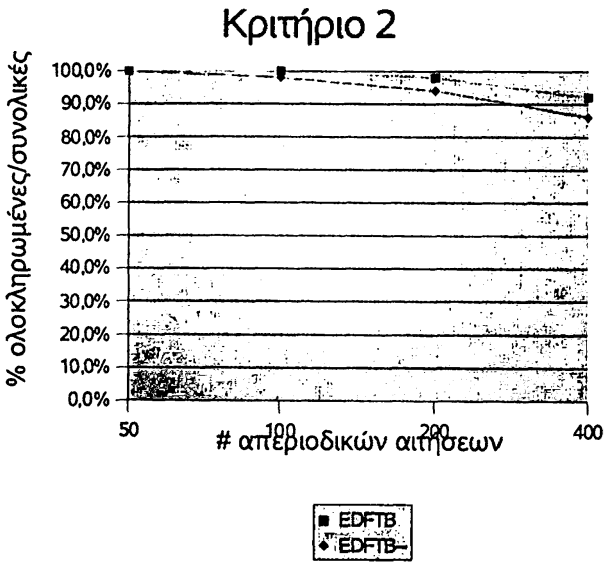


Σχήμα 5.18 Τρίτη Κατηγορία Μετρήσεων. $C \in (1, \frac{clientLifetime}{320})$

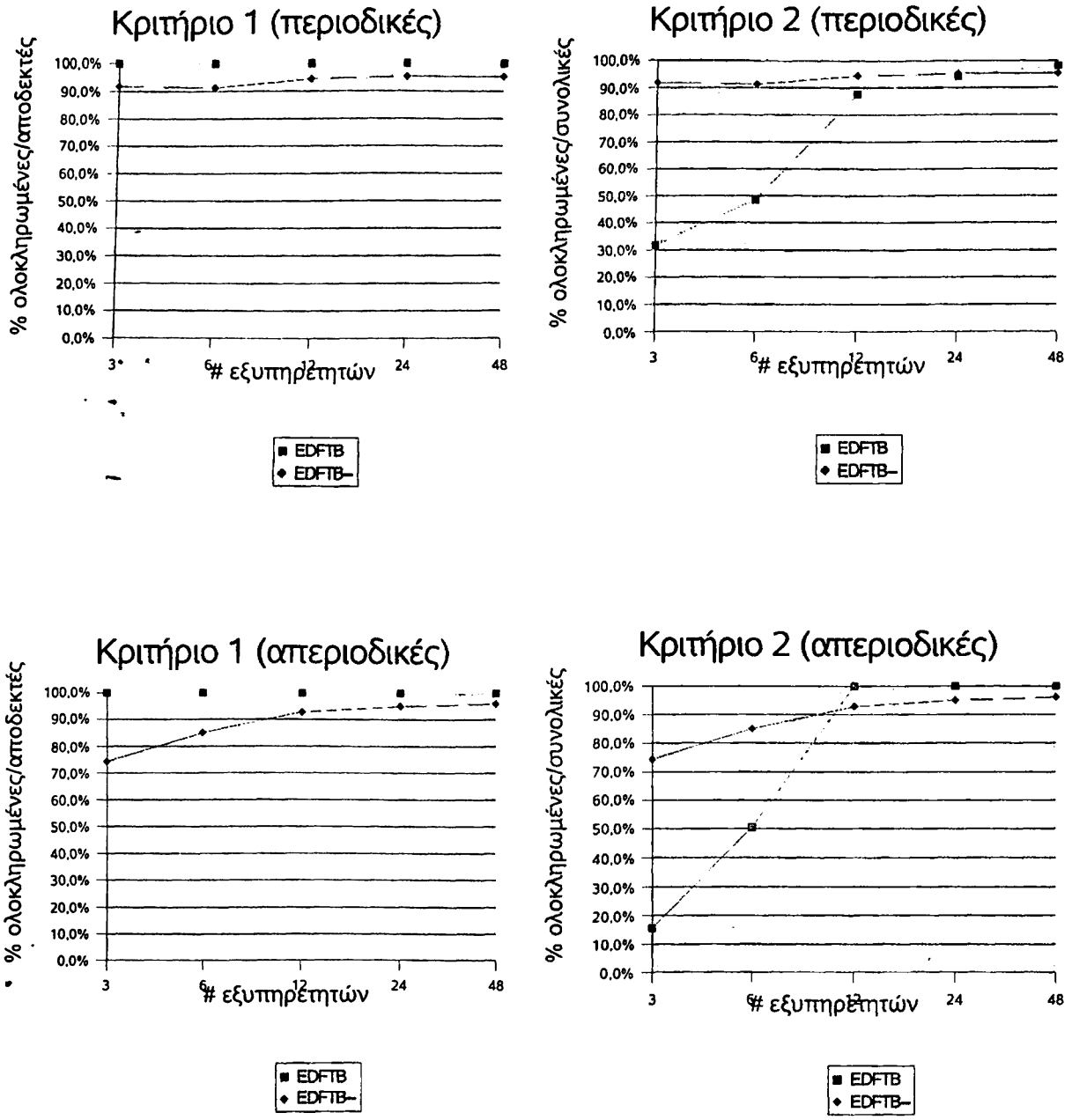








Σχήμα 5.19 Τέταρτη Κατηγορία Μετρήσεων. $C \in (1, \frac{clientLifetime}{640})$



Σχήμα 5.20 Σύγκριση με Αυξανόμενο Αριθμό Εξυπηρετητών.

Όπως παρατηρούμε στα παραπάνω σχήματα, το ποσοστό της επίδοσης του πρώτου κριτηρίου στην περίπτωση της πολιτικής EDFTB--, μειώνεται σχεδόν γραμμικά καθώς αυξάνεται ο αριθμός των αιτήσεων. Κάτι τέτοιο ισχύει και για τις περιοδικές αλλά και για τις απεριοδικές αιτήσεις. Από την άλλη πλευρά, η πολιτική EDFTB εγγυάται ότι όλες οι αιτήσεις που έγιναν αποδεκτές εκτελούνται έγκαιρα και έτσι η επίδοση του πρώτου κριτηρίου είναι 100%. Επίσης παρατηρούμε ότι το ποσοστό του δεύτερου κριτηρίου, μειώνεται σχεδόν γραμμικά και στις δύο πολιτικές. Όσο μειώνεται ο χρόνος εκτέλεσης C, αυξάνεται το ποσοστό επίδοσης των δύο κριτηρίων. Αυτό προκύπτει διότι καθώς μειώνονται οι χρόνοι εκτέλεσης των υπηρεσιών, μειώνεται και ο φόρτος του κάθε εξυπηρετητή με αποτέλεσμα να μπορεί ο κάθε ένας να εξυ-

πηρετεί επιτυχημένα περισσότερες αιτήσεις μέσα στο ίδιο χρονικό διάστημα. Στις τέσσερις πρώτες κατηγορίες μετρήσεων παρατηρούμε ότι η επίδοση του δεύτερου κριτηρίου είναι καλύτερη, όσον αφορά τις περιοδικές αιτήσεις, στην πολιτική EDFTB--, ενώ όσον αφορά τις απεριοδικές, η πολιτική EDFTB-- έχει καλύτερες επιδόσεις μόνο στις δύο πρώτες κατηγορίες μετρήσεων. Όπως φαίνεται και στο Σχήμα 5.20, η κατάσταση αλλάζει όσον αφορά την επίδοση του δεύτερου κριτηρίου. Παρατηρούμε ότι καθώς αυξάνεται ο αριθμός των εξυπηρετητών, η επίδοση του δεύτερου κριτηρίου στις περιοδικές αιτήσεις, όσον αφορά την πολιτική EDFTB--, παραμένει σχεδόν σταθερή, επιτυγχάνοντας πολύ μικρή αύξηση. Από την άλλη πλευρά, στην πολιτική EDFTB η αύξηση είναι αρκετά μεγάλη, ξεπερνώντας μάλιστα την επίδοση της πολιτικής EDFTB--, όταν έχουμε 48 εξυπηρετητές. Παρόμοια είναι και η κατάσταση στις απεριοδικές αιτήσεις, όπου οι επιδόσεις της πολιτικής EDFTB στο δεύτερο κριτήριο ξεπερνάνε αυτές της EDFTB--, όταν έχουμε περισσότερους από 12 εξυπηρετητές.

Συνεπώς, οι επιδόσεις της πολιτικής EDFTB στο πρώτο κριτήριο, υπερτερούν των επιδόσεων της EDFTB--, μιας και εγγυώνται 100% επιτυχία εκτέλεσης όλων των εργασιών που έχουν συμπεριληφθεί στο χρονοπρόγραμμα. Για να εγγυηθεί την 100% επίδοση του πρώτου κριτηρίου, η πολιτική EDFTB, αναγκάζεται να απορρίψει αρκετές αιτήσεις, κυρίως περιοδικές. Κατά την απόρριψη της αίτησης, ο πελάτης αναζητά κάποιον άλλο εξυπηρετητή ο οποίος θα μπορεί να εξυπηρετήσει την αίτηση. Στις πρώτες τέσσερις κατηγορίες μετρήσεων έχουμε μόνο 3 εξυπηρετητές με διαφορετικό χρόνο διάρκειας ζωής. Όπως είδαμε και στο Σχήμα 5.20, με μεγάλο αριθμό εξυπηρετητών αυξάνεται σημαντικά η επίδοση του δεύτερου κριτηρίου, στην περίπτωση της πολιτικής EDFTB, καθώς υπάρχουν περισσότεροι εναλλακτικοί εξυπηρετητές.

5.5. Σχολιασμός

Όπως είδαμε, σύμφωνα με τις παραπάνω πειραματικές μετρήσεις έγινε σύγκριση ανάμεσα στις πολιτικές με και χωρίς τις συνθήκες που προτείναμε. Με βάση τα αποτελέσματα στα δύο κριτήρια έγινε φανερό ότι οι πολιτικές που προτείναμε, εκτός της πολιτικής Lifetime που απλώς μας χρησίμευσε ως ένδειξη ότι πρέπει να συμπεριλαμβάνουμε υπόψη και το φόρτο του εξυπηρετητή, επιτυγχάνουν 100% επίδοση στο πρώτο κριτήριο. Ταυτόχρονα, επιτυγχάνουν επιδόσεις εφάμιλλες με αυτές των απλών πολιτικών και στο δεύτερο κριτήριο, καθιστώντας τες λειτουργικές.

Αν θέλαμε να επιλέξουμε μία πολιτική από τις 3 πολιτικές, LifetimeLoad, FIFO και EDFTB, τότε αρχικά θα πρέπει να προσδιορίζαμε εάν οι υπηρεσίες που θα παρέχουν οι εξυ-

πηρετητές θα είναι μόνο απεριοδικές ή συνδυασμός περιοδικών και απεριοδικών. Στην περίπτωση που θα απαιτούσαμε την ύπαρξη συνδυασμού περιοδικών και απεριοδικών υπηρεσιών, η μόνη επιλογή είναι η πολιτική EDFTB. Σε αντίθετη περίπτωση, αν επιθυμούσαμε την ύπαρξη μόνο απεριοδικών υπηρεσιών, τότε οι διαθέσιμες πολιτικές είναι η LifetimeLoad και η FIFO. Η επιλογή ανάμεσα στις δύο πολιτικές εξαρτάται, όπως φάνηκε και από τις μετρήσεις, από τον χρόνο εκτέλεσης των υπηρεσιών. Όσον αφορά το πρώτο κριτήριο, οι επιδόσεις είναι 100% οπότε δεν υπάρχει καμία διαφορά. Σχετικά με το δεύτερο κριτήριο, παρατηρούμε ότι στις δύο πρώτες κατηγορίες μετρήσεων οι επιδόσεις της πολιτικής FIFO είναι καλύτερες σε σχέση με την LifetimeLoad. Αντίθετα, στην τρίτη και τέταρτη κατηγορία, δηλαδή καθώς μικραίνει ο χρόνος εκτέλεσης των υπηρεσιών, υπερτερεί η πολιτική LifetimeLoad.

Ένα άλλο κριτήριο για την επιλογή της μίας ή της άλλης πολιτικής (ανάμεσα στις LifetimeLoad και FIFO), είναι και η υποστήριξη που παρέχει το υλικό στο οποίο υλοποιείται ο χρονοπρογραμματιστής. Για παράδειγμα, μπορεί να μην προσφέρεται υποστήριξη για προεκχωρητικό χρονοπρογραμματισμό με συνέπεια να αποκλείεται η δυνατότητα επιλογής της πολιτικής LifetimeLoad που βασίζεται στον αλγόριθμο Round Robin. Σε αυτήν την περίπτωση η μόνη δυνατή επιλογή θα ήταν η πολιτική FIFO.

ΚΕΦΑΛΑΙΟ 6. ΠΕΡΙΒΑΛΛΟΝ RTSJ

Όπως είδαμε στο Κεφάλαιο 5, παρουσιάστηκαν πειραματικές μετρήσεις της επίδοσης των πολιτικών που προτείναμε, οι οποίες υλοποιήθηκαν σε ένα περιβάλλον προσομοίωσης. Τα αποτελέσματα αυτά μας έδωσαν μια γενική εικόνα της συμπεριφοράς των πολιτικών και της απόδοσης τους. Ωστόσο στο προσομοιούμενο περιβάλλον οι συνθήκες εκτέλεσης του συστήματος ήταν ιδανικές. Πιο συγκεκριμένα, η καθυστέρηση που εισάγεται από τον χρονοπρογραμματιστή στο σύστημα θεωρήθηκε μηδενική. Στην πράξη βέβαια κάτι τέτοιο δεν ισχύει μιας και όπως θα δούμε, η καθυστέρηση του χρονοπρογραμματιστή στις πολιτικές που προτείναμε εξαρτάται από το μήκος της ουράς του. Για να μελετήσουμε στην πράξη τον βαθμό στον οποίο επηρεάζουν οι ανωτέρω συνθήκες την επίδοση των πολιτικών, υλοποιήσαμε την πολιτική EDFTB, που είναι και η πιο γενική από αυτές που προτείναμε, σε ένα πραγματικό σύστημα.

Η υλοποίηση βασίστηκε σε μια υλοποίηση του προτύπου *RTSJ (Real Time Specification for Java)*. Το πρότυπο RTSJ, επεκτείνει την γλώσσα προγραμματισμού Java έτσι ώστε να μπορεί να εκτελεί εφαρμογές που έχουν απαιτήσεις πραγματικού χρόνου. Η επιλογή του περιβάλλοντος RTSJ έγινε για τους εξής λόγους:

- Βασίζεται στην γλώσσα Java που λόγω της αντικειμενοστρέφειας και του υψηλού επιπέδου βιβλιοθηκών που παρέχει, ικανοποιεί τις ανάγκες της αρχιτεκτονικής της υπηρεσίας που προτείνουμε.
- Παρέχει όλα τα θεμελιώδη συστατικά, όπως νήματα πραγματικού χρόνου, χρονοπρογραμματιστές προτεραιοτήτων, μηχανισμούς συγχρονισμού κ.τ.λ, για την υλοποίηση μιας εφαρμογής πραγματικού χρόνου, με μεταφέρσιμο τρόπο. Συνεπώς η ίδια εφαρμογή μπορεί να εκτελεστεί χωρίς σχεδόν καμία αλλαγή, σε διαφορετικές αρχιτεκτονικές και σε διαφορετικά λειτουργικά συστήματα τα οποία εμπίπτουν στο πρότυπο RTSJ.

- Μπορεί να προσαρμοστεί εύκολα σε *ενσωματωμένα συστήματα (embedded systems)* και ιδιαίτερα σε κινητά τηλέφωνα νέας γενιάς και PDA, χάρη στην πλατφόρμα Java Micro Edition [21].

6.1. Βασικά Χαρακτηριστικά του Περιβάλλοντος RTSJ

Το πρότυπο RTSJ εμπλουτίζει την γλώσσα προγραμματισμού Java με τους εξής τρόπους[15]:

i. Προσθέτοντας *νήματα πραγματικού χρόνου (real-time threads)*. Σε αντίθεση με τα κοινά νήματα της Java, αυτού του τύπου τα νήματα έχουν χαρακτηριστικά χρονοπρογραμματισμού που έχουν οριστεί λαμβάνοντας υπόψη περιορισμούς πραγματικού χρόνου, όπως για παράδειγμα χρονικές προθεσμίες μέσα στις οποίες πρέπει να ολοκληρώσουν την εκτέλεση τους.

ii. Προσθέτοντας εργαλεία και μηχανισμούς που βοηθάνε τον προγραμματιστή να γράψει προγράμματα σε Java τα οποία δεν απαιτούν *συλλογή σκουπιδιών (garbage collection)*. Κάτι τέτοιο πραγματοποιείται ορίζοντας αντικείμενα με συγκεκριμένη *εμβέλεια (scope)* στην μνήμη.

iii. Προσθέτοντας μια κλάση *χειρισμού ασύγχρονων γεγονότων (asynchronous event handler)* και έναν μηχανισμό που συσχετίζει συμβάντα που λαμβάνουν χώρα εκτός της JVM, με ασύγχρονα γεγονότα της Java.

iv. Προσθέτοντας έναν μηχανισμό, που ονομάζεται *ασύγχρονη μεταβίβαση του ελέγχου (asynchronous transfer of control)*, ο οποίος επιτρέπει σε ένα νήμα να μεταβιβάσει την ροή του ελέγχου σε ένα άλλο νήμα. Ουσιαστικά, είναι ένας ελεγχόμενος τρόπος για να παράγει ένα νήμα μια *εξαίρεση (exception)* σε ένα άλλο νήμα.

v. Προσθέτοντας έναν μηχανισμό που επιτρέπει στον προγραμματιστή να έχει πρόσβαση σε διευθύνσεις της φυσικής μνήμης.

Για να μπορεί μια εφαρμογή να χρησιμοποιήσει τις παραπάνω δυνατότητες, θα πρέπει η *Εικονική Μηχανή Java (Java Virtual Machine – JVM)* στην οποία θα εκτελεστεί να υποστηρίζει το πρότυπο RTSJ.

Σε ένα περιβάλλον που δεν είναι πραγματικού χρόνου, όπως μια τυπική JVM, ο χρονοπρογραμματισμός μπορεί να γίνεται με απλό τρόπο, αλλά σε ένα περιβάλλον πραγματικού χρόνου πρέπει να γίνεται με ακριβή και καθορισμένο τρόπο. Το πρότυπο RTSJ παρέχει *χρονοπρογραμματισμό προτεραιοτήτων (priority scheduling)* διότι χρησιμοποιείται σχεδόν καθολικά στα εμπορικά συστήματα πραγματικού χρόνου και επειδή όλες οι τυπικές εφαρμογές Java χρησιμοποιούν χρονοπρογραμματισμό προτεραιοτήτων. Το πρότυπο RTSJ απαιτεί από

μία υλοποίησή του, την παροχή τουλάχιστον 28 προτεραιότητες πραγματικού χρόνου επιπρόσθετα από τις 10 προτεραιότητες που απαιτούνται από τις προδιαγραφές μιας τυπικής JVM. Επίσης, απαιτεί αυστηρό προεκχωρητικό χρονοπρογραμματισμό σταθερών προτεραιοτήτων πραγματικού χρόνου. Κάτι τέτοιο σημαίνει ότι ένα νήμα χαμηλότερης προτεραιότητας δεν θα πρέπει να εκτελείται όταν είναι έτοιμο προς εκτέλεση ένα νήμα υψηλότερης προτεραιότητας.

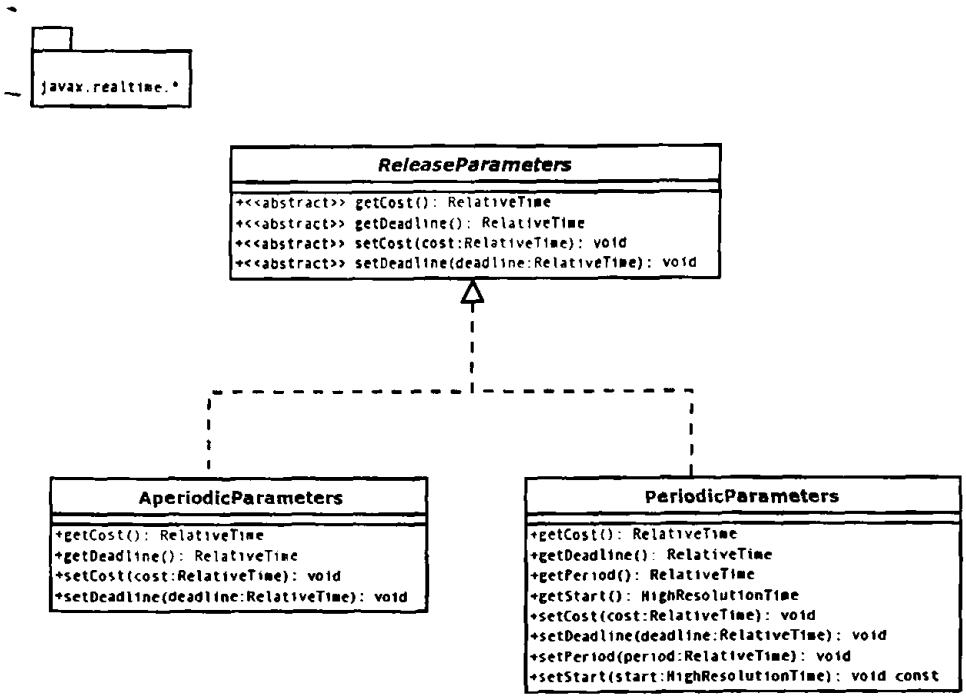
Μια υλοποίηση του προτύπου RTSJ θα πρέπει να συμπεριλαμβάνει τα πακέτα *javax.realtime*. Πρέπει επίσης να παρέχει και μια ειδική JVM, και ίσως και έναν *μεταγλωττιστή (compiler)* που μπορεί να βοηθήσει στην εύρεση *σφαλμάτων αναφοράς στην μνήμη (memory reference errors)*. Η παροχή ενός ειδικού μεταγλωττιστή είναι προαιρετική μιας και οποιοσδήποτε Java μεταγλωττιστής μπορεί να παράγει κλάσεις που χρησιμοποιούν το RTSJ. Μια πλήρης υλοποίηση του προτύπου RTJS περιέχει προσθήκες και βελτιώσεις στην JVM. Το πρότυπο δεν προσθέτει νέα bytecodes, αλλά θα είναι δύσκολο ή σχεδόν αδύνατο να επιβληθούν κανόνες πρόσβασης στην φυσική μνήμη χωρίς να ενισχυθεί η υλοποίηση των bytecodes που χειρίζονται τις αναφορές.

6.1.1. Νήματα Πραγματικού Χρόνου

Όλες οι εργασίες σε μια Java εφαρμογή πραγματικού χρόνου, γίνονται μέσω των νημάτων πραγματικού χρόνου και μέσω των χειριστών ασύγχρονων γεγονότων, οι οποίοι είναι αρκετά παρόμοιοι με τα νήματα πραγματικού χρόνου. Η κλάση *javax.realtime.RealtimeThread* επεκτείνει την κλάση *java.lang.Thread*, συνεπώς τα νήματα πραγματικού χρόνου μπορούν να χρησιμοποιηθούν οπουδήποτε απαιτείται ένα νήμα, αλλά πολλά από τα χαρακτηριστικά του περιβάλλοντος RTSJ θα είναι διαθέσιμα μόνο στα νήματα πραγματικού χρόνου, όπως επιπρόσθετες προτεραιότητες και περιοδική εκτέλεση.

Η προγραμματιστική *διασύνδεση (interface)* της κλάσης *RealtimeThread* παρέχει μια σειρά από *κατασκευαστές (constructors)* και *μεθόδους*, μέσω των οποίων μπορούμε να καθορίσουμε την συμπεριφορά του νήματος και τα χαρακτηριστικά χρονοπρογραμματισμού του. Πιο συγκεκριμένα, μέσω της κλάσης *PriorityParameters* μπορούμε να καθορίσουμε την προτεραιότητα ενός νήματος. Η προτεραιότητα ορίζεται ως ένας ακέραιος αριθμός μεταξύ μιας ελάχιστης και μιας μέγιστης τιμής. Όσο μεγαλύτερος είναι ο αριθμός αυτός, τόσο μεγαλύτερη είναι και η προτεραιότητα του νήματος. Η *αφηρημένη (abstract)* κλάση *ReleaseParameters* ορίζει μια διασύνδεση που καθορίζει τις παραμέτρους *έναρξης (release)* του νήματος. Οι παράμετροι αυτοί περιλαμβάνουν τον καθορισμό του κόστους και της προθεσμίας του νήματος.

Το κόστος ενός νήματος ουσιαστικά αποτελεί τον χρόνο εκτέλεσης *C* της χειρότερης περίπτωσης. Στην περίπτωση που το νήμα εκτελείται περιοδικά τότε πρέπει να χρησιμοποιηθεί η υποκλάση της *ReleaseParameters*, η *PeriodicParameters*, μέσω της οποίας καθορίζεται η περίοδος του νήματος, η χρονική στιγμή κατά την οποία θα ξεκινήσει η εκτέλεση του νήματος, καθώς και το κόστος και η προθεσμία του. Από την άλλη εάν το νήμα είναι απεριοδικό τότε θα πρέπει να χρησιμοποιηθεί η κλάση *AperiodicParameters*, που είναι και αυτή υποκλάση της *ReleaseParameters*. Αυτού του είδους η κλάση χαρακτηρίζει ένα νήμα το οποίο μπορεί να ξεκινήσει την εκτέλεση του οποιαδήποτε χρονική στιγμή.



Σχήμα 6.1 Υποκλάσεις της *ReleaseParameters*.

Ένα νήμα πραγματικού χρόνου πρέπει να υλοποιεί τουλάχιστον την μέθοδο *run()* στην οποία θα υπάρχει η λογική του νήματος, δηλαδή ο κώδικας της εργασίας που πρέπει να επιτελέσει το νήμα. Ουσιαστικά η μέθοδος *run* αποτελεί το *σημείο εισόδου (entry point)* του νήματος. Στο παράδειγμα 6.1 φαίνεται η δομή ενός απλού περιοδικού νήματος. Ο έλεγχος του προγράμματος εισέρχεται στο νήμα την χρονική στιγμή έναρξης του, που έχει οριστεί στην κλάση *PeriodicParameters*, και εκτελείται η πρώτη περίοδος. Η εκτέλεση συνεχίζεται έως ότου ο έλεγχος φτάσει στην κλήση της μεθόδου *waitForNextPeriod()*, όπου σε εκείνο το σημείο σταματάει μέχρι να έρθει η χρονική στιγμή για την έναρξη της επόμενης περιόδου.

```
public void run()
{
  while(true)
  {
    do
    {
      /* εδώ υπάρχουν εντολές για την υλοποίηση
           της εργασίας του νήματος */

    } while(waitForNextPeriod());
  }
  return;
}
```

Σχήμα 6.2 Βασική Δομή ενός Περιοδικού Νήματος.

6.1.2. Χρονοπρογραμματιστής Προτεραιοτήτων

Η αφηρημένη κλάση *Scheduler* καθορίζει τις βασικές μεθόδους που χειρίζονται την εκτέλεση των νημάτων πραγματικού χρόνου. Μέσω της κλάσης αυτής υλοποιείται ένας αλγόριθμος *εφικτότητας (feasibility)*. Ο αλγόριθμος εφικτότητας προσδιορίζει εάν το σύνολο των νημάτων, βάσει των προτεραιοτήτων τους, είναι εφικτό.

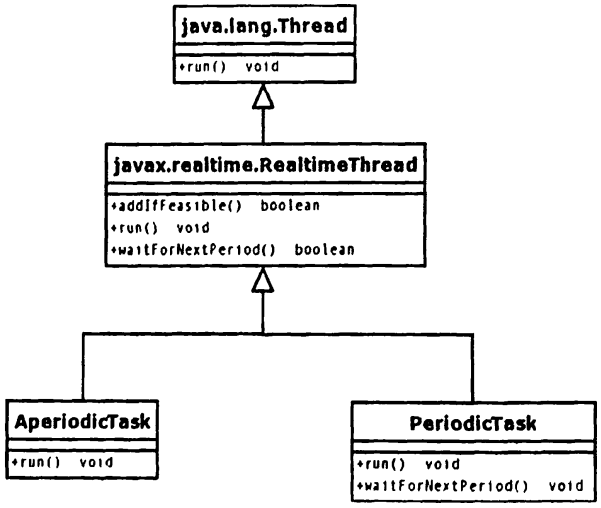
Υποκλάσεις της κλάσης *Scheduler* χρησιμοποιούνται για την υλοποίηση των πολιτικών χρονοπρογραμματισμού και πρέπει τουλάχιστον να έχουν ορίσει την μέθοδο *instance()* που επιστρέφει το προκαθορισμένο στιγμιότυπο της υποκλάσης. Η υποκλάση *PriorityScheduler* υλοποιεί έναν προεκχωρητικό χρονοπρογραμματιστή προτεραιοτήτων και το πρότυπο RTSJ ορίζει ότι όλες οι υλοποιήσεις του προτύπου θα πρέπει να υλοποιούν τουλάχιστον αυτήν την υποκλάση χρονοπρογραμματιστή.

6.2. Υλοποίηση της Πολιτικής EDFTB

Η υλοποίηση της πολιτικής EDFTB έγινε χρησιμοποιώντας μια υλοποίηση του προτύπου RTSJ [23] για το λειτουργικό σύστημα GNU/Linux. Για την υλοποίηση της πολιτικής EDFTB απαιτείται ο χρονοπρογραμματιστής EDF ο οποίος αναθέτει δυναμικά προτεραιότητες στα νήματα που συμπεριλαμβάνονται στο χρονοπρόγραμμα. Όπως αναφέρθηκε παραπάνω, το πρότυπο RTSJ ορίζει έναν χρονοπρογραμματιστή σταθερών προτεραιοτήτων, μέσω της κλάσης *PriorityScheduler*. Η υλοποίηση του χρονοπρογραμματιστή EDF βασίστηκε στην κλάση *PriorityScheduler*. Πιο συγκεκριμένα, ορίστηκε η κλάση *EDFTBScheduler* η οποία

επεκτείνει την κλάση του προκαθορισμένου χρονοπρογραμματιστή. Ο ορισμός της κλάσης είναι ο εξής: *public class EDFTBScheduler extends PriorityScheduler*. Συνεπώς στην υποκλάση *EDFTBScheduler* υλοποιήθηκαν οι απαραίτητες μέθοδοι, που *υπερισχύουν (override)* τις αντίστοιχες της κλάσης *PriorityScheduler*, έτσι ώστε να υλοποιηθεί η πολιτική χρονοπρογραμματισμού που καθορίζεται από τον αλγόριθμο EDF. Στο Σχήμα 6.4 απεικονίζεται η ιεραρχία των κλάσεων χρονοπρογραμματισμού. Οι κλάσεις *Scheduler* και *PriorityScheduler* παρέχονται από το πρότυπο RTSJ, ενώ οι κλάσεις *EDFTBScheduler*, *FIFOScheduler* και *LifetimeLoadScheduler* περιγράφουν την διασύνδεση των χρονοπρογραμματιστών των αντίστοιχων πολιτικών που έχουν μελετηθεί στο Κεφάλαιο 4. Στην εργασία αυτή, έχουμε υλοποιήσει μόνο την πολιτική EDFTB λόγω της υποστήριξης της για περιοδικές και απεριοδικές υπηρεσίες. Επίσης, στην ιεραρχία δεν συμπεριλαμβάνεται η πολιτική *Lifetime* λόγω της απλότητας της.

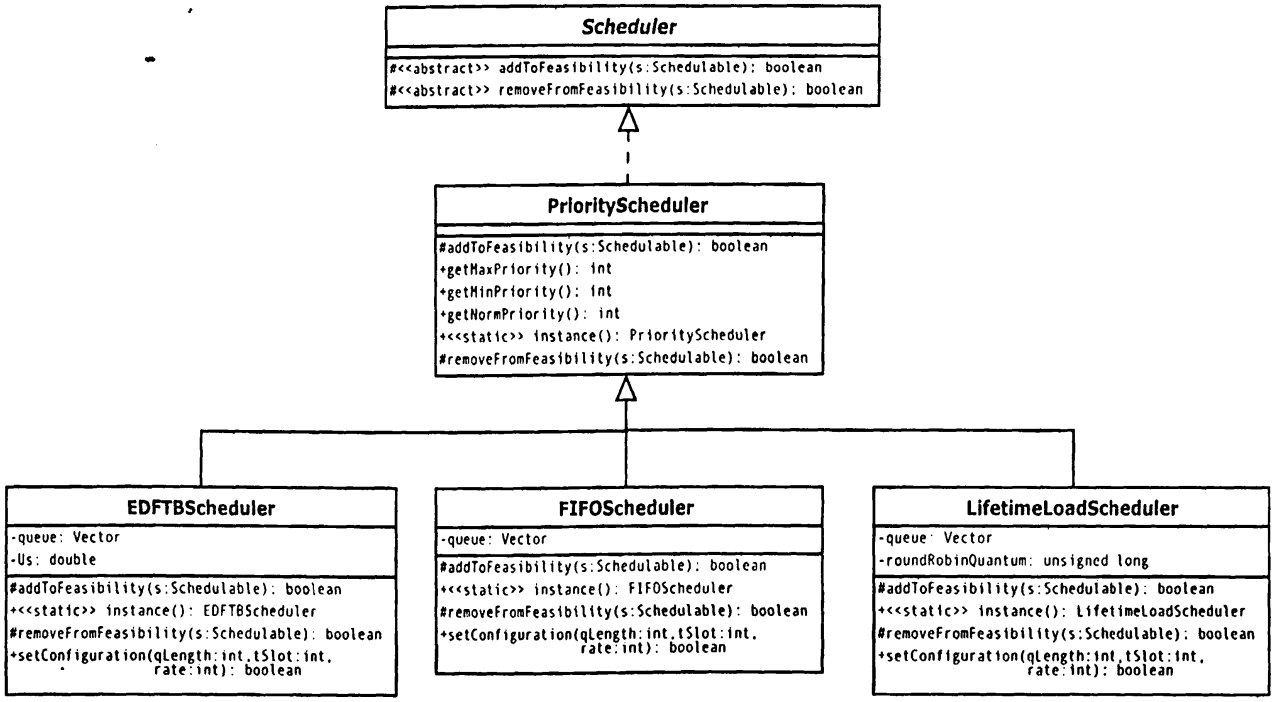
Η πολιτική EDFTB υποστηρίζει τον από κοινού χρονοπρογραμματισμό περιοδικών και απεριοδικών υπηρεσιών. Συνεπώς, έχουν υλοποιηθεί δύο κλάσεις για αυτόν τον σκοπό. Οι κλάσεις *AperiodicTask* και *PeriodicTask*, που και οι δύο επεκτείνουν την κλάση *RealtimeThread*. Ένα στιγμιότυπο της κλάσης *AperiodicTask* αποτελεί μια απεριοδική εργα-



Σχήμα 6.3 Ιεραρχία Κλάσεων *PeriodicTask* και *AperiodicTask*.

σία που εκτελείται μία φορά. Μέσω του κατασκευαστή της κλάσης *AperiodicTask* καθορίζεται ο χρόνος εκτέλεσής της στην χειρότερη περίπτωση καθώς και η χρονική στιγμή έναρξής της. Αντίστοιχα, ένα στιγμιότυπο της κλάσης *PeriodicTask* αποτελεί μια περιοδική εργασία. Μέσω του κατασκευαστή της κλάσης μπορούμε να καθορίσουμε τον χρόνο εκτέλεσης στην χειρότερη περίπτωση, την περίοδο, την προθεσμία της και τον αριθμό των περιόδων που θα εκτελεστεί η εργασία.

Σύμφωνα με την αρχιτεκτονική που περιγράφηκε στο Κεφάλαιο 3, μια οντότητα-εξυπηρετητής προσφέρει μια σειρά από υπηρεσίες τις οποίες μπορεί να καλέσει μια οντότητα-πελάτης μέσω του αποτιμητή της. Η κλάση *Assessor* υλοποιεί τον αποτιμητή, ο οποίος αποστέλλει αιτήσεις στους διαθέσιμους εξυπηρετητές. Η επικοινωνία ανάμεσα στον αποτιμητή και στον εξυπηρετητή γίνεται χρησιμοποιώντας *υποδοχείς (sockets)* σε περιβάλλον ασύρματου δικτύου WLAN.



Σχήμα 6.4 Ιεραρχία Κλάσεων Χρονοπρογραμματισμού.

Από την άλλη, η κλάση *Server* συμπεριλαμβάνει ένα στιγμιότυπο του χρονοπρογραμματιστή *EDFTBScheduler* έτσι ώστε κάθε εξυπηρετητής να περιέχει και έναν χρονοπρογραμματιστή που υλοποιεί την πολιτική EDFTB. Ο κάθε εξυπηρετητής προσφέρει κάποιες συγκεκριμένες υπηρεσίες τις οποίες υλοποιεί μέσω των κλάσεων *AperiodicTask* και *PeriodicTask*, ανάλογα με το εάν οι υπηρεσίες εκτελούνται απεριοδικά ή περιοδικά. Η συμπεριφορά και η λειτουργία των υπηρεσιών αυτών είναι καθορισμένη, συνεπώς είναι γνωστός ο χρόνος εκτέλεσης τους C στην χειρότερη περίπτωση, δηλαδή το κόστος τους. Για κάθε αίτηση που καταφθάνει από τον αποτιμητή του πελάτη, ο εξυπηρετητής δημιουργεί ένα αντικείμενο *r*, τύπου *AperiodicTask* ή *PeriodicTask*, ανάλογα με το είδος της υπηρεσίας, και καλεί την μέθοδο *addIfFeasible()* του αντικειμένου *r*, η οποία επιστρέφει την τιμή *true* εάν το αντικείμενο προστέθηκε στο χρονοπρόγραμμα ή την τιμή *false* σε αντίθετη περίπτωση. Αυτή η μέθοδος πρώτα καλεί αυτόματα την μέθοδο *addToFeasibility()* του χρονοπρογραμματιστή στην οποία ο χρονοπρογραμματιστής υλοποιεί τον αλγόριθμο εφικτότητας. Συγκεκριμένα, σε αυτήν την

μέθοδο ο χρονοπρογραμματιστής ελέγχει εάν με την προσθήκη του νέου αντικειμένου συνεχίζουν να ισχύουν οι συνθήκες της πολιτικής EDFTB. Σε περίπτωση που το αντικείμενο αντιστοιχεί σε μια απεριοδική εργασία τότε του αποδίδεται μια προθεσμία βάσει του μηχανισμού TB.

Εάν το αντικείμενο δεν παραβιάζει τις συνθήκες της πολιτικής, τότε προστίθεται στο χρονοπρόγραμμα και βάσει του αλγορίθμου EDF ελέγχεται εάν η προθεσμία του είναι η μικρότερη ανάμεσα στις προθεσμίες όλων των εργασιών που περιέχονται στο χρονοπρόγραμμα. Σε αυτήν την περίπτωση θα πρέπει να διακοπεί η εκτέλεση της εργασίας που εκτελείται εκείνη την στιγμή, για να ξεκινήσει η εκτέλεση της νέας εργασίας. Η εργασία που διακόπτεται τοποθετείται σε μια ουρά *Q* του χρονοπρογραμματιστή, όπου αποθηκεύονται όλες οι *εν αναμονή* εργασίες. Η διακοπή της εργασίας που εκτελείται, γίνεται μέσω του μηχανισμού *ασύγχρονης μεταβίβασης ελέγχου* που παρέχει το πρότυπο RTSJ. Σύμφωνα με αυτόν τον μηχανισμό, ο χρονοπρογραμματιστής αποστέλλει μια ειδικού τύπου εξαίρεση στο νήμα της εργασίας που θέλει να διακόψει, με αποτέλεσμα να μεταφερθεί ο έλεγχος του προγράμματος στον κώδικα του νήματος. Το νήμα εμπεριέχει ένα αντικείμενο της κλάσης *EventVariable*, την οποία έχουμε υλοποιήσει έτσι ώστε να λειτουργεί ως *κλειδαριά (lock)*, με την βοήθεια του οποίου θα γίνει η *αναστολή (suspend)* του νήματος. Η βασική μέθοδος της κλάσης *EventVariable* παρου-

```
public class EventVariable
{
   public synchronized void await()
   {
      this.wait();
   }
}
```

Σχήμα 6.5 Μέθοδος *await()*.

σιάζεται με συνοπτικό τρόπο στο Σχήμα 6.5. Το νήμα καλεί την μέθοδο *await()* στο αντικείμενο της κλάσης *EventVariable* για να αναστείλει τον εαυτό του. Η μέθοδος *await()* έχει οριστεί ως *synchronized* και καλεί την μέθοδο *wait()*. Κάθε αντικείμενο της Java κληρονομεί την μέθοδο *wait()* από την κλάση *java.lang.Object*, η οποία αποτελεί την υπερκλάση όλων των αντικειμένων στην γλώσσα Java. Η λέξη-κλειδί *synchronized* χρησιμοποιείται για να καθορίσει ότι μια μέθοδος ή ένα μπλοκ κώδικα μπορεί να κληθεί από ένα μόνο νήμα ταυτόχρονα. Η είσοδος στον κώδικα της μεθόδου προστατεύεται από έναν *παρακολουθητή (monitor)*. Σύμφωνα με τις προδιαγραφές της Java, κάθε αντικείμενο έχει έναν παρακολουθητή. Ένα νήμα γίνεται ο κάτοχος του παρακολουθητή του αντικειμένου με τους παρακάτω τρόπους [22]:

i. Εκτελώντας μια *synchronized* μέθοδο του αντικειμένου.
ii. Εκτελώντας ένα *synchronized* μπλοκ κώδικα του αντικειμένου.

iii. Εκτελώντας μια στατική *synchronized* μέθοδο αντικειμένων τύπου *Class*.

Μόνο ένα νήμα τη φορά μπορεί να κατέχει τον παρακολουθητή ενός αντικειμένου. Συνεπώς, η κλήση της μεθόδου *wait()* μέσα από μια *synchronized* μέθοδο, *προκαλεί το τρέχον νήμα να κοιμηθεί*. Για να ξυπνήσει ένα νήμα, ο χρονοπρογραμματιστής καλεί την μέθοδο *notify()* στο αντικείμενο του οποίου κλήθηκε η μέθοδος *await()*. Η μέθοδος *notify()* ξυπνάει ένα νήμα που περιμένει. Κάθε αντικείμενο της Java κληρονομεί την μέθοδο *notify()* από την κλάση *Object*.

Λόγω του ότι η πολιτική EDFTB υλοποιείται μέσω του χρονοπρογραμματιστή προτεραιοτήτων, έχουμε αντιστοιχήσει προτεραιότητες στις εργασίες που χειρίζεται ο χρονοπρογραμματιστής *EDFTBScheduler*. Πιο συγκεκριμένα, οι απεριοδικές εργασίες έχουν σαν προτεραιότητα, την προτεραιότητα που επιστρέφει η μέθοδος *getMinPriority()* του χρονοπρογραμματιστή *PriorityScheduler*, όπως φαίνεται και στο Σχήμα 6.5. Η προτεραιότητα αυτή είναι πάντα μεγαλύτερη από τις 10 "κοινές" προτεραιότητες μιας τυπικής JVM και συνήθως ισούται με τον αριθμό 11. Σημειώνουμε, ότι σύμφωνα με την γλώσσα Java, όσο μεγαλύτερος είναι ο αριθμός της προτεραιότητας τόσο μεγαλύτερη είναι και η προτεραιότητα εκτέλεσης του νήματος. Οι περιοδικές εργασίες έχουν σαν προτεραιότητα τον αριθμό *getMinPriority()* + 2, ενώ όταν μια εργασία εκτελείται, ανεξάρτητα εάν είναι περιοδική ή απεριοδική, αποκτά σαν προτεραιότητα την *getMinPriority()* + 1. Συνεπώς, έχουμε τρεις κατηγορίες προτεραιοτήτων που χρησιμοποιούνται στον χρονοπρογραμματιστή *EDFTBScheduler*. Όπως αναφέραμε, οι περιοδικές εργασίες έχουν την μεγαλύτερη προτεραιότητα, ακόμα και από αυτές που εκτελούνται. Κάτι τέτοιο δεν σημαίνει ότι θα εκτελούνται αυτές αντί της εργασίας που πρέπει να εκτελεστεί, εφόσον εάν βρίσκονται στην ουρά του χρονοπρογραμματιστή θα έχουν ανασταλλεί και θα δεν θα έχουν δικαίωμα εκτέλεσης. Ο λόγος που έχουν μεγαλύτερη προτεραιότητα, είναι για να μπορούν να ενεργοποιηθούν μόλις έρθει η χρονική στιγμή για την έναρξη της επόμενης περιόδου. Έτσι, ο χρονοπρογραμματιστής θα μπορεί να ελέγξει εάν το περιοδικό νήμα που μόλις ενεργοποιήθηκε, έχει την μικρότερη προθεσμία ή όχι. Όπως έχει αναφερθεί, μόλις ολοκληρωθεί μία περίοδος ενός περιοδικού νήματος, τότε καλούμε την μέθοδο *waitForNextPeriod()* και το νήμα απενεργοποιείται.

Μόλις ολοκληρωθεί η εκτέλεση ενός νήματος, αποστέλλεται μια απάντηση με τα αποτελέσματα πίσω στον αποτιμητή της οντότητας-πελάτη. Κάθε απάντηση απαιτεί κάποιον χρόνο για να φτάσει πίσω στον αποτιμητή ο οποίος είναι ανάλογος με το μέγεθος της. Όπως αναφέρθηκε στο Κεφάλαιο 4, σε ορισμένες συνθήκες της πολιτικής EDFTB συμπεριλαμβάνεται και το κόστος c_{rep} για την αποστολή της απάντησης στον πελάτη, όπως στην παρακάτω συνθήκη:

$$m_{server}Lifetime \leq m_{client}Lifetime - c_{rep}$$

Επειδή οι υπηρεσίες που προσφέρει ο εξυπηρετητής είναι προκαθορισμένες, είναι γνωστό το μέγεθος της κάθε απάντησης, για την κάθε υπηρεσία, στην χειρότερη περίπτωση. Σύμφωνα με την αρχιτεκτονική που έχουμε προτείνει, όλα τα μηνύματα συγχρονίζονται μέσω ενός σημείου πρόσβασης χρησιμοποιώντας τον μηχανισμό PCF. Συνεπώς, όπως έχει αναφερθεί και στο Κεφάλαιο 3, το κόστος c_{rep} υπολογίζεται ως εξής:

$$c_{rep} = n_s * N * t_{slot}$$

Αν υποθέσουμε ότι το μέγεθος της απάντησης είναι αρκετά μεγάλο, με συνέπεια να απαιτηθεί παραπάνω του ενός χρονικού slot για την μετάδοση της, τότε θα χρειαστεί να αποθηκευτεί σε μια *ενδιάμεση μνήμη (buffer)* μέχρι να ολοκληρωθεί η μετάδοση της. Κατά την διάρκεια αναμονής της απάντησης στην ενδιάμεση μνήμη για την πλήρη μετάδοση της, μπορεί να προστεθούν και άλλα μηνύματα απαντήσεων λόγω της ολοκλήρωσης άλλων εργασιών. Συνεπώς, εάν δεν είναι άδεια η ενδιάμεση μνήμη, τότε θα υπάρξει επιπλέον καθυστέρηση για μια απάντηση όταν αυτή προστεθεί στην ενδιάμεση μνήμη. Υποθέτοντας ότι η ενδιάμεση μνήμη λειτουργεί με τρόπο FIFO, η καθυστέρηση θα εξαρτάται από τον χρόνο που θα παραμείνουν οι προηγούμενες απαντήσεις στην ενδιάμεση μνήμη. Στην πράξη, το μέγεθος της ενδιάμεσης μνήμης είναι πεπερασμένο και έτσι μπορεί να υπολογιστεί ο χρόνος καθυστέρησης μιας απάντησης, στην χειρότερη περίπτωση.

Σύμφωνα με τον αλγόριθμο EDF, θα πρέπει κατά την άφιξη μιας εργασίας να υπολογίζονται οι τρέχουσες προθεσμίες όλων των *εν αναμονή* εργασιών που βρίσκονται στην ουρά Q. Η προσπέλαση ολόκληρης της ουράς προκαλεί χρονική επιβάρυνση στον χρονοπρογραμματιστή, η οποία είναι ανάλογη με το μήκος της ουράς. Συνεπώς, αν η ουρά έχει n στοιχεία, τότε το κόστος θα είναι της τάξης $\Theta(n)$. Αν ο λόγος της μέγιστης χρονικής επιβάρυνσης προς τον μικρότερο χρόνο εκτέλεσης C ανάμεσα σε όλες τις εργασίες, είναι μεγάλος, τότε η επιπλέον επιβάρυνση μπορεί να προκαλέσει σε κάποιες εργασίες να χάσουν την προθεσμία τους. Συνεπώς, θα πρέπει να ορίζεται μια μέγιστη τιμή για το μέγεθος της ουράς, τέτοια ώστε η επιπλέον επιβάρυνση να είναι σχεδόν αμελητέα σε σχέση με τους χρόνους εκτέλεσης των εργασιών.

Για τον καθορισμό όλων των ανωτέρω παραμέτρων, που επηρεάζουν τον χρονοπρογραμματιστή, η κλάση *EDFTBScheduler* διαθέτει την μέθοδο *setConfiguration()* μέσω της οποίας μπορούν να καθοριστούν οι τιμές των παραμέτρων όπως είναι το μέγιστο μήκος της ουράς, ο ρυθμός μετάδοσης του μέσου και ο χρόνος του κάθε slot.

6.2.1. Αποτίμηση της Υλοποίησης

Σε αυτήν την ενότητα παρουσιάζονται πειραματικά αποτελέσματα της επίδοσης της πολιτικής EDFTB σε περιβάλλον RTSJ. Οι πειραματικές μετρήσεις που πραγματοποιήθηκαν χωρίστηκαν σε δύο κατηγορίες. Στην πρώτη κατηγορία οι μετρήσεις έγιναν χρησιμοποιώντας μια οντότητα-πελάτη και μία οντότητα-εξυπηρετητή. Στην δεύτερη κατηγορία, χρησιμοποιήθηκαν δύο οντότητες-εξυπηρετητές, έτσι ώστε ο πελάτης να μπορεί να έχει δύο εναλλακτικές επιλογές σε περίπτωση που ο ένας εξυπηρετητής δεν μπορεί να εξυπηρετήσει την αίτηση του. Και στις δύο περιπτώσεις, ο παράγοντας αξιοποίησης *Us* του εξυπηρετητή TB ήταν 0,25 και τα κριτήρια που χρησιμοποιήθηκαν για την μέτρηση της επίδοσης των πολιτικών είναι τα ίδια με αυτά του Κεφαλαίου 5. Το πρώτο κριτήριο είναι το κλάσμα του αριθμού των αιτήσεων που ολοκληρώθηκαν έγκαιρα, προς τον αριθμό των αιτήσεων που έγιναν αποδεκτές και προστέθηκαν στο χρονοπρόγραμμα του χρονοπρογραμματιστή. Το δεύτερο κριτήριο είναι το κλάσμα του αριθμού των αιτήσεων που ολοκληρώθηκαν έγκαιρα, προς τον συνολικό αριθμό των αιτήσεων που πραγματοποίησαν οι πελάτες σε όλους τους εξυπηρετητές. Επίσης, μετρήθηκε η χρονική επιβάρυνση που αποφέρει η ουρά του χρονοπρογραμματιστή, καθώς επίσης μετρήθηκαν και οι απαιτήσεις σε μνήμη του εξυπηρετητή.

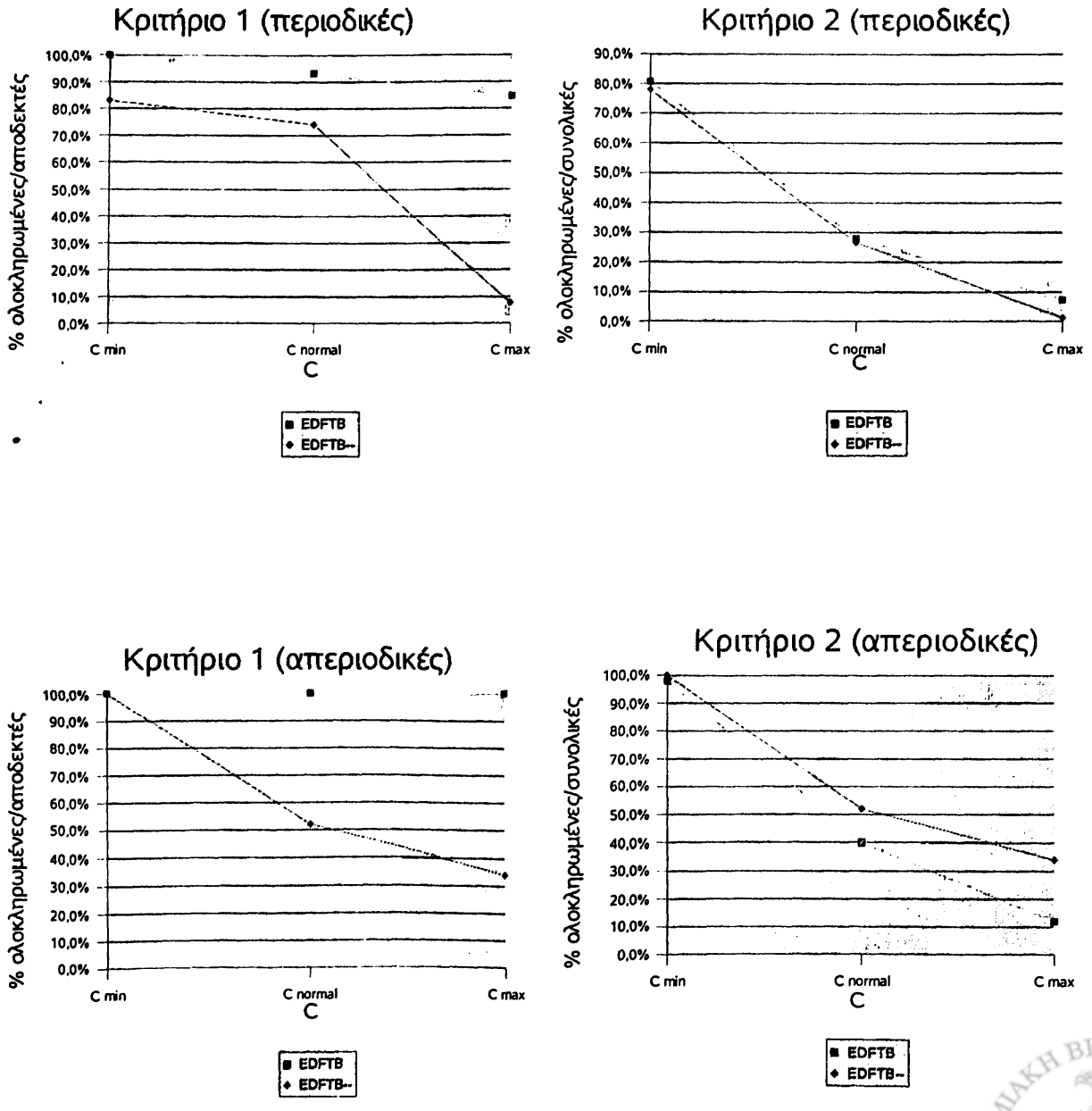
Για την πραγματοποίηση των μετρήσεων χρησιμοποιήθηκαν 3 διαφορετικοί προσωπικοί υπολογιστές που εκτελούσαν το λειτουργικό σύστημα GNU/Linux με πυρήνα της έκδοσης 2.6. Η ταχύτητα των επεξεργαστών ήταν αντίστοιχα 1,4 Ghz, 1,5 Ghz και 3,3 Ghz. Ο κάθε εξυπηρετητής και ο πελάτης εκτελούνταν σε διαφορετική JVM, η οποία εκτελούταν σε διαφορετικό υπολογιστή, έτσι ώστε να έχει στην διάθεση του την πλήρη αξιοποίηση του επεξεργαστή. Η κάθε JVM εκτελούνταν με προνόμια *υπερχρήστη (root)* και είχε την μεγαλύτερη προτεραιότητα ανάμεσα στις διεργασίες του λειτουργικού συστήματος.

6.2.1.1. Μετρήσεις με έναν πελάτη και έναν εξυπηρετητή

Σε αυτήν την περίπτωση οι μετρήσεις πραγματοποιήθηκαν με τις ακόλουθες παραμέτρους: η διάρκεια ζωής του πελάτη είναι 300.000 ms, του εξυπηρετητή 100.000 ms και ο ρυθμός δημιουργίας των αιτήσεων, από τον αποτιμητή του πελάτη προς τον εξυπηρετητή, είναι ανά 500 ms. Ο εξυπηρετητής παρέχει 3 περιοδικές υπηρεσίες και 1 απεριοδική. Το 75% των αιτήσεων του αποτιμητή αφορούν περιοδικές υπηρεσίες του εξυπηρετητή και το 25% απεριοδικές. Οι πειραματικές μετρήσεις πραγματοποιήθηκαν χρησιμοποιώντας τρία διαφορετικά σύνολα χρόνων εκτέλεσης για τις υπηρεσίες, τα C_{min}, C_{normal} και C_{max}. Όσον αφορά τις περιοδικές υπηρεσίες, στο σύνολο C_{min} οι χρόνοι εκτέλεσης είναι μικροί (της τάξης των 50 ms), στο C_{normal} είναι της τάξης των 150 ms και στο C_{max} είναι της τάξης των 450 ms (δηλ. σε κάθε

σύνολο οι χρόνοι εκτέλεσης τριπλασιάζονται). Η περίοδος για την πρώτη υπηρεσία είναι 2800 ms, για την δεύτερη 1400 ms και για την τρίτη 700 ms. Ο αριθμός εκτελέσεων των περιόδων της κάθε περιοδικής υπηρεσίας είναι 10. Συνεπώς, η κάθε υπηρεσία αποφέρει διαφορετικό φόρτο στον εξυπηρετητή.

Αντίστοιχα, όσον αφορά την απεριοδική υπηρεσία, στο σύνολο C_{min} ο χρόνος εκτέλεσης είναι της τάξης των 400 ms, στο C_{normal} είναι της τάξης των 1200 ms και στο C_{max} είναι της τάξης των 3600 ms (δηλ. σε κάθε σύνολο ο χρόνος εκτέλεσης τριπλασιάζεται). Ο χρόνος εκτέλεσης της απεριοδικής υπηρεσίας είναι μεγαλύτερος σε σχέση με των περιοδικών, διότι κάθε στιγμιότυπο της εκτελείται μία φορά, για μία περίοδο, και συνήθως οι απεριοδικές υπηρεσίες εκτελούν εργασίες που εκτελούνται λιγότερο συχνά σε σχέση με τις περιοδικές.



Σχήμα 6.6 Πρώτη Κατηγορία Μετρήσεων.

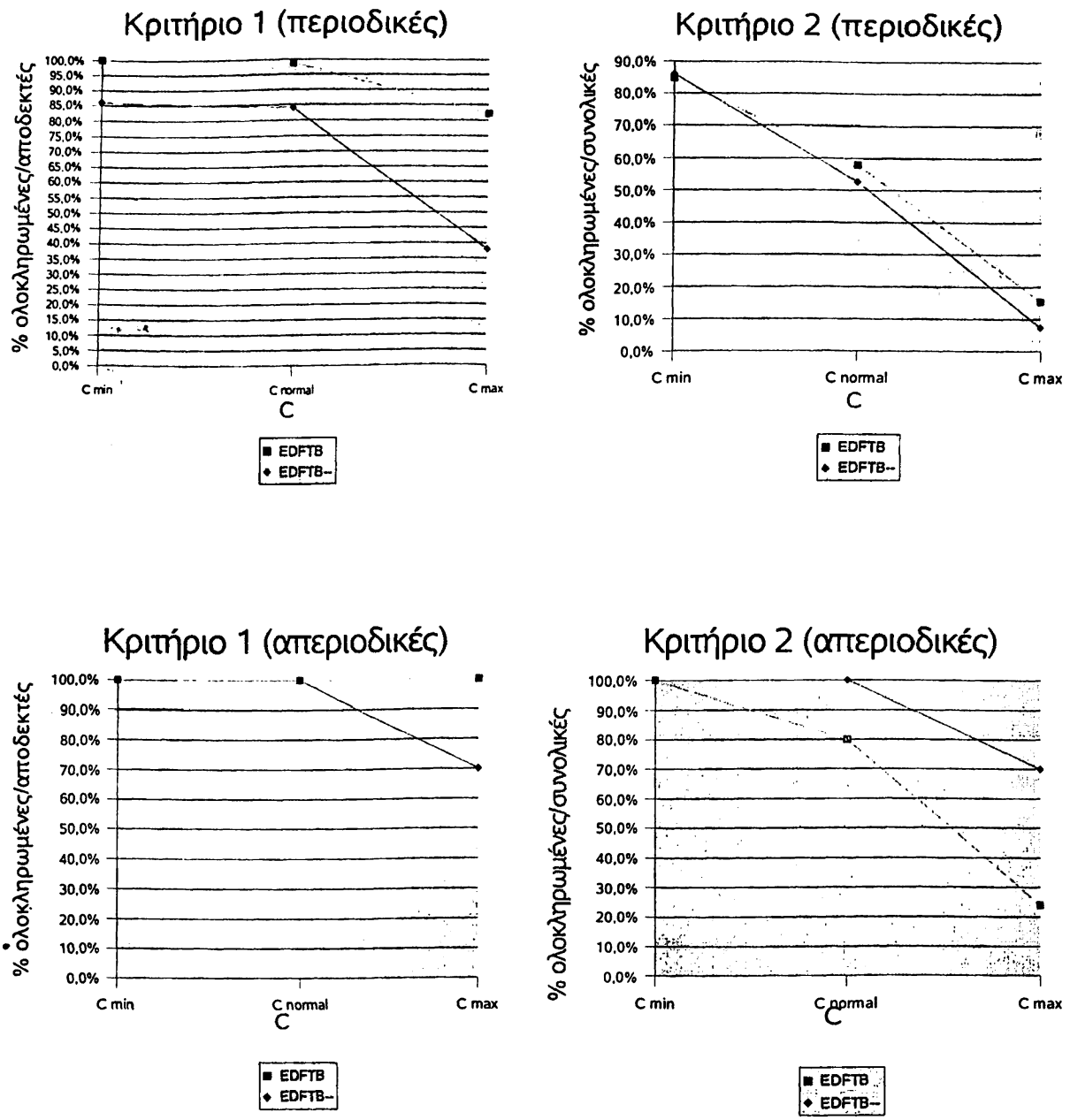
Στις μετρήσεις έγινε σύγκριση ανάμεσα στην πολιτική EDFTB, με και χωρίς τις συνθήκες που προτείναμε. Συνολικά δημιουργήθηκαν 200 αιτήσεις εκ των οποίων οι 150 (το 75%) ήταν περιοδικές και οι υπόλοιπες 50, απεριοδικές.

6.2.1.2. Μετρήσεις με Έναν Πελάτη και Δύο Εξυπηρετητές

Σε αυτήν την περίπτωση οι μετρήσεις πραγματοποιήθηκαν με τις ίδιες παραμέτρους όπως και πριν: η διάρκεια ζωής του πελάτη είναι 300.000 ms, των εξυπηρετητών 100.000 ms και ο ρυθμός δημιουργίας των αιτήσεων, από τον αποτιμητή του πελάτη προς τον κάθε εξυπηρετητή, είναι 1000 ms. Ο κάθε ένας από τους δύο εξυπηρετητές, παρέχει 3 περιοδικές υπηρεσίες και 1 απεριοδική. Το 75% των αιτήσεων του αποτιμητή αφορούν περιοδικές υπηρεσίες των εξυπηρετητών και το 25% απεριοδικές. Οι πειραματικές μετρήσεις πραγματοποιήθηκαν χρησιμοποιώντας τρία διαφορετικά σύνολα χρόνων εκτέλεσης για τις υπηρεσίες, τα C_{min}, C_{normal} και C_{max}. Όσον αφορά τις περιοδικές υπηρεσίες, στο σύνολο C_{min} οι χρόνοι εκτέλεσης είναι μικροί (της τάξης των 50 ms), στο C_{normal} είναι της τάξης των 150 ms και στο C_{max} είναι της τάξης των 450 ms (δηλ. σε κάθε σύνολο οι χρόνοι εκτέλεσης τριπλασιάζονται). Η περίοδος για την πρώτη υπηρεσία είναι 2800 ms, για την δεύτερη 1400 ms και για την τρίτη 700 ms. Ο αριθμός εκτελέσεων των περιόδων της κάθε περιοδικής υπηρεσίας είναι 10. Συνεπώς, η κάθε υπηρεσία αποφέρει διαφορετικό φόρτο στον εξυπηρετητή.

Αντίστοιχα, όσον αφορά την απεριοδική υπηρεσία, στο σύνολο C_{min} ο χρόνος εκτέλεσης είναι της τάξης των 400 ms, στο C_{normal} είναι της τάξης των 1200 ms και στο C_{max} είναι της τάξης των 3600 ms (δηλ. όπως και πριν, σε κάθε σύνολο ο χρόνος εκτέλεσης τριπλασιάζεται).

Στις μετρήσεις έγινε σύγκριση ανάμεσα στην πολιτική EDFTB, με και χωρίς τις συνθήκες που προτείναμε. Συνολικά, δημιουργήθηκαν 200 αιτήσεις εκ των οποίων οι 150 (το 75%) ήταν περιοδικές και οι υπόλοιπες 50, απεριοδικές. Οι αιτήσεις μοιράστηκαν στους δύο εξυπηρετητές.



Σχήμα 6.7 Δεύτερη Κατηγορία Μετρήσεων.

Στην πρώτη κατηγορία των μετρήσεων, όσον αφορά το πρώτο κριτήριο, παρατηρούμε τα εξής: όσον αφορά την πολιτική EDFTB, καθώς αυξάνεται ο χρόνος εκτέλεσης η επίδοση στις περιοδικές εργασίες ξεκινά από το 100% και μειώνεται σε μικρό βαθμό φτάνοντας στο 84,6%. Η επίδοση στις απεριοδικές εργασίες παραμένει στο 100%. Παρατηρούμε ότι η επιβάρυνση της ουράς του χρονοπρογραμματιστή επηρεάζει την επίδοση των περιοδικών εργασιών. Κάτι τέτοιο είναι λογικό μιας και αυξάνεται ο χρόνος εκτέλεσης των εργασιών. Συνεπώς, αυξάνεται και ο λόγος της μέγιστης χρονικής επιβάρυνσης της ουράς προς τον μικρότε-

ρο χρόνο εκτέλεσης C ανάμεσα σε όλες τις εργασίες. Επιπρόσθετα, οι περιοδικές εργασίες εκτελούνται για *n* περιόδους και έχουν συντομότερες προθεσμίες σε σχέση με τις απεριοδικές εργασίες. Από την άλλη πλευρά, στην πολιτική EDFTB-- η επίδοση του πρώτου κριτηρίου στις περιοδικές εργασίες σημειώνει ραγδαία μείωση, ξεκινώντας από το 83% και φτάνοντας στο 7,7%. Αντίστοιχα, μεγάλη μείωση παρατηρείται και στην επίδοση των απεριοδικών εργασιών. Όσον αφορά το δεύτερο κριτήριο, οι επιδόσεις των δύο πολιτικών είναι σχεδόν όμοιες. Στην περίπτωση των περιοδικών εργασιών προηγείται ελαφρώς η πολιτική EDFTB, ενώ στην περίπτωση των απεριοδικών εργασιών προηγείται η πολιτική EDFTB--.

Τα αποτελέσματα των μετρήσεων στην δεύτερη κατηγορία, είναι παρόμοια με αυτά της πρώτης. Αυτό που παρατηρείται, είναι η αύξηση των επιδόσεων των πολιτικών και στα δύο κριτήρια. Κάτι τέτοιο είναι λογικό εφόσον ο αριθμός των αιτήσεων παραμένει ο ίδιος, σε σχέση με την πρώτη κατηγορία μετρήσεων, αλλά πλέον έχουμε διαθέσιμους δύο εξυπηρετητές.

Αυτό που αξίζει να παρατηρήσουμε, σε σχέση με τις πειραματικές μετρήσεις της Ενότητας 5.4 του Κεφαλαίου 5, είναι ότι η πολιτική EDFTB έχει καλύτερες επιδόσεις στο δεύτερο κριτήριο σε σχέση με την πολιτική EDFTB--. Κάτι τέτοιο δεν ισχύει στην περίπτωση των πειραματικών μετρήσεων σε περιβάλλον προσομοίωσης. Αυτή η συμπεριφορά εξηγείται λόγω του ότι στις δύο ανωτέρω κατηγορίες μετρήσεων η διάρκεια ζωής του εξυπηρετητή ήταν πάντα μικρότερη από του πελάτη. Συνεπώς, η συνθήκη

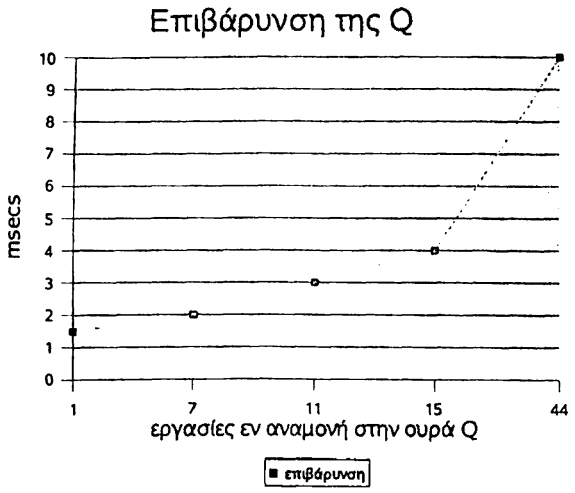
$$m_{server}Lifetime \leq m_{client}Lifetime - c_{rep}$$

ήταν πάντα αληθής, με αποτέλεσμα να γίνονται αποδεκτές περισσότερες αιτήσεις. Αυτό είχε ως συνέπεια την αύξηση της επίδοσης του δεύτερου κριτηρίου.

6.2.1.3. Μέτρηση της Επιβάρυνσης της Ουράς του Χρονοπρογραμματιστή

Στο Σχήμα 6.8 απεικονίζεται η χρονική επιβάρυνση που αποφέρει η ουρά του χρονοπρογραμματιστή. Για την μέτρηση της επιβάρυνσης της ουράς, χρησιμοποιήσαμε τις ίδιες παραμέτρους με την μέτρηση στην περίπτωση ενός πελάτη και ενός εξυπηρετητή. Η μέτρηση έγινε αυξάνοντας τον αριθμό των εν αναμονή εργασιών που βρίσκονταν στην ουρά Q του χρονοπρογραμματιστή και παρατηρώντας τον χρόνο που απαιτούνταν για να γίνει η επεξεργασία των εργασιών στην ουρά. Όπως έχει αναφερθεί, σύμφωνα με τον αλγόριθμο EDF, θα πρέπει κατά την άφιξη μιας εργασίας στον χρονοπρογραμματιστή να υπολογίζονται οι τρέχουσες προθεσμίες όλων των εν αναμονή εργασιών που βρίσκονται στην ουρά Q. Η προσπέλαση ολόκληρης της ουράς προκαλεί χρονική επιβάρυνση στον χρονοπρογραμματιστή, η οποία, όπως φαίνεται και στο Σχήμα 6.8, είναι ανάλογη με το μήκος της ουράς. Αν ο λόγος της μέγι-

στης χρονικής επιβάρυνσης προς τον μικρότερο χρόνο εκτέλεσης C ανάμεσα σε όλες τις εργασίες, είναι μεγαλύτερος από ένα όριο, τότε η επιπλέον επιβάρυνση μπορεί να προκαλέσει σε κάποιες εργασίες να χάσουν την προθεσμία τους. Το όριο αυτό εξαρτάται κάθε φορά από την εφαρμογή που χρησιμοποιεί τον χρονοπρογραμματιστή και έτσι καθορίζεται από τον προγραμματιστή της εφαρμογής. Συνεπώς, θα πρέπει να ορίζεται μια μέγιστη τιμή για το μέγεθος της ουράς, τέτοια ώστε η επιπλέον επιβάρυνση να είναι σχεδόν αμελητέα σε σχέση με τους χρόνους εκτέλεσης των εργασιών.



Σχήμα 6.8 Επιβάρυνση της Ουράς.

6.2.1.4. Μέτρηση των Απαιτήσεων σε Μνήμη του Εξυπηρετητή

Για την μέτρηση των απαιτήσεων σε μνήμη του εξυπηρετητή, χρησιμοποιήσαμε τις ίδιες παραμέτρους με την μέτρηση στην περίπτωση ενός πελάτη και ενός εξυπηρετητή. Η απαιτούμενη μνήμη υπολογίστηκε ως η διαφορά της ελεύθερης μνήμης κατά την εκκίνηση του εξυπηρετητή και της μικρότερης τιμής της ελεύθερης μνήμης που μετρήθηκε κατά τη διάρκεια της εκτέλεσης του. Ο κάθε εξυπηρετητής εμπεριέχει και έναν χρονοπρογραμματιστή. Συνεπώς, ο εξυπηρετητής μαζί με τον χρονοπρογραμματιστή εκτελούνται στην ίδια εικονική μηχανή της Java. Η μέτρηση της ελεύθερης μνήμης πραγματοποιείται χρησιμοποιώντας την μέθοδο *freeMemory()* της κλάσης *java.lang.Runtime*. Η μέθοδος αυτή επιστρέφει την διαθέσιμη ελεύθερη μνήμη, σε bytes, στην εικονική μηχανή της Java. Η διαθέσιμη ελεύθερη μνήμη κατά την εκκίνηση του εξυπηρετητή ήταν 3,99 MB και στο τέλος της εκτέλεσης ήταν 2,89 MB. Συνεπώς, η απαιτούμενη μνήμη από τον εξυπηρετή που υλοποιήθηκε, είναι 1,1 MB. Αυτή η ποσότητα μνήμης μπορεί παλαιότερα να ήταν υπερβολική για τις δυνατότητες μιας φορητής

υπολογιστικής συσκευής και έτσι να καθιστούσε απαγορευτική την εκτέλεση των πολιτικών που προτείνουμε σε αυτήν την εργασία. Αν αναλογιστούμε την τρέχουσα κατάσταση της τεχνολογίας των φορητών συσκευών, η συγκεκριμένη και πολλές φορές περισσότερη ποσότητα μνήμης θεωρείται πλέον δεδομένη στην πλειοψηφία των συσκευών. Συνεπώς, λόγω του γεγονότος ότι μια συσκευή εκτελεί μόνο έναν εξυπηρετητή, οι απαιτήσεις σε μνήμη δεν είναι απαγορευτικές για την υλοποίηση των πολιτικών σε μία φορητή υπολογιστική συσκευή.

ΚΕΦΑΛΑΙΟ 7. ΣΥΜΠΕΡΑΣΜΑΤΑ

Σε αυτήν την εργασία, προτείναμε μια υπηρεσία ενδιάμεσου λογισμικού που παρέχει την εκτέλεση υπηρεσιών πραγματικού χρόνου σε κρίσιμα περιβάλλοντα κινητού υπολογισμού. Η αρχιτεκτονική της υπηρεσίας βασίζεται σε κινητές οντότητες και υποστηρίζει τέσσερις διαφορετικές πολιτικές για την εξυπηρέτηση των αιτήσεων των οντοτήτων-πελάτη που αποστέλλονται στις οντότητες-εξυπηρετητή. Η πρώτη πολιτική λαμβάνει υπόψη της τις διάρκειες ζωής του πελάτη και του εξυπηρετητή και εγγυάται ότι θα αποσταλεί μια απάντηση πίσω στον πελάτη, εφόσον όμως ο εξυπηρετητής καταφέρει να εξυπηρετήσει την αίτηση του πελάτη. Η δεύτερη και η τρίτη πολιτική παρέχουν ισχυρότερες εγγυήσεις για την εκπλήρωση της επιλογής του εξυπηρετητή που μπορεί να εξυπηρετήσει την αίτηση μέσα στην διάρκεια ζωής εξίσου του πελάτη και του εξυπηρετητή. Αυτό πραγματοποιείται λαμβάνοντας υπόψη και το φόρτο των διαθέσιμων εξυπηρετητών που παρέχουν τις υπηρεσίες. Η τέταρτη πολιτική χειρίζεται επιπρόσθετα και αιτήσεις για υπηρεσίες που εκτελούνται περιοδικά. Για τον χειρισμό υπηρεσιών που εκτελούνται περιοδικά, επεκτείνουμε τους αλγορίθμους χρονοπρογραμματισμού EDF και TB, οι οποίοι εφαρμόζονται σε συστήματα πραγματικού χρόνου. Η επέκταση αυτή γίνεται για την κάλυψη των αναγκών που προκύπτουν σε ένα περιβάλλον κινητών οντοτήτων. Έως σήμερα, υπήρχαν διάφορες προσεγγίσεις για τον χειρισμό διαφόρων *ιδιοτήτων αξιοπιστίας (dependability attributes)* [25]. Αυτές οι προσεγγίσεις συνήθως επικεντρώνονται στην απόδοση, τη διαθεσιμότητα, την ασφάλεια, τη φήμη, κτλ. Η προσέγγιση που προτείνουμε σε αυτήν την εργασία είναι η πρώτη προσπάθεια που επικεντρώνεται σε χρονικούς περιορισμούς και διάρκειες ζωής των οντοτήτων. Στη συνέχεια, θα αναφερθούμε σε επιπλέον προσθήκες που είναι συμπληρωματικές και επεκτείνουν την υπηρεσία που προτείναμε.

Η προτεινόμενη υπηρεσία χειρίζεται την εκτέλεση ανεξάρτητων μεταξύ τους αιτήσεων και εργασιών. Μια ενδιαφέρουσα επέκταση θα ήταν η υποστήριξη για την εκτέλεση *ροής εργασιών (workflows)*. Ο υπολογισμός της διάρκειας ζωής των κινητών οντοτήτων είναι σημαντικός σύμφωνα με την προσέγγιση μας. Επι του παρόντος, υπάρχουν χρήσιμες τεχνικές που σαν στόχο έχουν τον υπολογισμό της φυσικής κίνησης των κινητών οντοτήτων [12] και τεχνικές που είναι ανεξάρτητες της κίνησης και υπολογίζουν την διαθεσιμότητα των κινητών οντοτήτων [24].

Όσον αφορά τον χρονοπρογραμματισμό εργασιών πραγματικού χρόνου, έχουν προταθεί στο παρελθόν πολλοί κλασικοί αλγόριθμοι. Ανάμεσα στους πιο γνωστούς είναι ο αλγόριθμος EDF, τον οποίο χρησιμοποιούμε στην παρούσα εργασία. Ωστόσο, πολλοί άλλοι αλγόριθμοι όπως ο MLF (minimum laxity first) και ο MUF (maximum urgency first) [20], μπορεί να αποβούν χρήσιμοι σε κρίσιμα περιβάλλοντα κινητού υπολογισμού. Όπως και στην περίπτωση του EDF, αυτοί οι αλγόριθμοι θα πρέπει πρώτα να ενισχυθούν κατάλληλα έτσι ώστε να χρησιμοποιηθούν σε τέτοια περιβάλλοντα. Στην περίπτωση μας, θα μπορούσαμε να χρησιμοποιήσουμε τον αλγόριθμο MUF αντί του EDF, για να χειριστούμε καταστάσεις όπου κανένας από τους διαθέσιμους εξυπηρετητές δεν μπορεί να εξυπηρετήσει μια αίτηση ενός πελάτη. Ο αλγόριθμος MUF συσχετίζει τις εργασίες με έναν παράγοντα *σπουδαιότητας (importance)*, κάτι το οποίο μπορεί να χρησιμεύσει ως κριτήριο για την απόρριψη εργασιών εν αναμονή, με σκοπό να εξυπηρετηθούν εργασίες μεγαλύτερης σημασίας.

Επίσης, μια ενδιαφέρουσα επέκταση στον χρονοπρογραμματιστή του κάθε εξυπηρετητή θα ήταν η δυναμική εναλλαγή της πολιτικής χρονοπρογραμματισμού ανάλογα με κάποια κριτήρια όπως για παράδειγμα ο φόρτος του εξυπηρετητή και η διαθέσιμη διάρκεια ζωής του. Η εναλλαγή αυτή θα πρέπει να βασίζεται σε κάποια προκαθορισμένα κριτήρια τα οποία θα βασίζονται στην συμπεριφορά της κάθε πολιτικής χρονοπρογραμματισμού. Ωστόσο, τόσο ο προσδιορισμός αυτών των κριτηρίων, όσο και η υλοποίηση της δυναμικής αυτής εναλλαγής φαίνεται να μην είναι απλή και τετριμμένη διαδικασία.

Σχετικά με την επιλογή του κατάλληλου εξυπηρετητή από τον αποτιμητή, για την αποστολή των αιτήσεων του πελάτη, θα μπορούσαμε να ακολουθήσουμε και μια εναλλακτική προσέγγιση βασιζόμενοι στις ακόλουθες παρατηρήσεις. Σε ένα ασύρματο δίκτυο, όταν ένας αποτιμητής αποστέλλει μια αίτηση σε έναν εξυπηρετητή, τότε είναι πολύ πιθανό όλοι οι εξυπηρετητές να "ακούσουν" αυτήν την αίτηση. Συνεπώς, όλοι θα μπορούσαν να απαντήσουν εάν μπορούν ή όχι να εξυπηρετήσουν την συγκεκριμένη αίτηση. Έτσι ο αποτιμητής θα μπορούσε να επιλέξει τον καταλληλότερο εξυπηρετητή βάσει ενός κριτηρίου όπως π.χ ο τρέχων φόρτος. Κάτι τέτοιο θα γινόταν εύκολα υλοποιήσιμο εάν ο κάθε εξυπηρετητής με κάθε μήνυμα που έστελνε, συμπεριλάμβανε μέσα και κάποιες επιπλέον πληροφορίες όπως είναι ο τρέχων φόρτος. Η προσθήκη μιας τέτοιας πληροφορίας, όπως είναι ο τρέχων φόρτος, θα προκαλούσε αμελητέα αύξηση στο μέγεθος του κάθε μηνύματος και έτσι δεν θα προκαλούσε επιβάρυνση στο δίκτυο. Ωστόσο, το πρόβλημα που προκύπτει με αυτήν την προσέγγιση είναι ότι σε κάθε αίτηση ενός αποτιμητή, θα πρέπει να απαντάνε όλοι οι εξυπηρετητές. Κάτι τέτοιο αυξάνει το πλήθος των μηνυμάτων που αποστέλλουν οι εξυπηρετητές με αποτέλεσμα να μειώνεται γρηγορότερα η μπαταρία τους, υποθέτοντας ότι χρησιμοποιούν μπαταρία, και παράλληλα να αυ-

ξάνεται και η κίνηση στο δίκτυο.

[2] [illegible]

[3] [illegible]

[4] [illegible]

[5] [illegible]

[6] J.P. [illegible] Algorithms, Real-Time Systems Symposium, [illegible]

[7] J.P. [illegible] Algorithms, Real-Time Systems Symposium, [illegible]

[8] J.P. [illegible] Real-Time [illegible] 1989.

ΑΝΑΦΟΡΕΣ

[1] N. Audsley and A. Burns. "Real-time System Scheduling", Predictably Dependable Computer Systems, Vol. 2, Ch. 2, Part II.

[2] H. Chetto and M. Chetto. "Some Results of the Earliest Deadline Scheduling Algorithm", IEEE Transactions on Software Engineering, 15(10), pp. 1261-1269, 1989.

[3] S. Corson and J. Macker. "Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations", RFC 2501, January 1999.

[4] IEEE. "IEEE Standard for Wireless LAN Medium Access Control (MAC)", Technical Report, IEEE, 1999.

[5] V. Issarny, D. Sacchetti, F. Tartanoglu, F. Sailhan, R. Chibout, N. Levy and A. Talamona. "Developing Ambient Intelligence Systems: A Solution based on Web Services", Automated Software Engineering, v.12 n.1, pp.101-137, January 2005.

[6] J.P Lehoczky and S. Ramos-Thuel. "An Optimal Algorithm for Scheduling Soft Aperiodic Tasks in Fixed Priority Preemptive Systems", Proceedings of Real-Time Systems Symposium, pp. 110-123, 1992.

[7] J.P Lehoczky, L. Sha, and Y. Ding. "The Rate Monotonic Scheduling Algorithm, Exact Characterization and Average Case Behavior", Proceedings of the IEEE Real-Time System Symposium, pp. 166-171, 1989.

[8] J.P Lehoczky, L. Sha and J.K. Strosnider. "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", Proceedings of Real-Time Systems Symposium, pp. 166-171, 1989.

[9] C.L. Liu and J.W Layland. "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", JACM 20(1), pp. 46-61, 1973.

[10] J.W.S Liu. "Real-Time Systems", Prentice Hall, 2000.

[11] J. Manner and M. Kojo. "Mobility Related Terminology", RFC 3753, June 2004.

[12] H. M. O. Mokhtar and J. Su. "Universal Trajectory Queries for Moving Object Databases", Proceedings of the IEEE International Conference on Mobile Data Management (MDM'04), pp 133–146, 2004.

[13] J. Neilson. "PARASOL User's Manual (v 3.1)", Technical Report, School of Computer Science-Carleton University-Ottawa, K1S5B6.

[14] F. Papadopoulos, A. Zarras, P. Vassiliadis and E. Pitoura. "Timely Provisioning of Mobile Services in Critical Pervasive Environments", Proceedings of the 7th International Symposium on Distributed Objects and Applications (DOA'05), 2005.

[15] "Real-Time Specification for Java (v 1.0.1)", http://www.rtsj.org/, 2005.

[16] M. Satyanarayanan. "Pervasive Computing: Vision and Challenges", IEEE Personal Communications, 2001.

[17] B. Sprunt, L. Sha and J. Lehoczky. "Aperiodic Task Scheduling for Hard Real-Time Systems", The Journal of Real-Time Systems 1, pp. 27-60, 1989.

[18] M. Spuri, G. Buttazzo and F. Sensini. "Robust Aperiodic Scheduling under Dynamic Priority Systems", Proceedings of the 16th IEEE Real Time Systems Symposium, pp. 210–221, 1995.

[19] M. Spuri and G. Buttazzo. "Scheduling Aperiodic Tasks in Dynamic Priority Systems", Real-Time Systems, vol. 10, pp. 179-210, 1996.

[20] D. B. Stewart and P. K. Khosla. "Real-Time Scheduling of Sensor-Based Control Systems", Proceedings of the 8th IEEE International Workshop on Real-Time Operating Systems and Software (RTOSS'91), 1991.

[21] Sun Microsystems. "JAVA Micro Edition", http://java.sun.com/products/j2me/, 2001.

[22] Sun Microsystems. "Java Platform, Standard Edition, API Specification (v 1.4.2)", 2003.

[23] TimeSys Corporation. "Java Reference Implementation (RI) and Technology Compatibility Kit (TCK) for RTSJ", http://www.timesys.com/java/, 2005.

[24] Y. Xiong, X. Lin and J. Rowson. "Estimating Device Availability in Pervasive Peer-to-Peer Environment", Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'04), 2004.

[25] L. Zeng, B. Benatallah and M. Dumas. "Quality Driven Web Services Composition", Proceedings of the 12th ACM International Conference on the World Wide Web (WWW'03), pp 411-421, 2003

ΣΥΝΤΟΜΟ ΒΙΟΓΡΑΦΙΚΟ

Ο Φίλιππος Παπαδόπουλος γεννήθηκε το 1979 στην πόλη της Βέροιας. Εισέλθε στο τμήμα πληροφορικής του πανεπιστημίου Ιωαννίνων το 1999. Αποφοίτησε από το τμήμα το 2003 με βαθμό πτυχίου 6,96 'ΛΙΑΝ ΚΑΛΩΣ'. Η πτυχιακή του εργασία έχει τίτλο "Τοποθέτηση και Απόδοση 3Δ Αντικειμένων βάσει Γεωμετρικών Περιορισμών" και εκπονήθηκε υπό την επίβλεψη του καθηγητή κ. Ιωάννη Φούντου. Το 2003 έγινε δεκτός από το τμήμα πληροφορικής του πανεπιστημίου Ιωαννίνων ως μεταπτυχιακός φοιτητής. Κατά τη διάρκεια των ετών 2004 και 2005 δίδαξε τα μαθήματα "Λειτουργικό σύστημα UNIX" (Φθινοπωρινό εξάμηνο 2004) και "Διαθεματική εργασία" (Εαρινό εξάμηνο 2005), στο ΙΕΚ Ιωαννίνων. Στα ενδιαφέροντά του συμπεριλαμβάνονται η Τεχνολογία Λογισμικού και τα Κατανεμημένα Συστήματα.

ΔΗΜΟΣΙΕΥΣΕΙΣ ΣΥΓΓΡΑΦΕΑ

F. Papadopoulos, A. Zarras, P. Vassiliadis and E. Pitoura. "Timely Provisioning of Mobile Services in Critical Pervasive Environments", Proceedings of the 7th International Symposium on Distributed Objects and Applications (DOA'05), 2005.