ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΥΛΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΩΝ ΓΙΑ ΤΟ INTERVAL SCHEDULING ΠΡΟΒΛΗΜΑ ΣΕ EDGE COMPUTING ΠΕΡΙΒΑΛΛΟΝ

ΕΥΣΤΡΑΤΙΟΣ ΑΡΧΟΝΤΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

ΝΙΚΟΛΑΟΣ ΤΖΙΡΙΤΑΣ

ΑΝΑΠΛΗΡΩΤΗΣ ΚΑΘΗΓΗΤΗΣ

Λαμία έτος 2024-2025

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

DEVELOPMENT OF ALGORITHMS FOR THE INTERVAL SCHEDULING PROBLEM IN AN EDGE COMPUTING ENVIRONMENT

EFSTRATIOS ARCHONTIS

FINAL THESIS

ADVISOR

NIKOLAOS TZIRITAS

ASSOCIATE PROFESSOR

Lamia year 2024-2025

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί ***χωρίς να τα περικλείω σε εισαγωγικά*** *και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

2. Δέχομαι ότι η αυτολεξεί ***παράθεση χωρίς εισαγωγικά****, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια

4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: 07/01/2025

Ο – Η Δηλ.

(1) «Οποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση
του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων
σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

ΠΕΡΙΛΗΨΗ

Το Interval Scheduling αποτελούσε ανέκαθεν βασικό κλάδο της επιστήμης των υπολογιστών, διευκολύνοντας την αποτελεσματική εκτέλεση εργασιών υψηλής έντασης με την κατανομή τους σε πολλές μονάδες επεξεργασίας. Με την πάροδο του χρόνου, έχουν δημιουργηθεί διάφορες μέθοδοι χρονοπρογραμματισμού (scheduling) για την βελτιστοποίηση της χρήσης των πόρων και την ελαχιστοποίηση της διάρκειας εκτέλεσης (makespan). Οι συμβατικές στατικές μέθοδοι χρονοπρογραμματισμού συχνά δυσκολεύονται να προσαρμοστούν στα μεταβαλλόμενα χαρακτηριστικά που συχνά συνοδεύουν περίπλοκα και υψηλά φορτία εργασίας, με αποτέλεσμα την αναποτελεσματικότητα σε πολυπύρηνα συστήματα.

Η παρούσα εργασία διερευνά τον διαμορφώσιμο χρονοπρογραμματισμό (moldable scheduling), μια τεχνική που επιτρέπει την τροποποίηση των κατανομών εργασιών πριν από την εκτέλεση-ταξινόμηση τους, διατηρώντας την ισορροπία μεταξύ ευελιξίας και αποτελεσματικότητας του χρονοπρογραμματισμού. Ο κύριος στόχος αυτής της έρευνας είναι η αξιολόγηση της απόδοσης του moldable scheduling σε σχέση με τους παραδοσιακούς αλγορίθμους, ειδικά σε περιπτώσεις που διαβάζουμε περίπλοκα workload. Για να το πετύχουμε αυτό, αναπτύξαμε και αναλύσαμε δύο διαφορετικές προσεγγίσεις που βασίζονται σε όρια (thresholds), εξετάζοντας πώς η δυναμική και η στατική οριοθέτηση επηρεάζουν το makespan υπό διαφορετικά επίπεδα διαθεσιμότητας πυρήνων και μεταβλητότητας του φόρτου εργασίας.

Τρέχοντας αναλυτικά πειράματα με ποικιλία από workloads και αριθμό διαθέσιμων πυρήνων, αναλύουμε τις επιπτώσεις των φόρτων εργασίας χαμηλής και υψηλής διακύμανσης, συγκρίνοντας τον moldable scheduling αλγόριθμο μας, με τις παραδοσιακές μεθόδους First-Fit (FF) και Moldable First-Fit (MOLDFF). Τα αποτελέσματα αποδίδουν γνώσεις σχετικά με τον τρόπο με τον οποίο η μορφοποίηση των διεργασιών επηρεάζει την απόδοση, ιδίως σε περιβάλλοντα υψηλής διακύμανσης, όπου οι στατικές μέθοδοι χρονοπρογραμματισμού συνήθως δυσκολεύονται.

Με την εξέταση των χρόνων εκτέλεσης σε διάφορους αριθμούς πυρήνων και επίπεδα φόρτου εργασίας, η επικείμενη πτυχιακή προσφέρει ευρύτερη κατανόηση του αποτελεσματικού προγραμματισμού εργασιών σε περιβάλλοντα παράλληλων υπολογιστών, προτείνοντας πιθανές βελτιώσεις για τα επερχόμενα πλαίσια προγραμματισμού.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

Ευχαριστίες

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες σε όλους όσους συνέβαλαν στην ολοκλήρωση της παρούσας πτυχιακής εργασίας. Ιδιαίτερα απευθύνομαι στον επιβλέποντα καθηγητή μου κύριο Νικόλαο Τζιρίτα και τον υποψήφιο διδάκτορα Γιώργο Διαμαντόπουλο, για την υποστήριξη και τις πολύτιμες συμβουλές που συνείσφεραν στην ανάπτυξη της εργασίας μου. Επιπλέον ευχαριστώ τους γονείς, οικογένεια και φίλους μου για την ενθάρρυνση τους κατά την διάρκεια της ακαδημαϊκής μου εκπαίδευσης.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

Πίνακας περιεχομένων

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

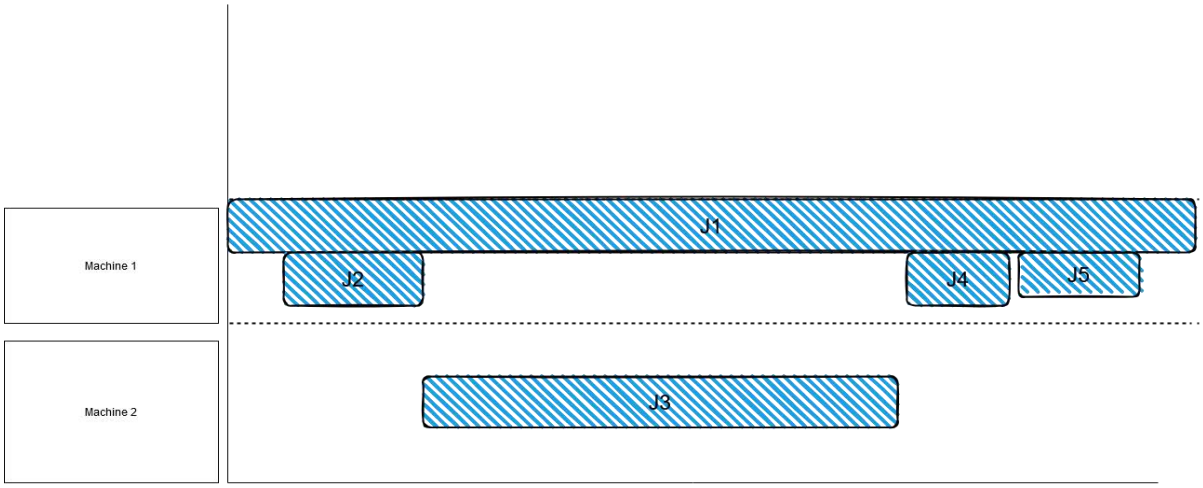
Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

ΚΕΦΑΛΑΙΟ 1 ΕΙΣΑΓΩΓΗ

1.1 INTERVAL SCHEDULING

Στο πεδίο του data scheduling, προκύπτουν αρκετές προκλήσεις. Για παράδειγμα, όταν μεταδίδεται ένα έκτακτο ενημερωτικό δελτίο, όλα τα δεδομένα από κάμερες και μικρόφωνα πρέπει να αποσταλούν στο cloud και να κωδικοποιηθούν, οπού μετά θα προβληθούν σε χρήστες ανάλογα τις απαιτήσεις τους. Η υπολογιστική ισχύς που απαιτείται για την κωδικοποίηση σε πραγματικό χρόνο είναι προβλέψιμη, πράγμα που επιτρέπει την προ-δέσμευση των κατάλληλων πόρων. Παρόμοιες απαιτήσεις συναντώνται και σε άλλες εφαρμογές, όπως το cloud gaming και η ανάλυση δεδομένων σε πραγματικό χρόνο. Τα περισσότερα συνήθως μπορούμε να τα ταξινομήσουμε σε διάφορες υποκατηγορίες και υπό-προβλήματα. Αυτά είναι κεντρικά σε διάφορες εφαρμογές σε διαφορετικούς τομείς, όπως η επιστήμη των υπολογιστών, η επιχειρησιακή έρευνα και η μηχανική. Ένα από αυτά είναι το λεγόμενο interval scheduling πρόβλημα, από το οποίο μπορούμε να αναθέσουμε “στόχους" για την μεγιστοποίηση ή ελαχιστοποίηση τους.

Το Intervel Scheduling είναι ένα θεμελιώδες πρόβλημα στην συνδυαστική βελτιστοποίηση που περιλαμβάνει την επιλογή ενός μέγιστου υποσυνόλου μη επικαλυπτόμενων διαστημάτων από ένα δεδομένο σύνολο διαστημάτων. Στο Interval Scheduling πρόβλημα, διεργασίες πράξεις ή δεδομένα (αναφερόμενα από εδώ και πέρα ως jobs/tasks ή απλά διεργασίες), έχουν συγκεκριμένα χρονικά διαστήματα για την έναρξη και λήξη τους, γνωστά ως job intervals, και πρέπει να ταξινομηθούν σε κόμβους επεξεργασίας (servers, virtual machines, cpu cores).



Παράδειγμα scheduled tasks. Πριν την βελτιστοποίηση

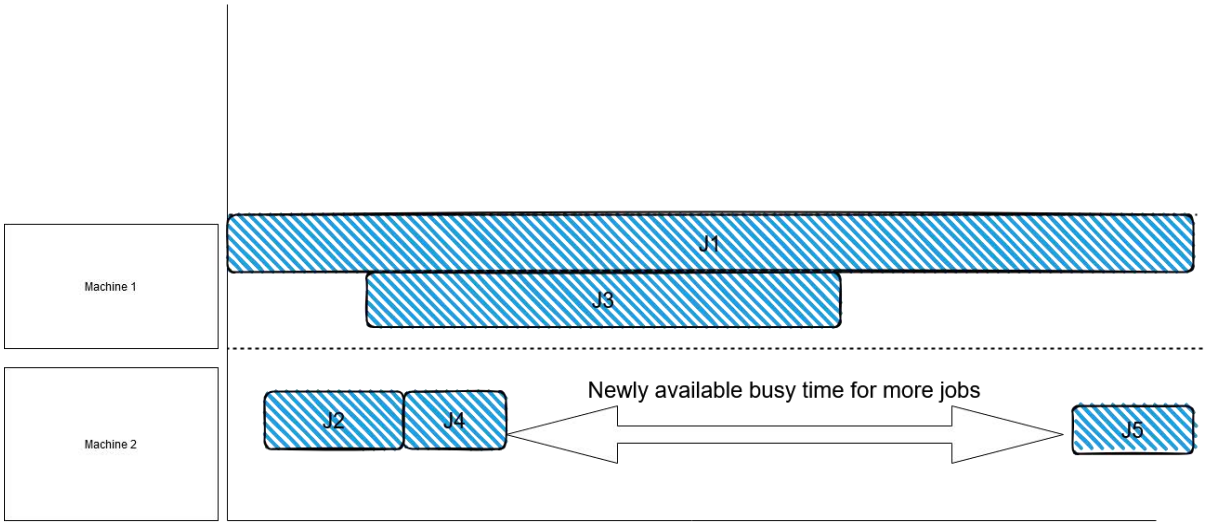
Στόχοι μας στο interval scheduling πρόβλημα είναι:

1. Κάθε εργασία που ανατίθεται σε έναν Server θα πρέπει να εκτελείται αποκλειστικά σε αυτόν καθ’ όλη τη διάρκεια της. Αυτό σημαίνει ότι από τη στιγμή που μια εργασία ξεκινά την εκτέλεσή της σε έναν συγκεκριμένο Server, δεν επιτρέπεται να μετακινηθεί σε κάποιον άλλο Server για την ολοκλήρωσή της. Αυτή η προϋπόθεση είναι σημαντική γιατί εξασφαλίζει τη σταθερότητα και την προβλεψιμότητα της εκτέλεσης των εργασιών, αποφεύγοντας επιπλέον κόστη ή

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

καθυστερήσεις που μπορεί να προκύψουν από την ανακατανομή εργασιών μεταξύ των servers. Επιπλέον, αυτή η προσέγγιση μειώνει την πολυπλοκότητα της διαχείρισης, καθώς οι servers δεν χρειάζεται να μοιράζονται ενδιάμεσα αποτελέσματα ή να συγχρονίζονται μεταξύ τους για την ολοκλήρωση μιας εργασίας.

2. Κάθε εργασία που προγραμματίζεται σε έναν Server απαιτεί συγκεκριμένους πόρους για την εκτέλεσή της. Ένας από τους βασικούς περιορισμούς είναι ότι το σύνολο των πόρων που απαιτούν οι ανατεθειμένες εργασίες σε έναν Server δεν πρέπει να ξεπερνά τους συνολικούς διαθέσιμους πόρους του Server αυτού. Ο περιορισμός αυτός είναι κρίσιμος για την αποφυγή υπερφόρτωσης των servers, που μπορεί να οδηγήσει σε μείωση της απόδοσης ή ακόμα και σε αστοχίες του συστήματος. Στόχος είναι να διασφαλιστεί ότι κάθε Server λειτουργεί εντός των δυνατοτήτων του, εξασφαλίζοντας έτσι σταθερή και αξιόπιστη απόδοση του συστήματος στο σύνολό του.
3. Η ελαχιστοποίηση του συνολικού χρόνου κατά τον οποίο οι servers είναι απασχολημένοι με την εκτέλεση εργασιών. Σκοπός είναι η επιδίωξη της μεγιστοποίησης της αποδοτικότητας των servers, ελαχιστοποιώντας ταυτόχρονα τους απαιτούμενους πόρους και το συνολικό κόστος. Η ελαχιστοποίηση αυτή επιτυγχάνεται μέσω του αποδοτικού προγραμματισμού των εργασιών. Αυτό μπορεί να περιλαμβάνει την εκτέλεση εργασιών παράλληλα (parallel execution), την κατανομή τους με τρόπο που να εξισορροπεί τον φόρτο εργασίας μεταξύ των servers, και τη βέλτιστη χρήση των διαθέσιμων πόρων για την αποφυγή χρόνων αδράνειας ή υποαπασχόλησης των servers. Επιπλέον, η ελαχιστοποίηση αυτή έχει και οικολογικές και οικονομικές προεκτάσεις, καθώς η αποδοτική χρήση των servers συνεπάγεται λιγότερη κατανάλωση ενέργειας και χαμηλότερο λειτουργικό κόστος, γεγονός που είναι σημαντικό σε μεγάλα κέντρα δεδομένων (data centers) και υποδομές υπολογιστικού νέφους (cloud computing).



Παράδειγμα scheduled tasks. Μετά την βελτιστοποίηση

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

Το Interval Scheduling διακρίνεται σε δύο κύριες κατηγορίες: offline & online scheduling. Η διαφορά αυτών των δύο μεθόδων προγραμματισμού βρίσκεται στις πληροφορίες που έχει ο scheduler πριν λάβει τις αποφάσεις του και στον τρόπο με τον οποίο αυτή η πληροφορία επηρεάζει τις διαδικασίες χρονοπρογραμματισμού.

- Στην offline εκδοχή όλες οι απαιτούμενες πληροφορίες για τα jobs, όπως η ποσότητά τους, τις χρονικές περιόδους τους καθώς και τις απαιτήσεις σε πόρους για κάθε εργασία, είναι γνωστές από την αρχή της εκτέλεσης του αλγόριθμου. Έτσι, ο προγραμματιστής έχει τη δυνατότητα να χρησιμοποιήσει αλγόριθμους βελτιστοποίησης που στοχεύουν στον καλύτερο δυνατό χρονοπρογραμματισμό. Αυτό περιλαμβάνει την εύρεση της βέλτιστης λύσης που μεγιστοποιεί την εκτέλεση των εργασιών ή ελαχιστοποιεί την κατανάλωση πόρων, με βάση τα δεδομένα που είναι γνωστά. Έτσι, η διαδικασία αυτή χαρακτηρίζεται ως στατικός προγραμματισμός.
- Αντίθετα με την offline, στην online έκδοση του προβλήματος οι πληροφορίες δεν είναι διαθέσιμες απ' την αρχή. Οι εργασίες εμφανίζονται δυναμικά, μία προς μία ή σε ομάδες, και ο χρονοπρογραμματιστής πρέπει να πάρει αποφάσεις σε πραγματικό χρόνο, χωρίς να γνωρίζει τι θα συμβεί στο μέλλον. Αυτό σημαίνει ότι η απόφαση για το αν θα εκτελεστεί μια εργασία πρέπει να ληφθεί αμέσως, με βάση τις διαθέσιμες εκείνη τη στιγμή πληροφορίες, χωρίς να υπάρχει η δυνατότητα πρόβλεψης για μελλοντικές εργασίες. (Ένα παράδειγμα είναι οι ζωντανές μεταδώσεις, όπου δεδομένα συνεχώς στέλνονται και λαμβάνονται, χωρίς να ξέρει ο αλγόριθμους τον πληθυσμό τους καθώς γίνεται εκείνη την στιγμή, αρά δεν υπάρχει γνωστός χρόνος διάρκειας, προσαρμοζόμενος στις τρέχουσες συνθήκες και λαμβάνοντας υπόψη την αβεβαιότητα των μελλοντικών γεγονότων.) Λόγω αυτού του κενού πληροφορίας, οι αλγόριθμοι του βασίζονται συχνά σε ευριστικές μεθόδους που έχουν ως στόχο τη λήψη γρήγορων αποφάσεων που επιτυγχάνουν καλή απόδοση, αν και όχι απαραίτητα βέλτιστη.

Ένας interval scheduling αλγόριθμος μπορεί να χρησιμοποιείται επίσης στην κατανομή εύρους ζώνης για δίκτυα επικοινωνίας, όπου τα πακέτα δεδομένων πρέπει να μεταδίδονται χωρίς σύγκρουση. Στην επιχειρησιακή έρευνα, χρησιμοποιείται σε προβλήματα κατανομής πόρων, όπως η ανάθεση αιθουσών συσκέψεων σε εκδηλώσεις με τρόπο που να μεγιστοποιεί τη χρήση του δωματίου.

Για την δική μας εργασία, στόχος μας είναι να έχουμε τον συνολικό χρόνο εκτέλεσης του scheduling tasks (γνωστό ως makespan) όσο πιο χαμηλά μπορούμε. Η ελαχιστοποίηση του makespan είναι ουσιαστική για τον περιορισμό του κόστους και τη μείωση της κατανάλωσης ενέργειας. Αυτό μπορεί να φανεί ιδιαίτερα κρίσιμο σε μεγάλα data centers, όπου οι απαιτήσεις απόδοσης και κατανάλωση ενέργειας είναι υψηλές. Μικρότερος χρόνος εκτέλεσης για τους servers σημαίνει ότι λειτουργούν πιο αποδοτικά και άμεσα, μειώνοντας το βάρος τους στο περιβάλλον.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

1.2 MOLDABLE TASKS

Οι διαμορφώσιμες εργασίες (mouldable tasks) αντιπροσωπεύουν μια ειδική κατηγορία εργασιών στον παράλληλο υπολογισμό, όπου ο αριθμός των επεξεργαστών που εκχωρούνται σε κάθε εργασία μπορεί να προσαρμοστεί πριν από την έναρξη της εργασίας. Σε αντίθεση με τις στατικές εργασίες (rigid) που απαιτούν σταθερό αριθμό επεξεργαστών ή τις εύπλαστες εργασίες που μπορούν να αλλάξουν δυναμικά τον αριθμό των επεξεργαστών κατά την εκτέλεση, οι διαμορφώσιμες εργασίες επιτρέπουν ευελιξία στην κατανομή πόρων διατηρώντας παράλληλα μια προκαθορισμένη δομή.

Τα Mouldable Tasks [5] είναι ιδιαίτερα σημαντικά σε περιβάλλοντα υπολογιστών υψηλής απόδοσης (HPC), όπου η αποτελεσματική χρήση των υπολογιστικών πόρων είναι κρίσιμη. Προσαρμόζοντας τον αριθμό των επεξεργαστών που εκχωρούνται σε κάθε εργασία με βάση το τρέχον φορτίο του συστήματος και τις απαιτήσεις εργασιών, με τα mouldable tasks μπορούμε να βελτιώσουμε την συνολική απόδοση του συστήματος και να μειώσει τους χρόνους εκτέλεσης.

Σε αυτήν την πτυχιακή εργασία, κάνουμε διασταύρωση του interval scheduling και των mouldable tasks, προτείνοντας έναν νέο αλγόριθμο (ονόματι MoldableSqueezy) για την αντιμετώπιση των προκλήσεων στον αποτελεσματικό προγραμματισμό διαμορφώσιμων εργασιών σε παράλληλα υπολογιστικά περιβάλλοντα. Συγκεκριμένα βλέπουμε πως μπορούμε αν αυξήσουμε ή μειώσουμε τους πυρήνες που “χρειάζεται” η εκτέλεση του ενός task για την εύρεση ενός μικρότερου makespan.

Κατά την διάρκεια του προγραμματισμού του τελικού αλγορίθμου, αναπτύξαμε μία ψευδό-δυναμική αναθέτηση ορίων για την ταξινόμηση διεργασιών σε ήδη υπάρχον διακομιστές. Στην πραγματικότητα θέτουμε ένα παράθυρο μεταξύ 1 και το 10, όπου θα δοκιμαστούν όλα τα ενδιάμεσα νούμερα ως threshold, με σκοπό την προσαρμογή του αλγόριθμου σε διαφορετικά φορτία εργασίας.

ΚΕΦΑΛΑΙΟ 2 Background

Για να κατανοήσουμε πλήρως την περιπλοκότητα του interval scheduling και του mouldable tasks, είναι απαραίτητο να κατανοήσουμε αρκετές θεμελιώδες έννοιες που αποτελούν την βάση αυτών των τομέων. Αυτή η ενότητα παρέχει μια λεπτομερή επισκόπηση αυτών των εννοιών, θέτοντας τις βάσεις για τις συζητήσεις και αναλύσεις που ακολουθούν.

2.1 ΑΛΓΟΡΙΘΜΟΙ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗΣ

Οι Αλγόριθμοι Βελτιστοποίησης είναι σημαντικά και ισχυρά εργαλεία στην λύση περίπλοκων προβλημάτων scheduling. Σκοπός τους είναι να βρουν την καλύτερη λύση ή αλλιώς βέλτιστη [1] (ή αλλιώς optimal. Ένα σύστημα θεωρείτε βέλτιστο ως προς ένα μέτρο επίδοσης και ένα σύνολο περιορισμών εφόσον μας αποδίδει τουλάχιστον ίσα ή/και καλύτερα αποτελέσματα από κάθε άλλο σύστημα που ικανοποιεί τους ίδιους περιορισμούς), συνήθως μεγιστοποιώντας ή ελαχιστοποιώντας την αντικειμενική ή στοχική συνάρτηση (objective function). Η συνάρτηση αυτή είναι μια μαθηματική έκφραση του μέτρου επίδοσης και συμβολίζεται f(x). Αν κάθε εφικτή λύση μπορεί να περιγράφει από ένα σύνολο μεταβλητών ελέγχου (control variables) x = {x1 , x2 , ..., xn }

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

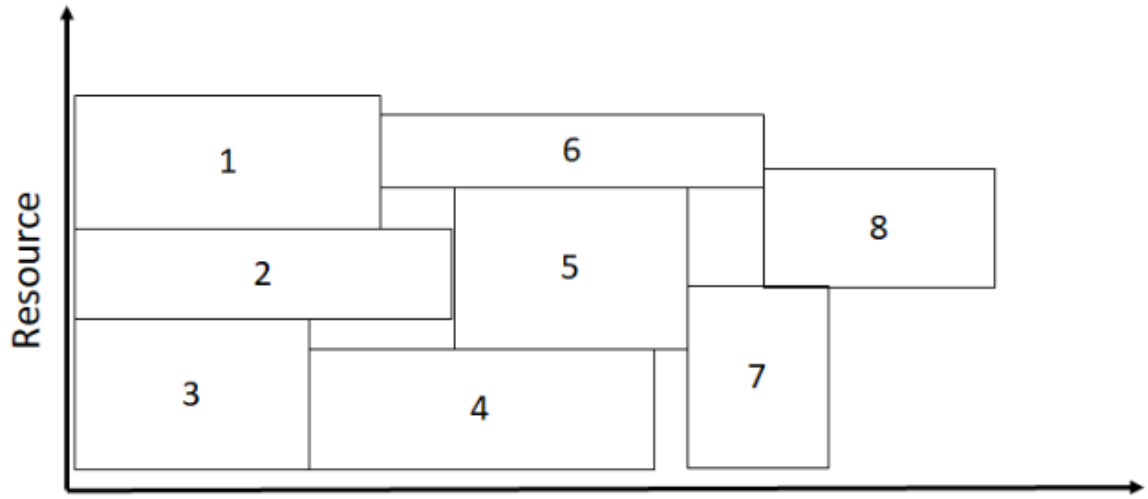
και αν σε κάθε τέτοια περιγραφή μπορεί να αντιστοιχιστεί ένα μέτρο επίδοσης, τότε ως βέλτιστη λαμβάνεται η απόφαση που μεγιστοποιεί το μέτρο αυτό. Η Βελτιστοποίηση περιλαμβάνει την στοχική συνάρτηση που προαναφέραμε, περιορισμούς που η λύση θα πρέπει να τηρεί, μια (τουλάχιστον) εφικτή λύση που υπακούει στους προαναφερόμενους περιορισμούς και, τέλος, την optimal λύση.

Η έννοια της βελτιστοποίησης εφαρμόζεται σε προβλήματα λήψης αποφάσεων, και προϋποθέτει μια διαδοχή από εναλλακτικές επιλογές και αξιολογήσεις των επιπτώσεων κάθε επιλογής. Οι αλγόριθμοι βελτιστοποίησης προσφέρουν μια ποικιλία στρατηγικών για την επίλυση σύνθετων προβλημάτων. Ο δυναμικός προγραμματισμός, για παράδειγμα, επιλύει ένα πρόβλημα χωρίζοντάς το σε μικρότερα, πιο διαχειρίσιμα υπό-προβλήματα. Οι λύσεις αυτών των υπό-προβλημάτων αποθηκεύονται για μελλοντική χρήση, αποφεύγοντας έτσι επαναλαμβανόμενους υπολογισμούς. Από την άλλη πλευρά, ο γραμμικός προγραμματισμός είναι μια μαθηματική τεχνική που χρησιμοποιείται ευρέως για την εύρεση της βέλτιστης λύσης σε προβλήματα που μπορούν να εκφραστούν ως μαθηματικά μοντέλα με γραμμικές συναρτήσεις και περιορισμούς. Τέλος, οι Greedy αλγόριθμοι αποτελούν μια απλή αλλά αποτελεσματική προσέγγιση για την επίλυση προβλημάτων βελτιστοποίησης. Σε κάθε βήμα, επιλέγεται η “λύση” που φαίνεται καλύτερη εκείνη τη στιγμή, χωρίς να εξετάζονται όλες οι πιθανές επιλογές.

Κατά την αξιολόγηση νέων αλγορίθμων οι ερευνητές θα πρέπει να ποσοτικοποιήσουν τις βελτιώσεις που προσφέρουν οι νέες μέθοδοι. Για να αξιολογήσουμε έναν αλγόριθμο βελτιστοποίησης, λαμβάνουμε υπόψη διάφορους παράγοντες. Εκτός από τον συνολικό χρόνο ολοκλήρωσης όλων των εργασιών (makespan), ενδιαφερόμαστε επίσης για την αποτελεσματική χρήση των πόρων του συστήματος. Επιπρόσθετα, η ομοιόμορφη κατανομή του φορτίου στα μηχανήματα, που ονομάζεται εξισορρόπηση φορτίου, είναι σημαντική. Τέλος, η ικανότητα του αλγορίθμου να προσαρμοστεί σε προβλήματα αυξανόμενου μεγέθους, δηλαδή η επεκτασιμότητά του, αποτελεί ένα κρίσιμο χαρακτηριστικό.

2.2 INTERVAL SCHEDULING

Το Interval Scheduling είναι ένα βασικό ζήτημα στη θεωρία του χρονοπρογραμματισμού που επικεντρώνεται στην επιλογή μιας συλλογής μη επικαλυπτόμενων διαστημάτων από ένα καθορισμένο σύνολο.



Γράφημα οπτικοποίησης list scheduling (Βιβλιογραφία 9)

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

Κάθε διάστημα χαρακτηρίζεται από μια ώρα έναρξης s και μια ώρα λήξης e, που αναπαρίσταται ως [s,e). Ο στόχος είναι να εντοπιστεί το μεγαλύτερο δυνατό υποσύνολο διαστημάτων που δεν συμπίπτουν. Το πρόβλημα αυτό μπορεί να απεικονιστεί ως χρονοδιάγραμμα όπου κάθε διάστημα σηματοδοτεί μια εργασία, με στόχο τον προγραμματισμό του μεγαλύτερου αριθμού αυτών των εργασιών χωρίς συγκρούσεις.

Στην τυπική του μορφή, ο χρονοπρογραμματισμός διαστημάτων λειτουργεί με βάση την παραδοχή ότι οι εργασίες είτε γίνονται αποδεκτές είτε απορρίπτονται με βάση την επικάλυψή τους με εργασίες που έχουν ήδη προγραμματιστεί. Ωστόσο, όπως θα δούμε και αναλυτικότερα στο κεφάλαιο 4, διευρύνουμε αυτή την ιδέα σε ένα πιο περίπλοκο σενάριο χρονοπρογραμματισμού. Αντί να επιλέγουμε απλώς μη επικαλυπτόμενες εργασίες, κατανέμουμε τις εργασίες σε πολλαπλές μηχανές με προκαθορισμένες δυνατότητες επεξεργασίας χάρις στους περιορισμένους τους πόρους (πυρήνες), διασφαλίζοντας ότι καμία μηχανή δεν υπερβαίνει το όριό της.

Αυτή η διαφορά προσθέτει επιπλέον απαιτήσεις:

1. Περιορισμοί πόρων: Κάθε μηχανή διαθέτει έναν πεπερασμένο αριθμό πόρων επεξεργασίας.

2. Μοναδική ανάθεση: Κάθε εργασία ανατίθεται σε μια συγκεκριμένη μηχανή.

3. Επικαλυπτόμενα χρονικά διαστήματα: Σε αντίθεση με τον παραδοσιακό χρονοπρογραμματισμό διαστημάτων, οι επικαλυπτόμενες εργασίες μπορούν να ανατεθούν στο ίδιο μηχάνημα, εφόσον τηρούνται οι περιορισμοί πόρων.

Επομένως, ενώ ο χρονοπρογραμματισμός κατά διαστήματα θέτει ένα θεμελιώδες πλαίσιο, το ζήτημα του χρονοπρογραμματισμού που συζητείται στην παρούσα διατριβή το επεκτείνει σε ένα πλαίσιο πολλαπλών μηχανών με περιορισμούς πόρων και έναν ξεχωριστό στόχο βελτιστοποίησης. Το πρόβλημα που αντιμετωπίζουμε ευθυγραμμίζεται περισσότερο με τον προγραμματισμό μηχανών που εστιάζει στην ελαχιστοποίηση του χρόνου απασχόλησης, όπου στοχεύουμε σε μια κατανομή που εξισορροπεί αποτελεσματικά το φόρτο εργασίας στις διαθέσιμες μηχανές.

Offline Προσέγγιση Λύση

Ένας δημοφιλής τρόπος για να επιλύσουμε το πρόβλημα του interval scheduling είναι ο άπληστος αλγόριθμος. Η στρατηγική του είναι απλή αλλά αποτελεσματική: **Ταξινομούμε** όλα τα διαστήματα με βάση τον χρόνο που ολοκληρώνονται (δηλαδή τον χρόνο τερματισμού τους). Στη συνέχεια, ξεκινάμε με ένα κενό σύνολο και προσθέτουμε διαδοχικά τα διαστήματα σε αυτό το σύνολο, **με την προϋπόθεση** ότι το νέο διάστημα που προσθέτουμε δεν επικαλύπτεται με κανένα από τα διαστήματα που έχουμε ήδη επιλέξει. Με αυτόν τον τρόπο, επιλέγουμε διαστήματα που τελειώνουν γρηγορότερα, αφήνοντας έτσι περισσότερο χρόνο για να προγραμματίσουμε άλλα διαστήματα.

Η άπληστη στρατηγική είναι βέλτιστη για την offline έκδοση, καθώς η απόδειξη βασίζεται σε επιχειρήματα ανταλλαγής, τα οποία δείχνουν ότι οποιαδήποτε διαφορετική επιλογή μπορεί να μετατραπεί σε άπληστη χωρίς να μειωθεί το μέγεθος του επιλεγμένου

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

συνόλου. Βέβαια ένας περιορισμός αυτής της στρατηγικής προϋποθέτει ότι έχουμε πρόσβαση σε όλες τις πληροφορίες (π.χ., αρχικοί και τελικοί χρόνοι για όλα τα διαστήματα) από την αρχή. Αυτό δεν ισχύει σε όλα τα περιβάλλοντα, όπως σε δυναμικές ή real-time εφαρμογές.

Online Προσέγγιση Λύση

Η διαχείριση του προβλήματος του interval scheduling σε ένα online περιβάλλον παρουσιάζει μοναδικές προκλήσεις. Σε αντίθεση με την offline έκδοση, όπου όλα τα διαστήματα είναι γνωστά εκ των προτέρων, στο online σενάριο τα διαστήματα φτάνουν σταδιακά. Αυτό σημαίνει ότι οι αποφάσεις σχετικά με το ποια διαστήματα θα επιλεγούν πρέπει να λαμβάνονται σε πραγματικό χρόνο, χωρίς τη δυνατότητα να έχουμε μια ολοκληρωμένη εικόνα του προβλήματος. Για να αντιμετωπίσουμε αυτή την αβεβαιότητα, οι online στρατηγικές βασίζονται συχνά σε ευρετικές μεθόδους, όπως η επιλογή των διαστημάτων που ξεκινούν πρώτα ή έχουν το μικρότερο μήκος. Ωστόσο, η αποτελεσματικότητα αυτών των μεθόδων εξαρτάται σε μεγάλο βαθμό από τη συγκεκριμένη σειρά που φτάνουν τα διαστήματα. Επομένως, σε αντίθεση με την offline έκδοση, δεν μπορούμε να είμαστε σίγουροι ότι η λύση που θα βρούμε είναι η βέλτιστη.

Ένα χαρακτηριστικό παράδειγμα είναι η διαχείριση ζωντανών δεδομένων (live streaming), όπου τα tasks δεν είναι γνωστά εκ των προτέρων και πρέπει να εκτελούνται καθώς προκύπτουν. Οι online αλγόριθμοι προσαρμόζονται συνεχώς στις συνθήκες, αλλά υστερούν σε σύγκριση με την offline βελτιστοποίηση.

Συγκρίνοντάς τους, βλέπουμε ότι η κάθε προσέγγιση έχει την χρησιμότητα του. Η offline μέθοδος είναι ιδανική για συστήματα όπου όλα τα tasks είναι προκαθορισμένα, όπως σε προγραμματισμό εργασιών σε batch. Αντίθετα, οι online μέθοδοι είναι κατάλληλες για συστήματα πραγματικού χρόνου, όπου οι αποφάσεις πρέπει να λαμβάνονται αμέσως. Ο συνδυασμός αυτών των στρατηγικών μπορεί να οδηγήσει σε υβριδικές λύσεις που προσφέρουν αποτελεσματικότητα και ευελιξία.

2.3 PARALLEL COMPUTING

Parallel Computing είναι ο συνδυασμός διαδικασιών ή εργασιών για την ταχύτερη επίλυση προβλημάτων υπολογιστή. Η κατανόηση αυτής της έννοιας είναι σημαντική για τον προσδιορισμό των εργασιών που μπορούν να εκτελεστούν από το πρόγραμμα.
Υπάρχουν διάφοροι τύποι παραλληλισμού στους υπολογιστές:
Παραλληλισμός δεδομένων: διανομή δεδομένων σε πολλαπλά νήματα και εκτέλεση της ίδιας λειτουργίας σε κάθε κομμάτι δεδομένων.

- **Καταμερισμός εργασίας**: διαίρεση της εργασίας σε πολλαπλά επίπεδα έτσι ώστε κάθε επεξεργαστής να εκτελεί διαφορετική λειτουργία.
- **Pipeline Parallelism**: διαίρεση εργασιών σε στάδια και εκτέλεση κάθε σταδίου παράλληλα

Συνοψίζοντας, οι θεμελιώδεις έννοιες που εξετάσαμε αποτελούν την απαραίτητη βάση για την κατανόηση των πιο σύνθετων αλγορίθμων και τεχνικών που θα παρουσιαστούν στη συνέχεια. Ο προγραμματισμός διαστημάτων και η βελτιστοποίηση πόρων είναι κρίσιμα θέματα σε συστήματα που απαιτούν αποδοτική χρήση υπολογιστικών πόρων, όπως

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

τα συστήματα παράλληλης επεξεργασίας. Η κατανόηση των βασικών αρχών του interval scheduling, των αλγορίθμων βελτιστοποίησης, και των τεχνικών παραλληλισμού είναι απαραίτητη για την ανάπτυξη αποτελεσματικών και κλιμακούμενων λύσεων. Οι έννοιες αυτές, αν και θεμελιώδεις, παρέχουν τα εργαλεία για την αντιμετώπιση πολύπλοκων προβλημάτων και τη βελτίωση της απόδοσης των συστημάτων που θα εξετάσουμε στις επόμενες ενότητες.

2.4 MOULDABLE TASKS

Τα "mouldable tasks" προσφέρουν μια ευέλικτη προσέγγιση στη διαχείριση υπολογιστικών πόρων. Αυτές οι εργασίες έχουν την ικανότητα να προσαρμόζουν δυναμικά τις απαιτήσεις τους σε επεξεργαστική ισχύ κατά τη διάρκεια της εκτέλεσής τους. Με άλλα λόγια, μπορούμε να αυξήσουμε ή να μειώσουμε τον αριθμό των επεξεργαστών που εκχωρούνται σε μια εργασία, ανάλογα με τις διαθέσιμες πόρους και τις τρέχουσες απαιτήσεις του συστήματος. Αυτή η ευελιξία επιτρέπει τη βελτιστοποίηση του χρόνου εκτέλεσης των εργασιών και την πιο αποτελεσματική αξιοποίηση των διαθέσιμων υπολογιστικών πόρων.

Κάθε mouldable task χαρακτηρίζεται από δύο βασικά στοιχεία:

- **Διάρκεια:** Ο ελάχιστος χρόνος που απαιτείται για την ολοκλήρωση της εργασίας, υπό την προϋπόθεση ότι διατίθενται όλοι οι απαραίτητοι πόροι.
- **Απαιτήσεις σε επεξεργαστές:** Ο αριθμός των επεξεργαστών που απαιτούνται για να ολοκληρωθεί η εργασία στον ελάχιστο δυνατό χρόνο.

Με την έννοια mouldable tasks, ονομάζουμε τις διεργασίες στα οποία ο αριθμός των επεξεργαστών, που χρησιμοποιούν για να εκτελεστούν, μπορεί να αποφασιστεί κατά την ώρα του προγραμματισμού (scheduling) των tasks άλλα πριν την εκτελεσή τους, σε αντίθεση με τα rigid tasks όπου οι επεξεργαστές του είναι πάντα σταθεροί και δεν αλλάζει η ποσότητα τους. Έτσι, τροποποιώντας τον αριθμό των επεξεργαστών, δυναμικά αλλάζεται και το runtime της διεργασίας. Ευνόητα αυτό σημαίνει πως η διάρκεια μπορεί να μειωθεί, αρά να εκτελεστεί πιο γρήγορα, αλλά και ότι μπορούμε να το καθυστερήσουμε δίνοντας του λιγότερους πόρους.

ΚΕΦΑΛΑΙΟ 3 RELATED WORK

Το πρόβλημα του scheduling σε υπολογιστικά περιβάλλοντα αποτελεί κεντρικό ζήτημα στην επιστήμη των υπολογιστών, με ευρείες εφαρμογές σε διάφορους τομείς, όπως το cloud computing, την βιομηχανική παραγωγή αλλά και την παράλληλη επεξεργασία. Οι αλγόριθμοι χρονοπρογραμματισμού διαδραματίζουν καθοριστικό ρόλο στη βελτιστοποίηση της χρήσης των πόρων, στη μείωση του συνολικού χρόνου εκτέλεσης των εργασιών και στη εξασφάλιση της ίσης μεταχείρισης των ανταγωνιστικών εργασιών. Αυτό το κεφάλαιο θα αναλύσει τους πιο γνωστούς αλγορίθμους, με έμφαση σε εκείνους που σχετίζονται με το πρόβλημα του χρονοπρογραμματισμού διαστημάτων και της παράλληλης επεξεργασίας.

Για να αντιμετωπίσουμε τις προκλήσεις του προγραμματισμού mouldable tasks σε περιβάλλοντα cloud, το έργο του Παναγιώτη Οικονόμου και συν. προτείνει μια πρωτοποριακή προσέγγιση που βασίζεται στην έννοια των "affinity pair thresholds". Αυτή η μέθοδος ομαδοποιεί παρόμοιες εργασίες με βάση συγκεκριμένα κριτήρια και στη

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

συνέχεια προχωρά σε έναν βελτιστοποιημένο προγραμματισμό, λαμβάνοντας υπόψη τόσο τις απαιτήσεις των εργασιών όσο και τη διαθεσιμότητα των πόρων. Σε σύγκριση με παραδοσιακές μεθόδους, όπως οι απλοί αλγόριθμοι First-fit ή Best-fit, η προσέγγιση των affinity pair thresholds προσφέρει μεγαλύτερη ευελιξία και αποτελεσματικότητα, ειδικά σε περιβάλλοντα με δυναμικά μεταβαλλόμενο φορτίο.

Συγκεκριμένα, η προσέγγιση αυτή προσφέρει τα ακόλουθα πλεονεκτήματα:

- **Βελτιωμένη αξιοποίηση πόρων**: Οι εργασίες με παρόμοια χαρακτηριστικά μπορούν να συγχωνευθούν, μειώνοντας το overhead και αυξάνοντας την αποτελεσματικότητα.
- **Μεγαλύτερη ευελιξία**: Οι αλγόριθμοι που βασίζονται σε αυτήν την προσέγγιση μπορούν να προσαρμοστούν σε ένα ευρύ φάσμα σκηνικών εργασίας. Μειωμένη πολυπλοκότητα: Η έννοια των affinity pair thresholds απλοποιεί τη διαδικασία λήψης αποφάσεων για τον προγραμματισμό.

Ενσωματώνοντας τις αρχές αυτής της έρευνας, μπορούμε να αναπτύξουμε πιο αποτελεσματικούς αλγορίθμους για τον προγραμματισμό mouldable tasks σε περιβάλλοντα cloud. Συνοπτικά, η ενσωμάτωση αυτής της έρευνας ενισχύει την επιστημονική βάση της εργασίας και δείχνει ότι η μελέτη βασίζεται σε ένα ισχυρό θεωρητικό υπόβαθρο. Επιπλέον, υπογραμμίζει τη συνεισφορά της εργασίας στη διεύρυνση των γνώσεων στον τομέα των mouldable tasks.

Τελικός μας σκοπός με τα mouldable tasks είναι να βελτιώσουμε τον "στόχο" του προβλήματος μας, είτε αυτό περιλαμβάνει την μεγιστοποίηση ή ελαχιστοποίηση. Στα πλαίσια του project μας, χρησιμοποιούμε την δυνατότητα να αυξήσουμε ή να μειώσουμε τους ζητούμενους πόρους μιας διεργασίας για να μειωθεί ο συνολικός χρόνος εκτέλεσης. Ο προγραμματισμός των tasks περιλαμβάνει τον υπολογισμό του optimal αριθμού επεξεργαστών που θα δοθούν στην κάθε διεργασία, είτε για την επιτάχυνση της διάρκειας του και, ως αποτέλεσμα, του συνολικού χρόνου, ή είτε για την καθυστέρηση του για την ελευθέρωση των πόρων ώστε να χρησιμοποιηθούν από πιο απαιτητικές εργασίες. Η απόφαση αυτή είναι επηρεασμένη από τρείς παράγοντες:

1. Τα tasks που εξαρτούνται από αλλά tasks, που χρειάζονται να ολοκληρωθούν για να τελειώσει το ίδιο
2. Την διαθεσιμότητα των πόρων στην τωρινή στιγμή
3. Το τρέχον φορτίο στο σύστημα που μπορεί να επηρεάσει τον προγραμματισμό των διεργασιών

Η ανασκόπηση επικεντρώνεται σε αλγορίθμους οι οποίοι χρησιμοποιούνται στον προσομοιωτή μας ως μέτρο σύγκρισης για τον δικό μας αλγόριθμο(τον οποίο θα αναλύσουμε σε ανερχόμενο κεφάλαιο),όπως οι BestFit, FirstFit και Bucket, αλλα και αλλούς moldable task algorithms και πως διαφέρουν με τον δικό μας.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

3.1 BEST FIT

Ο αλγόριθμος BestFit είναι ένα τυπικό παράδειγμα προσέγγισης συμπλήρωσης κάδου.(Οι Bin-packing algorithms χρησιμοποιούνται όταν οι εργασίες πρέπει να τοποθετηθούν σε έναν περιορισμένο αριθμό πόρων, όπως επεξεργαστές ή χρονικές θυρίδες). Λειτουργεί με την επιλογή του κάδου (ή του μηχανήματος) που αφήνει το μικρότερο ποσό εναπομείναντος χώρου μετά την κατανομή μιας εργασίας. Τα βήματα του αλγορίθμου BestFit έχουν ως εξής:

- **Αξιολόγηση καταλληλόλητας**: για κάθε εργασία, ο αλγόριθμος αξιολογεί όλες τις διαθέσιμες μηχανές και υπολογίζει την εναπομένουσα διαθέσιμη χωρητικότητα (εναπομείναντες πυρήνες) μετά την κατανομή της εργασίας.
- **Επιλογή της καλύτερης καταλληλόλητας**: η μηχανή με τη χαμηλότερη εναπομένουσα χωρητικότητα επιλέγεται για την κατανομή της εργασίας.
- **Δημιουργία νέας μηχανής**: εάν δεν μπορεί να βρεθεί κατάλληλη μηχανή, δημιουργείται νέα μηχανή.

Αυτός ο αλγόριθμος είναι ιδιαίτερα αποτελεσματικός όταν οι πόροι (π.χ. πυρήνες CPU) είναι περιορισμένοι και είναι σημαντικό να τοποθετηθούν όσο το δυνατόν περισσότερες εργασίες στους διαθέσιμους πόρους Το BestFit τείνει να μειώνει τον κατακερματισμό και είναι συχνά πιο αποδοτικότερος από τον FirstFit.

Ο BestFit θεωρείται γενικά ότι λειτουργεί καλά σε σενάρια όπου η κατανομή των μεγεθών των εργασιών είναι σχετικά ομοιόμορφη. Ωστόσο, σε περιπτώσεις όπου υπάρχουν πολλές μικρές εργασίες ή ένα μείγμα πολύ μεγάλων και πολύ μικρών εργασιών, το BestFit δεν δίνει πάντα τα καλύτερα αποτελέσματα. Παρ' όλα αυτά, το BestFit παραμένει μια δημοφιλής επιλογή λόγω της απλότητας και της αποτελεσματικότητάς του σε πολλά πρακτικά σενάρια.

3.2 FIRST FIT

Ο αλγόριθμος FirstFit [10] είναι ένας από τους απλούστερους και βασικότερους ευρετικούς αλγορίθμους χρονοπρογραμματισμού. Λειτουργεί με τη σάρωση του καταλόγου των διαθέσιμων μηχανημάτων και την τοποθέτηση κάθε εργασίας στο πρώτο μηχάνημα με επαρκείς πόρους για να την εξυπηρετήσει. Ο αλγόριθμος εκτελεί τα παρακάτω στάδια:

- **Αξιολογεί τα μηχανήματα με τη σειρά**: για κάθε εργασία, ο αλγόριθμος ελέγχει κάθε μηχάνημα με τη σειρά και σταματά μόλις βρει ένα μηχάνημα που μπορεί να φιλοξενήσει την εργασία.
- **Κατανομή εργασιών**: οι εργασίες κατανέμονται στο πρώτο μηχάνημα που πληροί τις απαιτήσεις.
- **Άνοιγμα νέου μηχανήματος**: εάν δεν βρεθεί κατάλληλο μηχάνημα, δημιουργείται ένα νέο μηχάνημα.

Ο FirstFit είναι πολύ αποδοτικός όσον αφορά το υπολογιστικό κόστος, καθώς δεν απαιτεί ταξινόμηση ή εκτεταμένες συγκρίσεις. Ωστόσο, δεν είναι απαραίτητα η πιο αποδοτική λύση από άποψη χρήσης πόρων, καθώς δεν λαμβάνει υπόψη τη συνολική κατανομή των

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

πόρων ή τη δυνατότητα καλύτερης κατανομής αργότερα στην ουρά. Ο FirstFit είναι γρήγορος λόγω της απλότητάς του, αλλά σε σύγκριση με τον BestFit, μπορεί να είναι πιο κατακερματισμένος, ιδίως σε σενάρια με εκτεταμένα μεγέθη εργασιών. Παρά αυτά τα πιθανά μειονεκτήματα, παραμένει δημοφιλής λόγω της απλότητάς του και της ευκολίας εφαρμογής του. Η ταχύτητα και η ευχρηστία του είναι οι βασικοί λόγοι που επιλέχθηκε για τα παραδείγματα μας, ενώ επίσης προσαρμόστηκε για να γίνει πιο ευέλικτος και προσαρμοστικός, αντί για έναν αυστηρό αλγόριθμο.

3.3 BUCKET

Ο αλγόριθμος Bucket εισάγει μια μέθοδο ταξινόμησης των εργασιών σε «κάδους» με βάση τη διάρκειά τους. Κάθε «κάδος» αντιστοιχεί σε ένα εύρος διάρκειας και οι εργασίες εντός αυτού του εύρους τοποθετούνται στον αντίστοιχο «κάδο» για περαιτέρω επεξεργασία. Η ιδέα πίσω από αυτή τη μέθοδο είναι η αποτελεσματικότερη κατανομή των πόρων, λαμβάνοντας υπόψη την ομοιογένεια των εργασιών εντός κάθε κατηγορίας. Ο αλγόριθμος bucket λειτουργεί με τα ακόλουθα βήματα:

- **Ταξινόμηση εργασιών**: οι εργασίες ταξινομούνται σε κατηγορίες (κάδους) ανάλογα με τη διάρκειά τους. Κάθε κατηγορία ορίζεται από ένα κατώτερο και ανώτερο όριο διάρκειας, που υπολογίζεται ως δύναμη του 2. Για παράδειγμα, εάν ένας κάδος ορίζεται με κατώτερο όριο 2^k και ανώτερο όριο 2^(k+1)-1, όλες οι εργασίες με διάρκεια εντός αυτού του εύρους θα βρίσκονται σε αυτόν τον κάδο.
- **Ανάθεση σε μηχανήματα**: για κάθε εργασία, ο αλγόριθμος ελέγχει σε ποιο κάδο ανήκει και στη συνέχεια την αναθέτει στο κατάλληλο μηχάνημα χρησιμοποιώντας μια τροποποιημένη έκδοση του αλγορίθμου FirstFit.
- **Δημιουργία νέου μηχανήματος**: εάν δεν μπορεί να βρεθεί κατάλληλο μηχάνημα για την επεξεργασία μιας εργασίας, δημιουργείται ένα νέο μηχάνημα, το οποίο επίσης ταξινομείται με βάση την ανατεθείσα εργασία.

Ο αλγόριθμος προσαρμόζεται δυναμικά στις προδιαγραφές κάθε εργασίας και παρέχει ένα ευέλικτο και αποτελεσματικό πλαίσιο για τη διαχείριση εργασιών με μεγάλες διακυμάνσεις στη διάρκεια.

Τέλος, ο Bucket προσφέρει αρκετά πλεονεκτήματα. Η κατηγοριοποίηση των εργασιών σε συγκεκριμένα εύρη διάρκειας βοηθά στην καλύτερη διαχείριση των διαθέσιμων πόρων, καθώς οι εργασίες με παρόμοιες απαιτήσεις επεξεργάζονται από κοινού, και, η ευελιξία του αλγορίθμου μπορεί να προσαρμοστεί εύκολα σε διαφορετικές κατηγορίες εργασιών, επιτρέποντας την καλύτερη εκμετάλλευση των δυνατοτήτων του συστήματος. Παρόλα αυτά, ο αλγόριθμος έχει και περιορισμούς. Η ανάγκη ορισμού κατάλληλων κατηγοριών και ορίων μπορεί να περιπλέξει τη σχεδίαση του αλγορίθμου, και αν τα tasks δεν κατανέμονται ομοιόμορφα στους κάδους, μπορεί να προκύψουν προβλήματα ανισορροπίας, με ορισμένα μηχανήματα να παραμένουν αδρανή ενώ άλλα να υπερφορτώνονται.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

3.4 MAX-MIN

Ο Max-Min Αλγόριθμος, όπως αναφέρουν οι Y.Mao και X. Li [4], για τον προγραμματισμό εργασιών σε ένα cloud περιβάλλον επικεντρώνεται στην εξισορρόπηση φορτίου και στη βελτιστοποίηση της διαχείρισης των εργασιών μεταξύ των εικονικών μηχανών (VMs). Είναι μια μέθοδος κατανομής εργασιών που δίνει προτεραιότητα στην ισορροπία φόρτου και τη βέλτιστη εκτέλεση μέσω κατανομής των εργασιών με βάση τον αναμενόμενο χρόνο εκτέλεσής τους. Κύριος στόχος είναι η ελαχιστοποίηση του χρόνου ολοκλήρωσης των εργασιών μέσω δυναμικής κατανομής, όπου κάθε εργασία ανατίθεται σε μια VM με βάση την αναμενόμενη διάρκεια εκτέλεσής της και το φορτίο κάθε VM.

Ο αλγόριθμος λειτουργεί ως εξής:

1. **Ταξινόμηση Εργασιών με Βάση τη Διάρκεια Εκτέλεσης**: Αρχικά, οι εργασίες ταξινομούνται ανάλογα με τον χρόνο που απαιτούν για εκτέλεση. Αυτό γίνεται για να εντοπιστεί η εργασία με τη μεγαλύτερη διάρκεια, καθώς η Max–Min προσέγγιση στοχεύει στην εκτέλεση αυτών των εργασιών πρώτα.
2. **Επιλογή Εικονικής Μηχανής (VM) με το Ελάχιστο Φορτίο**: Η εργασία με τον μεγαλύτερο χρόνο εκτέλεσης κατανέμεται στην VM με το χαμηλότερο φορτίο, δηλαδή στη μηχανή με τον μικρότερο αναμενόμενο χρόνο ολοκλήρωσης. Ο αλγόριθμος υπολογίζει αυτόν τον χρόνο μέσω της τρέχουσας κατάστασης της VM, συμπεριλαμβανομένων των υπόλοιπων εργασιών που είναι ήδη σε εκτέλεση.
3. **Διαδικασία Ενημέρωσης των Πινάκων Κατάστασης**: Ο πίνακας κατάστασης της εργασίας (Task Status Table) και ο πίνακας κατάστασης της VM (VM Status Table) ενημερώνονται για να αντικατοπτρίζουν την τρέχουσα κατανομή, τους χρόνους ολοκλήρωσης και την κατάσταση της VM.
4. **Επαναληπτική Διαδικασία Κατανομής**: Αυτή η διαδικασία συνεχίζεται μέχρι όλες οι εργασίες να κατανεμηθούν. Κάθε φορά που μια εργασία ολοκληρώνεται, αφαιρείται από τον πίνακα εργασιών και ενημερώνεται η κατάσταση των VM ώστε να παραμένει δυναμική η κατανομή εργασιών ανάλογα με την τρέχουσα φόρτωση του συστήματος.

Ο αλγόριθμος Max–Min επιτυγχάνει εξισορρόπηση φορτίου μειώνοντας τον χρόνο καθυστέρησης των εργασιών με τη μεγαλύτερη διάρκεια, βελτιστοποιώντας έτσι τη χρήση πόρων του cloud και την αποδοτικότητα του συστήματος. Η συνεχής ενημέρωση των πινάκων κατάστασης βοηθά στην ταχύτερη και πιο αποτελεσματική εκτέλεση των εργασιών, ενώ παράλληλα διασφαλίζει ότι καμία VM δεν υπερφορτώνεται. Ο αλγόριθμος διαφοροποιείται από τον δικό μας αλγόριθμο, καθώς ο δικός μας εστιάζει σε δυναμικό έλεγχο βασισμένο στην εκτίμηση διάρκειας και προσαρμογής των εργασιών με συγκεκριμένες μετρικές ταχύτητας και απαιτήσεων, κάτι που επιτρέπει πιο λεπτομερή έλεγχο κατανομής και ισορροπίας.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

ΚΕΦΑΛΑΙΟ 4 PROBLEM FORMULATION

Η αποτελεσματική διαχείριση των υπολογιστικών πόρων αποτελεί κεντρικό ζήτημα σε πολλές σύγχρονες εφαρμογές, από τα κέντρα δεδομένων μέχρι τα διανεμημένα συστήματα. Ένα βασικό υπό-πρόβλημα που προκύπτει σε αυτό το πλαίσιο είναι η κατανομή των εργασιών σε διαθέσιμες μηχανές, με στόχο την μεγιστοποίηση της απόδοσης και την ελαχιστοποίηση του χρόνου ολοκλήρωσης. Σε αυτό το κεφάλαιο, διατυπώνεται με ακρίβεια το πρόβλημα που επιχειρεί να επιλύσει η παρούσα εργασία. Η ανάλυση αφορά τη διαχείριση και τον προγραμματισμό ενός συνόλου διεργασιών (jobs) σε ένα σύνολο μηχανών (machines), λαμβάνοντας υπόψη περιορισμούς και απαιτήσεις πόρων.

4.1 ΒΑΣΙΚΕΣ ΟΝΤΟΤΗΤΕΣ ΚΑΙ ΟΡΙΣΜΟΙ

NOTATION TABLE

J	Σύνολο εργασιών= $\{j_1,j_2,j_3,\ldots,j_N\}$
J_y	Μία εργασία y που ανήκει στο σύνολο J.
s_y	Χρόνος εκκίνησης της εργασίας j_y.
c_y	Χρόνος ολοκλήρωσης της εργασίας j_y.
I_y	Διάστημα εκτέλεσης της εργασίας j_y: $Iy=[s_y,c_y)$
$L(I_y)$	Μήκος (διάρκεια) της εργασίας j_y:$L(I_y)=c_y-s_y$
d_y	Απαιτήσεις επεξεργασίας της εργασίας j_y
M	Σύνολο μηχανών: $M=\{m_1,m_2,\ldots,m_N\}$
m_i	Μία μηχανή i που ανήκει στο σύνολο M.
k_i	Διαθέσιμοι πόροι (π.χ. πυρήνες) της μηχανής m_i
F	Δυαδική μήτρα κατανομής των εργασιών στις μηχανές.
F_{ij}	Στοιχείο της μήτρας F: $F_{i,j}=1$ αν η εργασία j εκχωρείται στη μηχανή m_i, αλλιώς $F_{i,j}=0$.

Έστω ένα σύνολο εργασιών $J=\{j_1,j_2,j_3,\ldots,j_N\}$, όπου κάθε εργασία j_y χαρακτηρίζεται από τον χρόνο εκκίνησης s_y, που δηλώνει πότε ξεκινά η εργασία, τον χρόνο ολοκλήρωσης c_y, που δηλώνει πότε ολοκληρώνεται, το διάστημα εκτέλεσης $I_y=[s_y,c_y)$, το οποίο ορίζει την περίοδο εκτέλεσης της εργασίας, το μήκος εργασίας $L(I_y)=c_y-s_y$, που είναι η διάρκειά της, και τις απαιτήσεις επεξεργασίας d_y, που ορίζουν τους πόρους που απαιτούνται. Παράλληλα, θεωρούμε ένα σύνολο μηχανών $M=\{m_1,m_2,\ldots,m_N\}$, όπου κάθε μηχανή έχει συγκεκριμένο αριθμό διαθέσιμων πόρων k_i, οι οποίοι αντιστοιχούν σε πυρήνες ή μονάδες επεξεργασίας της μηχανής m_i. Η κατανομή των εργασιών στις μηχανές περιγράφεται μέσω μιας δυαδικής μήτρας F, όπου κάθε στοιχείο $F_{i,j}$ δηλώνει αν η εργασία j_j εκχωρείται στη μηχανή m_i ($F_{i,j}=1$) ή όχι ($F_{i,j}=0$).

Με αυτόν τον τρόπο, μπορούμε να μοντελοποιήσουμε το πρόβλημα του προγραμματισμού εργασιών ως ένα μαθηματικό πρόβλημα, όπου ο στόχος είναι να βρεθεί μια κατανομή εργασιών που να μεγιστοποιεί κάποια συγκεκριμένη μετρική απόδοσης, όπως η συνολική διάρκεια εκτέλεσης όλων των εργασιών ή η μέγιστη απασχόληση μιας

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

μηχανής. Η F καθορίζει την κατανομή των διεργασιών στις μηχανές. Μια σωστή ανάθεση πρέπει να διασφαλίζει πως καμία μηχανή δεν υπερβαίνει τους διαθέσιμους πόρους της αλλά και όλες οι διεργασίες να είναι μοναδικά ταξινομημένες, δηλαδή να μην εκτελούνται ταυτόχρονα σε περισσότερες από μία μηχανές.

4.2 ΧΡΟΝΟΣ ΑΠΑΣΧΟΛΗΣΗΣ ΜΗΧΑΝΩΝ

Ο χρόνος απασχόλησης (busy time) κάθε μηχανής mi δίνεται από $B_i(F)$, όπου F είναι η μήτρα ανάθεσης. Ο υπολογισμός του συνολικού χρόνου απασχόλησης εξαρτάται από τα διαστήματα εκτέλεσης των εργασιών που εκχωρούνται στη μηχανή m_i, όπως φαίνεται στην παρακάτω εξίσωση:

$$Bi(F) = L\left(\bigcup_{y=1}^{N} Fi\,, y\, Iy\right)$$

Όπου $F_{i,y}$ I_y συμβολίζει την ένωση όλων των διαστημάτων I_y των διεργασιών που έχουν εκχωρηθεί στη μηχανή, και L την λειτουργία που υπολογίζει το συνολικό μήκος ενός συνόλου μη επικαλυπτόμενων διαστημάτων. Σαν σύνολο, η εξίσωση υπολογίζει το συνολικό χρόνο κατά τον οποίο η μηχανή mi είναι ενεργή, λαμβάνοντας υπόψη τυχόν επικαλύψεις διαστημάτων που καταλαμβάνονται από διαφορετικές διεργασίες.

4.3 ΣΤΟΧΟΣ ΚΑΙ ΠΕΡΙΟΡΙΣΜΟΙ

Ο βασικός στόχος του προβλήματος βελτιστοποίησης που εξετάζουμε είναι να βρεθεί ένας τρόπος κατανομής των εργασιών στις διαθέσιμες μηχανές, έτσι ώστε να ελαχιστοποιηθεί ο συνολικός χρόνος ολοκλήρωσης όλων των εργασιών. Με άλλα λόγια, επιδιώκουμε να ολοκληρωθούν όλες οι εργασίες στο συντομότερο δυνατό χρόνο.

$$\min \sum_{i=1}^{M} Bi(F)$$

Ωστόσο, αυτή η βελτιστοποίηση πρέπει να γίνει υπό συγκεκριμένους περιορισμούς: Αρχικά πρέπει να λειτουργούμε υπό **περιορισμούς πόρους** καθώς αν είχαμε απεριόριστους δεν θα είχαμε πρόβλημα να λύσουμε. Κάθε μηχανή διαθέτει έναν περιορισμένο αριθμό πόρων (π.χ., πυρήνες επεξεργαστή), και το σύνολο των εργασιών που εκτελούνται ταυτόχρονα σε μια μηχανή δεν πρέπει να απαιτεί περισσότερους πόρους από όσους είναι διαθέσιμοι. Για να είναι πιο πιστός σε πρακτικά σενάρια (όπως προαναφέραμε cloud gaming, broadcasting κτλ.) ο προσομοιωτής μας, περιορίζουμε τις διεργασίες μας σε μοναδική εκχώρηση, δηλαδή κάθε εργασία πρέπει να εκχωρηθεί σε μία και μόνο μία μηχανή. Δεν επιτρέπουμε μια εργασία να εκτελείται ταυτόχρονα σε πολλές μηχανές. Τέλος, σε ορισμένες περιπτώσεις, η κατανομή των εργασιών μπορεί να περιορίζεται από την τοπολογία του συστήματος. Για παράδειγμα, εργασίες που επικοινωνούν έντονα

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

μεταξύ τους μπορεί να είναι προτιμότερο να εκχωρηθούν σε μηχανές που βρίσκονται κοντά μεταξύ τους.

Η παραπάνω διατύπωση παρέχει ένα σαφές μαθηματικό πλαίσιο για την ανάλυση του προβλήματος και τον σχεδιασμό κατάλληλων αλγορίθμων προγραμματισμού εργασιών. Στα επόμενα κεφάλαια θα παρουσιαστούν συγκεκριμένες τεχνικές και μέθοδοι που επιλύουν αυτό το πρόβλημα, με στόχο τη μεγιστοποίηση της αποδοτικότητας και την ελαχιστοποίηση του χρόνου απασχόλησης.

ΚΕΦΑΛΑΙΟ 5 ΠΡΟΤΙΝΟΜΕΝΟΙ ΑΛΓΟΡΙΘΜΟΙ

5.1 GREEDY

Ο αλγόριθμος Greedy βασίζεται στη θεώρηση ότι η άμεση και πλήρης κατανομή πόρων (π.χ. πυρήνων) σε κάθε νέα εργασία οδηγεί σε ταχύτερη εκτέλεση, αν και ενδέχεται να μην είναι η πιο αποδοτική κατανομή μακροπρόθεσμα. Το σκεπτικό του βασίζεται στην απλότητα και στην άμεση ανταπόκριση στις ανάγκες των εργασιών, χωρίς να επιδιώκει εξειδικευμένες βελτιστοποιήσεις στην κατανομή των πόρων.

```
1. CLASS Greedy
2.    METHOD INIT(scheduler)
3.       SET self.scheduler TO scheduler
4.    END METHOD
5.
6.    METHOD new_machine(job) # Μέθοδος που αναλαμβάνει την δημιουργία νέου server
7.       CREATE new_machine WITH (scheduler.cores, job.ar)
8.       CALL new_machine.add_job(job)
9.       CALL self.scheduler.add_machine(new_machine)
10.   END METHOD
11.
12.   METHOD calculate_duration_with_new_req(job, new_req)
13.      IF new_req IS 0 THEN
14.         RETURN INFINITY  # Ένα task δεν μπορεί να επεξεργαστεί με 0 cores
15.      END IF
16.      SET current_cores TO job.req
17.      SET current_dur TO job.dur
18.      SET new_dur TO (current_cores / new_req) * current_dur
19.      RETURN new_dur
20.   END METHOD
21.
22.   METHOD pack(job, machines=None, opennew=True)
23.      IF machines IS NULL THEN
24.         SET machines TO self.scheduler.machines
25.      END IF
26.
27.      # Αν δεν υπάρχουν διαθέσιμα μηχανήματα, δημιουργούμε ένα καινούργιο
28.      IF machines IS EMPTY THEN
29.         CALL self.new_machine(job)
30.         RETURN 1  # Scheduling is done
31.      END IF
32.
33.      # Κάνουμε schedule για αρχή με τον First-fit αλγόριθμο
34.      FOR EACH machine IN machines DO
```

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

```
35.          IF self.scheduler.check_fit(machine, job) THEN
36.             CALL machine.add_job(job)
37.             RETURN 1  # Τερματισμός προγραμματισμού των jobs
38.          END IF
39.       END FOR
40.
41.       # Κρατάμε ένα αντίγραφο του αρχικού task πριν μειώσουμε τα απαιτούμενα cores
42.       SET original_job TO COPY(job)
43.
44.       # Μειώνουμε τα cores του job και ψάχνουμε ένα machine
45.       WHILE job.req > 1 DO
46.          DECREMENT job.req BY 1
47.          SET job.dur TO self.calculate_duration_with_new_req(job, job.req)
48.
49.          FOR EACH machine IN machines DO
50.             SET max_duration TO MAXIMUM duration OF jobs IN machine
51.             IF job.dur <= max_duration * 2 THEN
52.                CALL machine.add_job(job)
53.                RETURN 1  # Τερματισμός προγραμματισμού
54.             END IF
55.          END FOR
56.       END WHILE
57.
58.       # Αν δεν βρούμε διαθέσιμο server, φτιάχνουμε καινούργιο
59.       SET job TO original_job  # Επαναφορά του job στην αρχική του μορφή
60.       IF opennew THEN
61.          CALL self.new_machine(job)
62.          RETURN 1  # Τερματισμός προγραμματισμού
63.       END IF
64.
65.       RETURN 0  # Αν φτάσουμε εδώ, δεν μπορούσαμε να αναθέσουμε κάπου το task
66.    END METHOD
67. END CLASS
68.
```

Greedy Algorithm

Συγκεκριμένα, όταν μια νέα εργασία εμφανίζεται, ο Greedy προσπαθεί να της δώσει όσους περισσότερους πόρους γίνεται σε ένα VM, ώστε να εκτελεστεί γρήγορα. Στην περίπτωση που δεν υπάρχουν επαρκείς πόροι στις υπάρχουσες VMs, δημιουργεί ένα νέο VM και αναθέτει σε αυτήν όλους τους διαθέσιμους πυρήνες για την εκτέλεση της εργασίας. Αυτό μειώνει τη διάρκεια της εργασίας, καθώς η διάθεση περισσότερων πυρήνων μειώνει το χρόνο εκτέλεσης.

Θεωρητικά, ο Greedy αλγόριθμος είναι χρήσιμος σε περιπτώσεις όπου η άμεση ολοκλήρωση των εργασιών προέχει της αποδοτικής χρήσης των πόρων. Ωστόσο, λόγω της έλλειψης στρατηγικής στη διαχείριση των πόρων, ενδέχεται να δημιουργήσει αναποτελεσματική κατανομή των VMs, ειδικά σε περιπτώσεις που οι εργασίες έχουν διαφορετικές απαιτήσεις σε πόρους. Αυτή η προσέγγιση ταιριάζει σε συστήματα όπου η ταχύτητα εκτέλεσης υπερτερεί του κόστους που ενδέχεται να προκύψει από την υπερβολική κατανάλωση πόρων ή την υπερβολική δημιουργία VMs.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

5.2 MAX DURATION 2X

Ο αλγόριθμος Max_Dur2x εστιάζει στην ορθολογική διαχείριση των πόρων μέσω της σταδιακής προσαρμογής των απαιτήσεων σε πυρήνες για μια εργασία, μέχρι αυτή να μπορεί να προγραμματιστεί σε μια υπάρχουσα εικονική μηχανή χωρίς να προκαλέσει δυσανάλογη καθυστέρηση. Η προσέγγιση αυτή αποσκοπεί στη βελτιστοποίηση της απόδοσης με τον περιορισμό της διάρκειας κάθε νέας εργασίας σε σχέση με τη διάρκεια των υφιστάμενων εργασιών στις διαθέσιμες VMs.

```
1. CLASS MaxDuration2X
2.    METHOD INIT(scheduler)
3.       SET self.scheduler TO scheduler
4.    END METHOD
5.
6.    METHOD new_machine(job) # Μέθοδος που αναλαμβάνει την δημιουργία νέου server
7.       CREATE new_machine WITH (scheduler.cores, job.ar)
8.       CALL new_machine.add_job(job)
9.       CALL self.scheduler.add_machine(new_machine)
10.   END METHOD
11.
12.   METHOD calculate_duration_with_new_req(job, new_req)
13.      IF new_req IS 0 THEN
14.         RETURN INFINITY  # Ένα task δεν μπορεί να επεξεργαστεί με 0 cores
15.      END IF
16.      SET current_cores TO job.req
17.      SET current_dur TO job.dur
18.      SET new_dur TO (current_cores / new_req) * current_dur
19.      RETURN new_dur
20.   END METHOD
21.
22.   METHOD pack(job, machines=None, opennew=True)
23.      IF machines IS NULL THEN
24.         SET machines TO self.scheduler.machines
25.      END IF
26.
27.      # Αν δεν υπάρχουν διαθέσιμα μηχανήματα, δημιουργούμε ένα καινούργιο
28.      IF machines IS EMPTY THEN
29.         CALL self.new_machine(job)
30.         RETURN 1  # Scheduling is done
31.      END IF
32.
33.      # Κάνουμε schedule για αρχή με τον First-fit αλγόριθμο
34.      FOR EACH machine IN machines DO
35.         IF self.scheduler.check_fit(machine, job) THEN
36.            CALL machine.add_job(job)
37.            RETURN 1  # Τερματισμός προγραμματισμού των jobs
38.         END IF
39.      END FOR
40.
41.      # Κρατάμε ένα αντίγραφο του αρχικού task πριν μειώσουμε τα απαιτούμενα cores
42.      SET original_job TO COPY(job)
43.
44.      # Μειώνουμε τα cores του job και ψάχνουμε ένα machine
45.      WHILE job.req > 1 DO
46.         DECREMENT job.req BY 1
47.         SET job.dur TO self.calculate_duration_with_new_req(job, job.req)
```

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

```
48.
49.          FOR EACH machine IN machines DO
50.             SET max_duration TO MAXIMUM duration OF jobs IN machine
51.             IF job.dur <= max_duration * 2 THEN # Αν η διάρκεια του task είναι <= της διάρκειας του
μεγαλύτερου ανατεθειμένου task στο τωρινό επιλεγμένο server
52.                CALL machine.add_job(job)
53.                RETURN 1  # Τερματισμός προγραμματισμού
54.             END IF
55.          END FOR
56.       END WHILE
57.
58.       # Αν δεν βρούμε διαθέσιμο server, φτιάχνουμε καινούργιο
59.       SET job TO original_job  # Επαναφορά του job στην αρχική του μορφή
60.       IF opennew THEN
61.          CALL self.new_machine(job)
62.          RETURN 1  # Τερματισμός προγραμματισμού
63.       END IF
64.
65.       RETURN 0  # Αν φτάσουμε εδώ, δεν μπορούσαμε να αναθέσουμε κάπου το task
66.    END METHOD
67. END CLASS
68.
```

Max_Dur2x Algorithm

Πιο συγκεκριμένα, ο αλγόριθμος πρώτα επιχειρεί να εντάξει την εργασία με το πλήρες αίτημα πυρήνων, αλλά αν δεν υπάρχει κατάλληλη VM, μειώνει το αίτημα σε πυρήνες για να βρει συμβατή VM. Για κάθε διαθέσιμη VM, ελέγχει αν η διάρκεια της εργασίας θα είναι μικρότερη από το διπλάσιο της μεγαλύτερης διάρκειας εργασίας στη VM αυτή. Αν δεν ικανοποιείται καμία συνθήκη, δημιουργεί νέα VM.

Αναλύοντας τον αλγόριθμο σε βήματα, λειτουργεί ως εξής:

1. **Δημιουργία νέας μηχανής αν δεν υπάρχουν διαθέσιμες**: Αν δεν υπάρχουν διαθέσιμες μηχανές για εκτέλεση εργασιών, ο αλγόριθμος δημιουργεί μια νέα μηχανή και εκχωρεί την εργασία εκεί.
2. **Κανονικός καταμερισμός**: Ο αλγόριθμος αρχικά επιχειρεί να τοποθετήσει την εργασία σε μια υπάρχουσα μηχανή, ελέγχοντας αν η εργασία μπορεί να εκτελεστεί με τον απαιτούμενο αριθμό πυρήνων χωρίς να ξεπεράσει τους περιορισμούς διαθεσιμότητας.
3. **Μείωση πυρήνων και επανεκτίμηση**: Αν η εργασία δεν μπορεί να εκτελεστεί σε καμία μηχανή, ο αλγόριθμος μειώνει σταδιακά τον αριθμό πυρήνων που απαιτεί η εργασία, προσπαθώντας να προσαρμόσει τη διάρκεια εκτέλεσης της σε τέτοιο τρόπο ώστε να είναι αποδεκτή από τις υπάρχουσες μηχανές. Κάθε φορά που μειώνεται ο αριθμός πυρήνων, υπολογίζεται εκ νέου η διάρκεια εκτέλεσης της εργασίας βάσει του νέου αριθμού πυρήνων.
4. **Έλεγχος διπλάσιου χρόνου εκτέλεσης**: Με κάθε νέα διαμόρφωση πυρήνων, ο αλγόριθμος ελέγχει αν η νέα διάρκεια της εργασίας είναι μικρότερη ή ίση με το διπλάσιο της μεγαλύτερης διάρκειας εκτέλεσης των εργασιών που ήδη τρέχουν στη συγκεκριμένη μηχανή. Αν ναι, η εργασία εκχωρείται σε αυτή τη μηχανή.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

5. **Δημιουργία νέας μηχανής αν αποτύχει ο προσαρμοσμένος καταμερισμός**: Αν δεν βρεθεί κατάλληλη μηχανή μετά από τις παραπάνω προσαρμογές, ο αλγόριθμος δημιουργεί μια νέα μηχανή και τοποθετεί την αρχική εργασία με τις απαιτούμενες ρυθμίσεις.

Η λογική αυτή επιτρέπει τον περιορισμό των καθυστερήσεων και των επανεκκινήσεων, διατηρώντας τη διάρκεια της εργασίας σε πιο εύκολα διαχειρίσιμα επίπεδα σε σχέση με τις υπόλοιπες ενεργές διεργασίες.

5.3 MOLDABLE PARALLEL TASKS

Ο αλγόριθμος **MPT** (Moldable Parallel Tasks) έχει σχεδιαστεί για τη διαχείριση «διαμορφώσιμων» εργασιών σε περιβάλλοντα υψηλών υπολογιστικών απαιτήσεων. Οι διαμορφώσιμες εργασίες είναι εκείνες που μπορούν να προσαρμόσουν δυναμικά τον αριθμό των πυρήνων τους, ανάλογα με τους διαθέσιμους πόρους, επιτρέποντας έτσι ευέλικτη και αποδοτική χρήση των μηχανών.

```
1. CLASS MPT
2.    METHOD INIT(scheduler)
3.       INPUT: scheduler
4.       SET self.scheduler TO scheduler
5.       SET self.blackbox TO FirstFit(self.scheduler)  # Αρχικοποίηση του First Fit σε ένα “black box”
6.       INITIALIZE self.core_capacity AS a dictionary with machine: self.scheduler.cores FOR EACH machine IN
self.scheduler.machines
7.    END METHOD
8.
9.    METHOD f(job, αi)
10.      INPUT: job, αi
11.      SET eligible_machines TO an empty list
12.      FOR EACH machine IN self.scheduler.machines DO
13.         FOR s IN RANGE(1 TO self.core_capacity[machine] + 1) DO
14.            IF calculate_dur(job, s) <= αi THEN
15.               APPEND (machine, s) TO eligible_machines
16.            END IF
17.         END FOR
18.      END FOR
19.      RETURN eligible_machines
20.   END METHOD
21.
22.   METHOD calculate_dur(job, new_cores) # Υπολογισμός διάρκειας ενός job
23.      INPUT: job, new_cores
24.      SET current_cores TO job.req
25.      SET current_dur TO job.dur
26.      SET new_dur TO (current_cores / new_cores) * current_dur
27.      RETURN new_dur
28.   END METHOD
29.
30.   METHOD update_duration_mouldable(job, new_cores) #Ενημέρωση του duration αφού μετατρέψουμε το
task σε moldable
31.      INPUT: job, new_cores
32.      SET job.dur TO calculate_dur(job, new_cores)
33.      SET job.req TO new_cores
34.      RETURN job
35.   END METHOD
```

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

```
36.
37.    METHOD new_machine(job) # Αν δεν υπάρχουν διαθέσιμα μηχανήματα, δημιουργούμε ένα καινούργιο
38.        INPUT: job
39.        CREATE new_machine WITH (self.scheduler.cores, job.ar)
40.        CALL new_machine.add_job(job)
41.        CALL self.scheduler.add_machine(new_machine)
42.        SET self.core_capacity[new_machine] TO self.scheduler.cores  # Αρχικοποίηση διαθέσιμων πυρήνων για
κάθε server
43.    END METHOD
44.
45.    METHOD update_core_capacity(machine, used_cores) #Ενημέρωση των διαθέσιμων πυρήνων στον server
46.        INPUT: machine, used_cores
47.        DECREMENT self.core_capacity[machine] BY used_cores
48.    END METHOD
49.
50.    METHOD pack(job)
51.        INPUT: job
52.        SET αi TO job.dur  # Αρχικοποίηση του αi με την αρχική διάρκεια ενός task
53.        SET Wi TO 0
54.        SET Is_scheduled TO FALSE
55.
56.        WHILE NOT Is_scheduled DO
57.            SET eligible_machines TO f(job, αi)
58.  # Όσο δεν είναι ανατεθειμένο σε ένα μηχάνημα
59.            IF eligible_machines IS NOT EMPTY THEN
60.                SET (min_machine, min_s) TO the pair (machine, cores) in eligible_machines with the minimum
value of cores * calculate_dur(job, cores)
61.                INCREMENT Wi BY min_s * calculate_dur(job, min_s)
62.            ELSE
63.                CALL new_machine(job)
64.
65.            IF Wi <= αi * LENGTH(self.scheduler.machines) AND eligible_machines IS NOT EMPTY THEN
66.                SET Is_scheduled TO TRUE
67.                SET job_with_new_duration TO update_duration_mouldable(job, min_s)
68.                CALL self.blackbox.pack(job_with_new_duration, [min_machine], opennew=False)  # Κάνουμε
schedule με τον First-fit αλγόριθμο
69.                CALL update_core_capacity(min_machine, min_s)
70.            ELSE
71.                MULTIPLY αi BY 2
72.                SET Wi TO 0
73.        END WHILE
74.    END METHOD
75. END CLASS
76.
```

MoldableParallelTasks Algorithm

Στόχος του MPT είναι να τοποθετήσει κάθε εργασία στην κατάλληλη μηχανή με τον βέλτιστο αριθμό πυρήνων, ώστε να ελαχιστοποιηθεί ο χρόνος εκτέλεσης και να αυξηθεί η αποδοτικότητα των πόρων. Ο αλγόριθμος ακολουθεί μία προσαρμοστική προσέγγιση, όπου οι εργασίες ανατίθενται δυναμικά στις μηχανές, ανάλογα με το αν πληρούνται τα χρονικά όρια εκτέλεσης.

1. **Εύρεση Κατάλληλων Μηχανών**: Ο MPT αρχικά ελέγχει τις διαθέσιμες μηχανές και καθορίζει αν κάποια από αυτές μπορεί να εκτελέσει την εργασία εντός ενός προκαθορισμένου χρονικού ορίου, το οποίο υπολογίζεται ως η διάρκεια της εργασίας (αi). Η συνάρτηση f αναζητά κατάλληλους συνδυασμούς

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

μηχανών και πυρήνων, όπου η προσαρμοσμένη διάρκεια εκτέλεσης της εργασίας δεν υπερβαίνει το ai.

2. **Υπολογισμός Διάρκειας**: Η συνάρτηση calculate_dur υπολογίζει τη διάρκεια εκτέλεσης της εργασίας για διαφορετικό αριθμό πυρήνων, επιτρέποντας τη δυναμική ρύθμιση των απαιτήσεων της εργασίας. Αυτός ο υπολογισμός διευκολύνει την προσαρμογή της διάρκειας στις δυνατότητες της μηχανής και εξασφαλίζει ότι η εργασία θα ολοκληρωθεί όσο το δυνατόν πιο γρήγορα.
3. **Δημιουργία Νέων Μηχανών**: Αν δεν υπάρχουν μηχανές που να πληρούν τα χρονικά κριτήρια εκτέλεσης, ο αλγόριθμος δημιουργεί νέα μηχανή και την προσθέτει στο σύστημα, εξασφαλίζοντας έτσι ότι όλες οι εργασίες θα εκτελεστούν χωρίς καθυστέρηση.
4. **Δυναμική Προσαρμογή Διάρκειας και Αριθμού Πυρήνων**: Η διάρκεια εκτέλεσης ai διπλασιάζεται σε κάθε κύκλο, ενώ ο MPT αναζητά εκ νέου κατάλληλες μηχανές. Αυτό επιτρέπει στην εργασία να διαμορφωθεί με βάση τους διαθέσιμους πόρους, αυξάνοντας ή μειώνοντας τους απαιτούμενους πυρήνες αναλόγως.

Ο MPT αξιοποιεί στο έπακρο τους υπολογιστικούς πόρους με δύο κύριους τρόπους: αφενός, διασφαλίζει ότι οι εργασίες θα εκτελεστούν εντός των χρονικών ορίων που έχουν οριστεί, και αφετέρου, μειώνει τον συνολικό χρόνο εκτέλεσης, μεγιστοποιώντας την αποδοτικότητα του συστήματος. Η δυνατότητα προσαρμογής του αριθμού πυρήνων για κάθε εργασία επιτρέπει την καλύτερη αξιοποίηση των διαθέσιμων μηχανών, καθιστώντας τον αλγόριθμο ιδανικό για σύγχρονα πολυπύρηνα και κατανεμημένα περιβάλλοντα.

5.4 MOLDABLE FIRST FIT

Ο αλγόριθμος **MoldFF** επεκτείνει την ευρετική στρατηγική του FirstFit με έμφαση στην αποδοτική κατανομή των εργασιών σε μηχανές. Η λειτουργία του βασίζεται σε ένα βήμα προσαρμογής, όπου πρώτα υπολογίζει τον βέλτιστο αριθμό πυρήνων για κάθε εργασία χρησιμοποιώντας ένα μοντέλο μεταβλητότητας ("low" ή "high variance"), προσαρμόζοντας τη διάρκεια εκτέλεσης ανάλογα με τη διαθεσιμότητα των πόρων.

1. **Υπολογισμός Βέλτιστου Αριθμού Πυρήνων**: Χρησιμοποιώντας τα μοντέλα optimal_coresLOW ή optimal_coresHIGH, ο αλγόριθμος καθορίζει τον ιδανικό αριθμό πυρήνων για την εκτέλεση της εργασίας με αποδοτική χρήση των πόρων. Εφαρμόζει στρογγυλοποίηση προς τα πάνω, για να διασφαλίσει ότι δεν θα προκύψουν σφάλματα κατά την ανάθεση.
2. **Αντιστοίχιση Εργασίας σε Μηχανήματα**: Ο MoldFF εφαρμόζει τον αλγόριθμο FirstFit, τοποθετώντας την εργασία στην πρώτη διαθέσιμη μηχανή που μπορεί να τη φιλοξενήσει χωρίς παραβίαση των προϋποθέσεων (όπως έλεγχο της διάρκειας της εργασίας σε σχέση με τις υπόλοιπες εργασίες στη μηχανή). Αν καμία μηχανή δεν πληροί τις απαιτήσεις, ο αλγόριθμος δημιουργεί νέα μηχανή.
3. **Ιστορικό Εργασιών**: Καταγράφει την αρχική και την προσαρμοσμένη κατάσταση των εργασιών για ανάλυση της αποδοτικότητας.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

Σε σχέση με τον κλασικό FirstFit, ο MoldFF προσθέτει βελτιώσεις στην κατανομή με την προσαρμογή των απαιτήσεων εργασιών, προκειμένου να ελαχιστοποιηθεί η δημιουργία νέων μηχανών και να βελτιωθεί η χρήση των πόρων. Ας εξετάσουμε τους αλγόριθμους σε ένα διάγραμμα χρόνου.

```
1. CLASS MoldFF
2.   METHOD INIT(scheduler)
3.     INPUT: scheduler
4.     SET self.scheduler TO scheduler
5.     SET self.variance_model TO 'high'
6.     INITIALIZE self.molded_job_history TO a dictionary with keys 'before' and 'after', both as empty lists
7.   END METHOD
8.
9.   METHOD new_machine(job)
10.    INPUT: job # δημιουργία καινούργιου server
11.    CREATE new_machine WITH (self.scheduler.cores, job.ar)
12.    CALL new_machine.add_job(job)
13.    CALL self.scheduler.add_machine(new_machine)
14.  END METHOD
15.
16.  METHOD pack(job, machines=None, opennew=True)
17.    INPUT: job, machines (optional), opennew (default True)
18.
19.    IF machines IS NULL THEN
20.      SET machines TO self.scheduler.machines
21.    END IF
22.
23.    APPEND a copy of job TO self.molded_job_history['before']
24.
25.    # Υπολογισμός του ιδανικού αριθμού πυρήνων
26.    SET A TO job.req
27.
28.    IF self.variance_model IS 'low' THEN
29.      SET sigma TO a random value between 0 and 1 (low variance)
30.      SET n TO moldable_model.optimal_coresLOW(A, sigma, max_cores=self.scheduler.cores)
31.      SET n TO CEIL(n)  # Prevent jobs with fractional cores from crashing
32.      SET new_dur TO moldable_model.duration_with_nLOW(job, A, n, sigma)
33.    ELSE
34.      SET sigma TO a random value between 1.01 and 10 (high variance)
35.      SET n TO moldable_model.optimal_coresHIGH(A, sigma, max_cores=self.scheduler.cores)
36.      SET n TO CEIL(n)  # Prevent jobs with fractional cores from crashing
37.      SET new_dur TO moldable_model.duration_with_nHIGH(job, A, n, sigma)
38.
39.    # Ενημέρωση του task
40.    SET job.req TO n
41.    SET job.dur TO new_dur
42.
43.    APPEND job TO self.molded_job_history['after']
44.
45.    # Αν δεν υπάρχουν μηχανήματα, δημιουργούμε ένα καινούργιο
46.    IF machines IS EMPTY THEN
47.      CALL new_machine(job)
48.      RETURN 1  # Τερματισμός προγραμματισμού
49.    END IF
50.
51.    # Κάνουμε schedule για αρχή με τον First-fit αλγόριθμο
52.    FOR EACH machine IN machines DO
```

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

```
53.          IF self.scheduler.check_fit(machine, job) THEN
54.             CALL machine.add_job(job)
55.             RETURN 1  # Scheduling is done
56.          END IF
57.       END FOR
58.
59.       # Αν δεν βρούμε διαθέσιμο server, φτιάχνουμε καινούργιο
60.       IF opennew THEN
61.          CALL new_machine(job)
62.          RETURN 1  # Scheduling is done
63.       END IF
64.
65.       RETURN 0  # Could not schedule the job
66.    END METHOD
67. END CLASS
68.
```

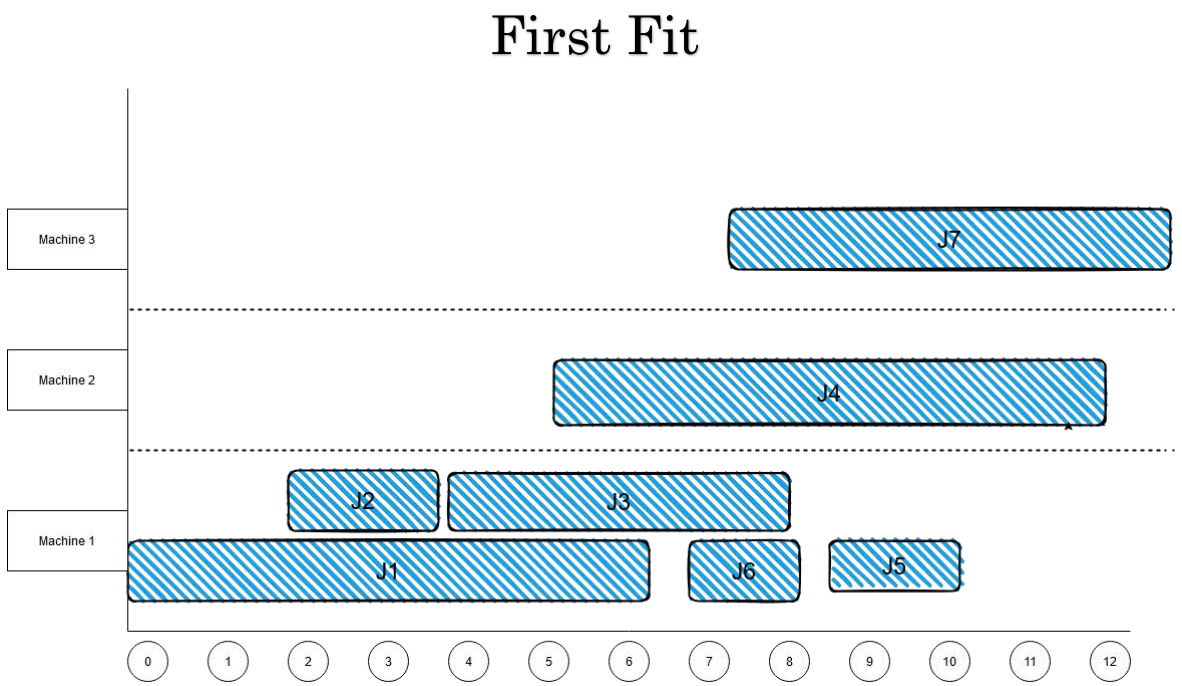
Mold_plus_FF Algorithm

```
1. CLASS FirstFit
2.    METHOD INIT(scheduler)
3.       INPUT: scheduler
4.       SET self.scheduler TO scheduler
5.    END METHOD
6.
7.    METHOD new_machine(job)
8.       INPUT: job # Δημιουργία καινούργιου server
9.       CREATE new_machine WITH (self.scheduler.cores, job.ar)
10.       CALL new_machine.add_job(job)  # Εισαγωγή του job στο μηχάνημα
11.       CALL self.scheduler.add_machine(new_machine)
12.    END METHOD
13.
14.    METHOD pack(job, machines=None, opennew=True)
15.       INPUT: job, machines (optional), opennew (default True)
16.
17.       IF machines IS NULL THEN
18.          SET machines TO self.scheduler.machines
19.       END IF
20.
21.       # Αν δεν υπάρχουν servers, φτιάχνουμε καινούργιο
22.       IF machines IS EMPTY THEN
23.          CALL new_machine(job)
24.       ELSE
25.          SET handled TO FALSE
26.
27.          # Try to find the first machine that can process the job
28.          FOR EACH machine IN machines DO
29.             SET ret TO self.scheduler.check_fit(machine, job)
30.             IF ret IS TRUE THEN
31.                CALL machine.add_job(job)
32.                SET handled TO TRUE
33.                RETURN TRUE  # Job successfully scheduled
34.             END IF
35.          END FOR
36.
37.          # Κάλεσμα μέθοδού για δημιουργία νέου machine
38.          IF NOT handled AND opennew IS TRUE THEN
39.             CALL new_machine(job)
```

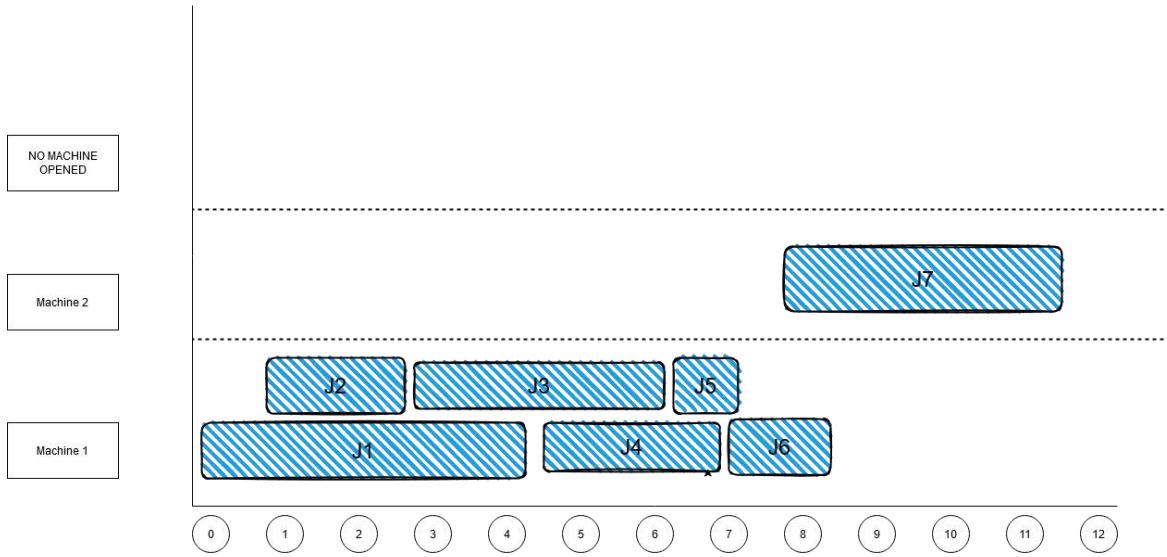
Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

```
40.          ELSE
41.              RETURN FALSE # Αν φτάσουμε εδώ, δεν μπορούσαμε να αναθέσουμε κάπου το task
42.      END IF
43.    END METHOD
44. END CLASS
45.
```

FirstFit Algorithm



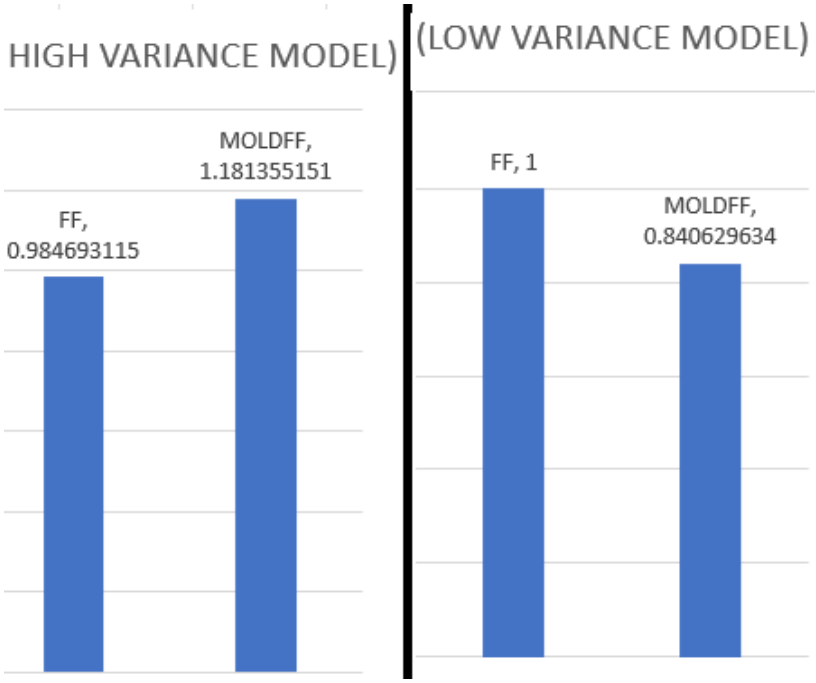
Βλέπουμε στην πράξη την ταξινόμηση που κάνει ο αρχικός μας αλγόριθμος. Για κάθε εργασία/δουλειά που του δίνεται, το ταξινομεί στο πρώτο διαθέσιμο μηχάνημα που βρίσκει, άρα θα γεμίσει όσο περισσότερο μπορεί κάθε machine πριν δημιουργήσει έναν νέο.



Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

Ο Moldable First Fit θα ξεκινήσει με την ίδια λογική, χρησιμοποιώντας την ίδια τεχνική για να ταξινομήσει την παρόν διεργασία σε ένα μηχάνημα. Αλλά έχοντας moldable tasks, και βρίσκοντας τον optimal αριθμό πυρήνων/πόρων που χρειάζεται η διεργασία, πιθανά μειώνουμε τον αριθμό μηχανών, αρά και κόστος, και συνολικό χρόνο εκτέλεσης του scheduler.

Όπως θα δούμε όμως, παρόλο που με την εφαρμογή moldable διεργασιών υπάρχει μια σημαντική βελτίωση στον FF αλγόριθμο στην low εκδοχή, άλλα και μια σχεδόν εξίσου υποβάθμισή του στην high. Η αυξημένη περιπλοκότητα των πράξεων επιτάχυνσης (speedup) μας προκαλεί καθυστέρηση στον συνολικό χρόνο εκτέλεσης σε όλα τα εξεταζόμενα workloads.



5.5 MOLDABLE SQUEEZY

Ο αλγόριθμος MoldSqueezy σχεδιάστηκε για την κατανομή φορτίων εργασιών σε ένα περιβάλλον που απαιτεί βέλτιστη χρήση των διαθέσιμων πόρων, όπως πυρήνες επεξεργαστών. Ακολουθεί μια προσαρμοστική στρατηγική κατανομής, αξιολογώντας τη βέλτιστη κατανομή πυρήνων για κάθε εργασία βάσει δύο μοντέλων μεταβλητότητας (“low” και “high variance”).

```
1. CLASS MoldSqueezy
2.     METHOD INIT(scheduler)
3.         INPUT: scheduler
4.         SET self.scheduler TO scheduler
5.         SET self.variance_model TO 'low' # μεταβλητή που αποφασίζει ποια εκδοχή θα χρησιμοποιήσουμε
6.         SET self.molded_job_history TO {'before': [], 'after': []}
7.         SET self.printed TO FALSE
8.         SET self.thresh TO NONE
9.     END METHOD
10.
```

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

```
11.   METHOD new_machine(job) # Δημιουργία καινούργιου server
12.     INPUT: job
13.     CREATE new_machine WITH (self.scheduler.cores, job.ar)
14.     CALL new_machine.add_job(job)
15.     CALL self.scheduler.add_machine(new_machine)
16.   END METHOD
17.
18.   METHOD pack(job, machines=None, opennew=True)
19.     INPUT: job, machines (optional), opennew (default TRUE)
20.
21.     ASSERT self.thresh IS NOT NONE  # Σιγουρεύει πως το threshold (το οποίο εφαρμόζει η main) είναι ρυθμισμένη πριν αρχίζει το πακετάρισμα των διεργασιών
22.
23.     IF NOT self.printed THEN
24.       # εκτύπωση threshold για debugging
25.       SET self.printed TO TRUE
26.     END IF
27.
28.     IF machines IS NULL THEN
29.       SET machines TO self.scheduler.machines
30.     END IF
31.
32.     APPEND a copy of job TO self.molded_job_history['before']
33.
34.     # Υπολογισμός του ιδανικού αριθμού πυρήνων
35.     SET A TO job.req
36.     IF self.variance_model IS 'low' THEN
37.       SET sigma TO RANDOM.uniform(0, 1)  # επιλογή ενός τυχαίου αριθμού μεταξύ [0,1]
38.       SET n TO MOLDABLE_MODEL.optimal_coresLOW(A, sigma, max_cores=self.scheduler.cores)
39.       SET n TO CEIL(n)  # Στρογγυλοποίηση για την αποφυγή fractional cores
40.       SET new_dur TO MOLDABLE_MODEL.duration_with_nLOW(job, A, n, sigma)
41.     ELSE
42.       SET sigma TO RANDOM.uniform(1.01, 10)  # Random number > 1
43.       SET n TO MOLDABLE_MODEL.optimal_coresHIGH(A, sigma, max_cores=self.scheduler.cores)
44.       SET n TO CEIL(n)  # Στρογγυλοποίηση για την αποφυγή fractional cores
45.       SET new_dur TO MOLDABLE_MODEL.duration_with_nHIGH(job, A, n, sigma)
46.     END IF
47.
48.     # Ενημέρωση του job
49.     SET job.req TO n
50.     SET job.dur TO new_dur
51.
52.     # Εάν δεν υπάρχουν μηχανήματα, δημιουργία νέου
53.     IF machines IS EMPTY THEN
54.       CALL new_machine(job)
55.       APPEND job TO self.molded_job_history['after']
56.       RETURN 1  # Τερματισμός προγραμματισμού
57.     END IF
58.
59.     # Δοκιμή scheduling με τον First Fit αλγόριθμο
60.     FOR EACH machine IN machines DO
61.       IF self.scheduler.check_fit(machine, job) THEN
62.         CALL machine.add_job(job)
63.         APPEND job TO self.molded_job_history['after']
64.         RETURN 1  # Τέλος προγραμματισμού
65.       END IF
66.     END FOR
67.
68.
```

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

```
69.       SET original_job TO COPY(job)
70.
71.       WHILE job.req > 1 DO
72.          # μειώνουμε τους απαιτούμενους πυρήνες κατά ένα την φορά
73.          DECREMENT job.req BY 1
74.          IF self.variance_model IS 'low' THEN
75.             SET job.dur TO MOLDABLE_MODEL.duration_with_nLOW(job, A, job.req, sigma)
76.          ELSE
77.             SET job.dur TO MOLDABLE_MODEL.duration_with_nHIGH(job, A, job.req, sigma)
78.          END IF
79.
80.          # Δοκιμή ταξινόμησής τις τροποποιημένης διεργασίας
81.          FOR EACH machine IN machines DO
82.             SET max_duration TO MAXIMUM(job.dur FOR job IN machine.jobs, DEFAULT=0)
83.             SET min_duration TO MINIMUM(job.dur FOR job IN machine.jobs, DEFAULT=0)
84.             SET fits TO self.scheduler.check_fit(machine, job)
85.
86.             # Ελέγχουμε αν χωράει η διεργασία σύμφωνα με τα εφαρμοσμένα μας όρια-thresholds
87.             IF fits AND job.dur <= (max_duration * self.thresh) AND job.dur >= (min_duration / self.thresh)
THEN
88.                CALL machine.add_job(job)
89.                APPEND job TO self.molded_job_history['after']
90.                RETURN 1  # Τέλος προγραμματισμού
91.             END IF
92.          END FOR
93.       END WHILE
94.
95.       # Επαναφορά τις αρχικής διεργασίας και άνοιγμα νέου μηχανήματος
96.       SET job TO original_job
97.       IF opennew THEN
98.          CALL new_machine(job)
99.          APPEND job TO self.molded_job_history['after']
100.         RETURN 1  # Τερματισμός προγραμματισμού
101.      END IF
102.
103.      RETURN 0  # Αν φτάσουμε εδώ, δεν μπορούσαμε να αναθέσουμε κάπου το task
104.   END METHOD
105. END CLASS
106.
```

MoldSqueezy Algorithm

Κατά την είσοδο μιας νέας εργασίας, ο αλγόριθμος υπολογίζει τον βέλτιστο αριθμό πυρήνων και τη διάρκεια εκτέλεσης, χρησιμοποιώντας τα μοντέλα μεταβλητότητας. Ανάλογα με την επιλογή του μοντέλου ("low" ή "high variance"), καθορίζεται μια τιμή σφάλματος (sigma) που επηρεάζει τον αριθμό των απαιτούμενων πυρήνων.

Ο αριθμός των βέλτιστων πυρήνων υπολογίζεται μέσω των συναρτήσεων optimal_coresLOW και optimal_coresHIGH, ενώ η διάρκεια εκτέλεσης προσαρμόζεται αντίστοιχα με τις συναρτήσεις duration_with_nLOW και duration_with_nHIGH. Αυτές οι συναρτήσεις υπολογίζουν την απόδοση της εργασίας και αναπροσαρμόζουν τη διάρκειά της, ανάλογα με τον αριθμό των πυρήνων που θα της αποδοθούν.

Ο αλγόριθμος ελέγχει πρώτα αν υπάρχει διαθέσιμη μηχανή που μπορεί να φιλοξενήσει την εργασία, κάνοντας χρήση της στρατηγικής “first fit”. Εάν βρεθεί κατάλληλη μηχανή, η εργασία προστίθεται σε αυτήν. Σε περίπτωση που δεν υπάρχει διαθέσιμη μηχανή για την εργασία, ο αλγόριθμος μειώνει τον αριθμό των απαιτούμενων

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

πυρήνων για να κάνει τη δουλειά πιο εύκολη στη διαχείριση και επαναϋπολογίζει τη διάρκειά της. Αυτή η μείωση γίνεται βήμα-βήμα, ελέγχοντας κάθε φορά αν η εργασία μπορεί να τοποθετηθεί σε κάποια από τις υπάρχουσες μηχανές.

Εάν καμία υπάρχουσα μηχανή δεν μπορεί να φιλοξενήσει την εργασία, ο αλγόριθμος δημιουργεί νέα μηχανή και αναθέτει την εργασία σε αυτήν. Η νέα μηχανή προστίθεται στον κατάλογο διαθέσιμων μηχανών του συστήματος. Τέλος, ο αλγόριθμος διατηρεί ένα ιστορικό των εργασιών που έχουν κατανεμηθεί, τόσο πριν όσο και μετά την επεξεργασία, καταγράφοντας τις πληροφορίες για ανάλυση και βελτιστοποίηση.

Ο MoldSqueezy παρέχει μια αποτελεσματική και ευέλικτη λύση για τη διαχείριση εργασιών, διασφαλίζοντας την αποδοτική αξιοποίηση των πυρήνων επεξεργασίας στο σύστημα, ενώ παράλληλα προσαρμόζει τη διάρκεια εκτέλεσης των εργασιών ανάλογα με τη διαθεσιμότητα των πόρων.

ΚΕΦΑΛΑΙΟ 6 ΠΡΟΣΟΜΟΙΩΤΗΣ

Σε αυτό το κεφάλαιο περιγράφεται η υλοποίηση του προσομοιωτή μας για προγραμματισμό εργασιών για κατανεμημένα υπολογιστικά περιβάλλοντα. Ο προσομοιωτής έχει σχεδιαστεί με σκοπό να αξιολογεί και να βελτιστοποιεί τη χρήση των διαθέσιμων υπολογιστικών πόρων, ενώ υποστηρίζει την πειραματική ανάλυση διαφορετικών αλγορίθμων κατανομής εργασιών. Μέσω της γενικής και επεκτάσιμης αρχιτεκτονικής του, παρέχεται η δυνατότητα προσαρμογής του σε ποικίλα περιβάλλοντα, συμπεριλαμβανομένων αυτών με περιορισμένους πόρους.

6.1 ΣΤΟΧΟΙ ΤΟΥ SCHEDULER

Ο προγραμματιστής εργασιών που αναπτύχθηκε στο πλαίσιο του εν λόγω έργου έχει σχεδιαστεί με στόχο την άκρως αποδοτική διαχείριση των διαθέσιμων υπολογιστικών πόρων, προάγοντας την οπτιμιστική κατανομή των εργασιών στους πυρήνες των επεξεργαστών. Ιδιαίτερη έμφαση δίδεται στη δυνατότητα λειτουργίας σε περιβάλλοντα edge computing, όπου οι πόροι είναι περιορισμένοι, διασφαλίζοντας έτσι την απαραίτητη ευελιξία. Ο προγραμματιστής επιτρέπει την επιλογή από διαφορετικούς αλγορίθμους κατανομής και την επιλογή πόρων, είτε αυτοί αφορούν cores servers είτε ακόμα και τη διάσπαση απαιτητικών διεργασιών σε πιο απλές, προσφέροντας έτσι υψηλή προσαρμοστικότητα σε ποικίλες απαιτήσεις. Εξάλλου, η λειτουργία του βασίζεται στη συνεχή παρακολούθηση της κατάστασης των πόρων, την ευφυή κατανομή των εργασιών και την αξιολόγηση της συνολικής απασχόλησης των μηχανών.

Η διαδικασία του προγραμματιστή ακολουθεί ένα σαφές και οργανωμένο μοτίβο. Αρχικά, αναγνωρίζονται και φορτώνονται οι εργασίες στο σύστημα, έπειτα, ένας επιλεγμένος αλγόριθμος αναλαμβάνει την κατανομή των εργασιών στους διαθέσιμους πόρους, λαμβάνοντας υπόψη τις ανάγκες κάθε εργασίας ως προς την υπολογιστική ισχύ και το χρόνο εκτέλεσης. Ο scheduler, εν τω μεταξύ, παρακολουθεί συνεχώς την κατάσταση του συστήματος, καταγράφοντας κρίσιμα δεδομένα όπως το φορτίο των μηχανών και την αποδοτικότητά τους, τα οποία αξιοποιούνται για την εκτίμηση της απόδοσης και την ανίχνευση πιθανών προβλημάτων.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

Το σύστημα αυτό συνιστά έναν εξαιρετικά αποδοτικό και επεκτάσιμο προγραμματιστή εργασιών, ιδανικό για κατανεμημένα συστήματα, δίνοντας τη δυνατότητα κατανομής εργασιών σε περιβάλλοντα με περιορισμένους πόρους. Η υποστήριξη πολλών αλγορίθμων, μέσω της modular δομής του προσομοιωτή, επιτρέπει την απόλυτη προσαρμοστικότητα του συστήματος στις ιδιαίτερες απαιτήσεις κάθε εφαρμογής. Παρά την ευρεία του χρήση, η αρχιτεκτονική του προσομοιωτή παραμένει γενικής φύσεως, χωρίς να επικεντρώνεται σε ένα συγκεκριμένο υπολογιστικό περιβάλλον, και παρόλο που δεν περιορίζεται σε τέτοιου είδους εφαρμογές, μπορεί εύκολα να προσαρμοστεί για να προσομοιώσει περιορισμένα περιβάλλοντα, προσφέροντας, για παράδειγμα, τη δυνατότητα προσθήκης ανώτατου ορίου στον αριθμό των διαθέσιμων μηχανών. Αυτή η ευχέρεια καθιστά τον προσομοιωτή εξαιρετικά χρήσιμο για τη δοκιμή συνθηκών που προσομοιώνουν το edge computing, όπου οι πόροι είναι περιορισμένοι και απαιτούν προσεκτική διαχείριση.

Συνοψίζοντας, ο προσομοιωτής παρέχει ένα εξαιρετικά ευέλικτο και δυναμικό πλαίσιο για την αξιολόγηση και ανάπτυξη αλγορίθμων προγραμματισμού, καθιστώντας τον αναντικατάστατο εργαλείο για ερευνητικούς σκοπούς και εφαρμογές σε κατανεμημένα συστήματα.

6.2 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ

Ο προσομοιωτής μας χωρίζεται σε αρκετά αρχεία για την πιο κατανοητική ανάγνωσή του, έχοντας διαφορετικές λειτουργίες χωρισμένες, και για πιο εύκολη τροποποίηση του για διαφορετικούς αλγόριθμους - απαιτήσεις.

```
1. CLASS job
2.     CONSTRUCTOR(ar, fin, dur, req)
3.         SET self.ar = ar
4.         SET self.fin = fin
5.         SET self.dur = dur
6.         SET self.req = req
7.         SET self.timestamp_beg = NULL
8.         SET self.timestamp_end = NULL
9.
10.    METHOD __str__() RETURNS STRING
11.        RETURN "|A:{self.ar} D:{self.dur} R:{self.req}|"
12.
13.    METHOD __repr__() RETURNS STRING
14.        RETURN self.__str__()
15.
16.    METHOD info()
17.        PRINT "Arrived at:", self.ar, "I will be done at:", self.fin,
18.            "Duration:", self.dur, "I need", self.req, "cores!"
19.
20.    METHOD info_time()
21.        PRINT "Arrived:", self.timestamp_beg, "Finished:", self.timestamp_end
22. END CLASS
23. FUNCTION parse(path, len = -1, cores = 32, cluster = FALSE)
24.    SET jobs = []
25.
26.    OPEN file at path for reading as f
27.
28.    SET cnt = 0
```

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

```
29.    FOR EACH line IN f
30.       // Remove trailing characters and split by tab
31.       SET job_arr = line.rstrip().split('\t')
32.
33.       // Convert each element to integer
34.       FOR EACH index, value IN job_arr
35.          SET job_arr[index] = INT(value)
36.
37.       /*
38.          EXAMPLE:
39.          job : [1860905, -1, 13, 16]
40.          - arrives at 1860905
41.          - does not wait (-1)
42.          - lasts 13 seconds
43.          - requires 16 cores
44.       */
45.
46.       // Check if the job fits the core requirements and has valid duration
47.       IF cores > job_arr[3] AND job_arr[3] > 0 AND job_arr[2] > 0
48.          // Calculate finish time
49.          IF job_arr[1] <= 0
50.             SET fin = job_arr[0] + job_arr[2]
51.          ELSE
52.             SET job_arr[0] = job_arr[0] + job_arr[1]
53.             SET fin = job_arr[0] + job_arr[2]
54.
55.          // Append job(s) to the list
56.          IF NOT cluster
57.             APPEND job(job_arr[0], fin, job_arr[2], job_arr[3]) TO jobs
58.          ELSE
59.             FOR _ IN RANGE(job_arr[3])
60.                APPEND job(job_arr[0], fin, job_arr[2], 1) TO jobs
61.
62.          // Increment job count
63.          SET cnt = cnt + 1
64.
65.          // Stop if the desired number of jobs is reached
66.          IF cnt == len
67.             BREAK
68.
69.    // Sort jobs by arrival time
70.    SET jobs = SORT(jobs, KEY = x.ar)
71.
72.    CLOSE file f
73.
74.    RETURN jobs
75. END FUNCTION
76.
```

my_parser

Ξεκινώντας με τον συντακτικό μας αναλυτή (parser), ο οποίος εξάγει και κατασκευάζει τις διεργασίες που θα χρησιμοποιήσουμε διαβάζοντας ‘τες από ένα αρχείο, της επιλογής μας, μετατρέποντας τα δεδομένα του σε “αντικείμενα” για ανάλυση και την ταξινόμησή τους. Η βάση του κώδικα είναι η class **Job**, που περιγράφει τις βασικές παραμέτρους μιας εργασίας όπως τον χρόνο άφιξης της εργασίας στο σύστημα (**self.ar**) και χρόνο τερματισμού (**self.fin**), τη διάρκεια που απαιτείται για την ολοκλήρωσή της (**self.dur**) , καθώς και τις απαιτήσεις σε πόρους που χρειάζεται (**self.req**). Αυτά τα χαρακτηριστικά

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

είναι η βάση της αναπαράστασης της εργασίας, επιτρέποντας τη λεπτομερή παρακολούθηση του κύκλου ζωής της. Συνεχίζοντας, η class περιλαμβάνει δύο προαιρετικές μεταβλητές**, timestamp_beg** και **timestamp_end**, οι οποίες μπορούν να χρησιμοποιηθούν για την παρακολούθηση των ακριβών χρόνων που ξεκινά η εργασία και ολοκληρώνεται η εκτέλεσή της. Αυτά τα προαιρετικά πεδία είναι ιδιαίτερα χρήσιμα για ανάλυση απόδοσης. Για να διευκολύνει τον εντοπισμό σφαλμάτων και την καταγραφή, η κλάση εφαρμόζει μεθόδους αναπαράστασης συμβολοσειρών **(__str__ και __repr__)** για να παρέχει συνοπτικές αλλά ενημερωτικές περιλήψεις των βασικών ιδιοτήτων μιας εργασίας. Τα functions **info** και **info_time** επιτρέπουν τη λεπτομερή αναφορά των χαρακτηριστικών μιας εργασίας, καθιστώντας εύκολη την εξαγωγή και την οπτικοποίηση πληροφοριών εργασίας όταν απαιτείται.

Ο parser είναι το βασικό στοιχείο που διαβάζει τα δεδομένα εργασίας από ένα αρχείο εισόδου και τα μετατρέπει σε δομημένα αντικείμενα. Ξεκινά ανοίγοντας το καθορισμένο αρχείο σε λειτουργία κειμένου και επαναλαμβάνεται σε κάθε γραμμή. Κάθε γραμμή στο αρχείο αντιπροσωπεύει μια εργασία, με τα χαρακτηριστικά της να κωδικοποιούνται ως τιμές διαχωρισμένες με tabs (/t). Η συνάρτηση επεξεργάζεται αυτές τις γραμμές αφαιρώντας πρώτα τυχόν χαρακτήρες που ακολουθούν και στη συνέχεια διαχωρίζοντας τη γραμμή στα στοιχεία της. Οι αναλυμένες τιμές μετατρέπονται σε ακέραιους για να διευκολυνθεί ο περαιτέρω υπολογισμός. Μετά την ανάλυση όλων των γραμμών, η προκύπτουσα λίστα εργασιών ταξινομείται με βάση τους χρόνους άφιξής τους για να διασφαλιστεί ότι οι εργασίες επεξεργάζονται με χρονολογική σειρά και, η συνάρτηση επιστρέφει την πλέον ταξινομημένη λίστα εργασιών, έτοιμη να χρησιμοποιηθεί για προσομοίωση ή ανάλυση.

```
1. DEFINE interval AS NAMED TUPLE WITH FIELDS:
2.     - beg (begin time)
3.     - fin (finish time)
4.     - cores (number of cores)
5. CLASS interval_maker
6.     CONSTRUCTOR()
7.         SET self.points = []  // List of dictionaries describing points
8.
9.     METHOD remove(time)
10.        // Removes all points before the given time
11.        WHILE self.points IS NOT EMPTY
12.            IF self.points[0]['time'] < time
13.                REMOVE first element from self.points
14.            ELSE
15.                BREAK
16.
17.    METHOD ret_index(strt, time, add_cores)
18.        // Starting from strt, find the position for the given time
19.        // and update cores for points in between
20.        SET i = strt
21.        WHILE i < LENGTH OF self.points
22.            IF time <= self.points[i]['time']
23.                RETURN i, time == self.points[i]['time'], FALSE
24.            ELSE
25.                INCREMENT self.points[i]['cores'] BY add_cores
26.                INCREMENT i BY 1
27.
28.        RETURN i, FALSE, TRUE
```

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

```
29.
30.    METHOD add(job)
31.        IF self.points IS EMPTY
32.            // First point: add arrival and finish points
33.            APPEND {'time': job.ar, 'cores': job.req} TO self.points
34.            APPEND {'time': job.fin, 'cores': 0} TO self.points
35.        ELSE
36.            // Find index for the start point (job.ar)
37.            SET strt, eq, _ = CALL ret_index(strt=0, time=job.ar, add_cores=0)
38.            IF NOT eq
39.                // Insert new start point
40.                SET cores = self.points[strt - 1]['cores']
41.                INSERT {'time': job.ar, 'cores': cores} INTO self.points AT INDEX strt
42.
43.            // Find index for the end point (job.fin) while adding cores to in-between points
44.            SET fin, eq, finished = CALL ret_index(strt=strt, time=job.fin, add_cores=job.req)
45.            IF NOT eq
46.                // Insert new end point
47.                IF finished
48.                    SET cores = 0
49.                ELSE
50.                    SET cores = self.points[fin - 1]['cores'] - job.req
51.
52.                INSERT {'time': job.fin, 'cores': cores} INTO self.points AT INDEX fin
53.
54.    METHOD to_list()
55.        // Convert points to a list of intervals
56.        SET l = []
57.        FOR i FROM 0 TO LENGTH OF self.points - 2
58.            SET s = self.points[i]['time']
59.            SET f = self.points[i + 1]['time']
60.            SET c = self.points[i]['cores']
61.            APPEND interval(s, f, c) TO l
62.        RETURN l
63. END CLASS
64.
```

interval_maker

Η class **interval maker** χρησιμεύει ως εργαλείο για τη δυναμική παρακολούθηση της κατανομής και της απελευθέρωσης υπολογιστικών πόρων με την πάροδο του χρόνου. Χρησιμοποιώντας μια λίστα σημείων με χρονική σήμανση και αξιοποιώντας την κλάση namedtuple για την αναπαράσταση διαστημάτων, αυτή η κλάση παρέχει μηχανισμούς για προσθήκη, αφαίρεση και ανάκτηση δομημένων χρονικών διαστημάτων. Η λίστα σημείων αποθηκεύει μια ακολουθία από points οπού το καθένα αντιπροσωπεύει ένα συγκεκριμένο χρονικό σημείο και περιέχει δύο κλειδιά: ώρα, που υποδεικνύει τη χρονική σήμανση του σημείου, και πυρήνες, που αντιπροσωπεύουν τον αθροιστικό αριθμό των υπολογιστικών πυρήνων που χρησιμοποιούνται εκείνη τη στιγμή. Αυτή η δομή επιτρέπει μια δυναμική και αποτελεσματική αναπαράσταση της χρήσης πόρων καθώς προστίθενται ή αφαιρούνται εργασίες. Ο σκοπός της **ret_index** μεθόδου είναι η απόφαση εισαγωγής ή ενημέρωσης σημείων στη λίστα σημείων. Ξεκινώντας από ένα δεδομένο **index** (strt), αναζητά τη σωστή θέση μιας καθορισμένης τιμής χρόνου. Εάν ο χρόνος ταιριάζει με ένα ήδη υπάρχον σημείο, η μέθοδος επιστρέφει το index και ένα **flag** που υποδεικνύει την ισότητα τους (True). Εάν ο χρόνος είναι μικρότερος από το επόμενο σημείο, επιστρέφει τον κατάλληλο index εισαγωγής. Επιπλέον, καθώς επαναλαμβάνει τα σημεία, ενημερώνει την τιμή των

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

πυρήνων σε κάθε σημείο προσθέτοντας μια καθορισμένη τιμή **add_cores**. Αυτό διασφαλίζει ότι τυχόν ενδιάμεσα σημεία αντικατοπτρίζουν τη αλλαγή στη χρήση πόρων που προκαλείται από μια εργασία. Εάν ο χρόνος είναι εκτός εύρος των υπαρχόντων σημείων, η μέθοδος επιστρέφει τη θέση στο τέλος της λίστας, μαζί με ένα **flag** που υποδεικνύει ότι έχει φτάσει στο τέλος.

Η **add** function είναι η καρδιά της κλάσης interval_maker, υπεύθυνη για την προσθήκη μιας νέας εργασίας στη γραμμή χρόνου και την ενημέρωση της λίστας σημείων ανάλογα. Εάν η λίστα σημείων είναι κενή, την αρχικοποιεί με δύο σημεία: ένα στην ώρα άφιξης της εργασίας **(job.ar)** με τους απαιτούμενους πυρήνες **(job.req)** και ένα άλλο στο χρόνο λήξης της εργασίας **(job.fin)** με μηδενικούς πυρήνες. Για τις επόμενες εργασίες, η μέθοδος καθορίζει πρώτα τη θέση του χρόνου άφιξης (job.ar) χρησιμοποιώντας το **ret_index**. Εάν η ώρα άφιξης δεν υπάρχει ήδη ως σημείο, εισάγει ένα νέο σημείο στην κατάλληλη θέση, διατηρώντας τη συνέχεια της γραμμής χρόνου.

Ομοίως, υπολογίζει τη θέση του χρόνου τερματισμού **(job.fin)** και εισάγει ένα σημείο εάν χρειάζεται. Κατά τη διάρκεια αυτής της διαδικασίας, η μέθοδος ενημερώνει τις τιμές πυρήνων για όλα τα ενδιάμεσα σημεία, διασφαλίζοντας ότι το χρονοδιάγραμμα αντικατοπτρίζει με ακρίβεια την τρέχουσα κατάσταση χρήσης πόρων.

```
1. CLASS machine
2.    CONSTRUCTOR(cores, time, indx = -1)
3.       SET self.indx = indx
4.       SET self.imaker = NEW interval_maker()
5.       SET self.cores = cores
6.       SET self.jobs = []
7.       SET self.intervals = []
8.       SET self.last_update = time
9.       SET self.opened = time
10.      SET self.shut_down = FALSE
11.      SET self.type = NULL
12.      SET self.t = NULL
13.      SET self.jobs_hist = []
14.
15.   METHOD info()
16.      PRINT "I have", LENGTH OF self.jobs, "jobs:"
17.      FOR EACH j IN self.jobs
18.         CALL j.info()
19.      PRINT "I have", LENGTH OF self.intervals, "intervals:"
20.      FOR EACH p IN self.intervals
21.         PRINT p
22.      SET bt = self.intervals[-1].fin - self.opened
23.      PRINT "I opened at:", self.opened, "Closed at:", self.intervals[-1].fin, "My bt is:", bt
24.
25.   METHOD find_min_max_rem(time)
26.      // Calculate remaining time for each job and return min, max
27.      SET rem_dur = [(x.fin - time) FOR EACH x IN self.jobs]
28.      SET rem_dur = [x FOR EACH x IN rem_dur IF x > 0]
29.      RETURN MIN(rem_dur), MAX(rem_dur)
30.
31.   METHOD update(time)
32.      SET max_time = self.intervals[-1].fin
33.      // Check if the machine should shut down
34.      IF time >= max_time
35.         SET time = max_time
```

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

```
36.          SET self.shut_down = TRUE
37.       // Calculate busy time
38.       SET busy_time = time - self.last_update
39.       SET self.last_update = time
40.
41.       IF NOT self.shut_down
42.          // Delete finished jobs
43.          WHILE self.jobs IS NOT EMPTY
44.             IF self.jobs[0].fin < time
45.                REMOVE first element FROM self.jobs
46.             ELSE
47.                BREAK
48.
49.       RETURN busy_time
50.
51.    METHOD dynamic(job)
52.       CALL self.imaker.add(job)
53.       CALL self.imaker.remove(job.ar)
54.       SET self.intervals = CALL self.imaker.to_list()
55.
56.    METHOD add_job(job)
57.       // Add new job to the array
58.       APPEND job TO self.jobs
59.       // Create intervals
60.       CALL self.dynamic(job)  // O(n)
61. END CLASS
62.
```

machine

Στο επόμενο αρχείο έχουμε την κατασκευή των μηχανών μας. Ορίζοντας την class machine, παράγουμε την λειτουργικότητα που είναι απαραίτητη για την παρακολούθηση, τη διαχείριση και την προσομοίωση της λειτουργίας μιας μηχανής σε περιβάλλον προγραμματισμού ή προσομοίωσης φόρτου εργασίας. Ο σχεδιασμός ενσωματώνει βασικά χαρακτηριστικά, μεθόδους για ενημερώσεις κατάστασης και μηχανισμούς για τη διαχείριση των διαστημάτων εκτέλεσης εργασιών.

Η function **info** χρησιμεύει ως διαγνωστικό εργαλείο, εκτυπώνοντας λεπτομερείς πληροφορίες σχετικά με την τρέχουσα κατάσταση του μηχανήματος. Απαριθμεί τον αριθμό των ενεργών εργασιών, τον αριθμό των διαστημάτων εκτέλεσης και βασικές μετρήσεις, όπως η ώρα που άνοιξε το μηχάνημα, η τελευταία φορά που έκλεισε και ο συνολικός χρόνος απασχόλησης. Η **find_min_max_rem** υπολογίζει τους υπόλοιπους χρόνους εκτέλεσης για τις ενεργές εργασίες του μηχανήματος. Αφαιρεί τον τρέχοντα χρόνο (time) από τον χρόνο τερματισμού (fin) κάθε εργασίας, φιλτράρει τυχόν ολοκληρωμένες εργασίες (με μη θετικούς υπολειπόμενους χρόνους) και επιστρέφει τους ελάχιστους και μέγιστους χρόνους που απομένουν.

Η **update**, όπως λέει και η ονομασία της, ενημερώνει την κατάσταση του μηχανήματος κατά την εκτέλεση του προσομοιωτή, καθορίζοντας πρώτα εάν το μηχάνημα πρέπει να τερματιστεί συγκρίνοντας τον δεδομένο χρόνο με τον χρόνο λήξης του τελευταίου διαστήματος εκτέλεσης. Εάν ο τρέχων χρόνος υπερβαίνει ή ταιριάζει με αυτόν τον χρόνο τερματισμού, το μηχάνημα επισημαίνεται ως απενεργοποιημένο. Υπολογίζει το makespan του μηχανήματος από την τελευταία ενημέρωση, αντιπροσωπεύοντας την περίοδο κατά την οποία το μηχάνημα εκτελούσε ενεργά εργασίες. Η update “καθαρίζει" επίσης τις

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

ολοκληρωμένες εργασίες επαναλαμβάνοντας τη λίστα εργασιών και αφαιρώντας τυχόν εργασίες των οποίων ο χρόνος λήξης προηγείται της τρέχουσας ώρας. Τέλος, επιστρέφει τελικά τον υπολογιζόμενο χρόνο κατειλημμένου, ο οποίος μπορεί να χρησιμοποιηθεί για ανάλυση απόδοσης ή για αποφάσεις προγραμματισμού. Η **add_job** επιτρέπει τον προγραμματισμό νέων εργασιών στο μηχάνημα, ενώ η **dynamic** αξιοποιεί το interval_maker για τη δυναμική διαχείριση των διαστημάτων εκτέλεσης του μηχανήματος.

```
1.    FUNCTION get_thrs(alg, cores)
2.   IF alg == 0
3.      RETURN -1
4.   ELSE IF alg == 'MET_AD'
5.      OPEN file "thresholds/actual_{cores}.txt" FOR READING AS f
6.   ELSE IF alg == 'MET_RD'
7.      OPEN file "thresholds/REM1_{cores}.txt" FOR READING AS f
8.
9.   SET thresholds = []
10.   FOR EACH line IN f
11.      SPLIT line BY "\t" AND STRIP TRAILING CHARACTERS
12.      APPEND SECOND ELEMENT (converted to FLOAT) TO thresholds
13.   CLOSE file f
14.
15.   RETURN thresholds
16. END FUNCTION
17. FUNCTION split_time(batch_dur, jobs)
18.   SET batches = []
19.   SET batch = []
20.   SET end_of_batch = batch_dur
21.
22.   FOR EACH job IN jobs
23.      IF job.ar > end_of_batch
24.         WHILE end_of_batch > batch_dur
25.            INCREMENT end_of_batch BY batch_dur
26.         APPEND batch TO batches
27.         SET batch = []
28.
29.      APPEND job TO batch
30.   APPEND batch TO batches
31.
32.   RETURN batches
33. END FUNCTION
34. FUNCTION get_sparsity(batch)
35.   IF LENGTH OF batch == 1
36.      RETURN 1
37.   ELSE
38.      SET intervals = NEW interval_maker()
39.      SET job_dur = 0
40.      SET interval_span = 0
41.
42.      FOR EACH j IN batch
43.         INCREMENT job_dur BY j.dur
44.         CALL intervals.add(j)
45.
46.      // Filter out intervals with 0 cores (idle time)
47.      SET i_list = [x FOR EACH x IN CALL intervals.to_list() IF x.cores != 0]
48.
49.      FOR EACH i IN i_list
```

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

```
50.          INCREMENT interval_span BY (i.fin - i.beg)
51.       SET s = interval_span / job_dur
52.       RETURN s
53. END FUNCTION
54. FUNCTION parse_workload(set, cluster, cores)
55.    SET data = LIST FILES IN "data/*"
56.    SORT data BY INTEGER VALUE OF THE SUBSTRING FROM INDEX 5 TO 7
57.    RETURN CALL parse(data[set], cores=cores, cluster=cluster)
58. END FUNCTION
```

functions

Το functions.py είναι μια συλλογή από βοηθητικές λειτουργίες που έχουν σχεδιαστεί για να βελτιώνουν και να διαχειρίζονται την προσομοίωση του φόρτου εργασίας προγραμματισμού εργασιών. Αυτές οι συναρτήσεις εκτελούν διάφορες εργασίες, όπως ανάλυση δεδομένων φόρτου εργασίας, προσδιορισμό τιμών κατωφλίου για συγκεκριμένους αλγόριθμους, διαχωρισμό εργασιών σε παρτίδες που βασίζονται στο χρόνο, υπολογισμός αραιότητας φόρτου εργασίας και ανάκτηση αναλυμένων συνόλων δεδομένων. Στα πλαίσια του έργου μας, χρησιμοποιούμε και θα εξετάσουμε μόνο μια συνάρτηση.

Η **parse_workload(set, cluister, cores)** ανακτά και αναλύει δεδομένα φόρτου εργασίας από ένα προκαθορισμένο σύνολο δεδομένων. Χρησιμοποιεί τη μονάδα **glob** για να εντοπίσει όλα τα αρχεία μέσα στα δεδομένα/κατάλογο και να τα ταξινομήσει με βάση ένα αριθμητικό αναγνωριστικό που εξάγεται από το όνομα του αρχείου. Αφού προσδιορίσει το σύνολο δεδομένων που αντιστοιχεί στη δεδομένη παράμετρο συνόλου, καλεί τη συνάρτηση ανάλυσης με πρόσθετες παραμέτρους (cluster και πυρήνες) για να επεξεργαστεί το αρχείο και να δημιουργήσει μια λίστα αντικειμένων εργασίας.

```
1. from Environment.machine import machine
2. import functions as f
3. import statistics as st
4.
5.
6. class ExceededCapacityError(Exception):
7.     pass
8.
9.
10. class Scheduler:
11.     def __init__(self, wl, cores, alg, is_edge=False, avail_servers=4, **kwargs):
12.         self.cores = cores
13.         self.alg = alg
14.         self.machines = []
15.         self.total_bt = 0
16.         self.total_machines = 0
17.
18.         # EDGE COMPUTING CASE
19.         self.is_edge = is_edge
20.         self.avail_servers = avail_servers
21.         self.max_concurrent_machines = 0
22.         self.jobs_rejected = 0
23.         self.busy_time_rejected = 0
24.
25.         # Select the appropriate scheduling algorithm
26.         match alg:
27.             case 'FF':
28.                 from Environment.Algorithms.FirstFit import FirstFit
29.                 self.algorithm = FirstFit(self)
```

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

```
30.         case 'BF':
31.             from Environment.Algorithms.BestFit import BestFit
32.             self.algorithm = BestFit(self)
33.         case 'Moldable':
34.             from Environment.Algorithms.MoldableParallelTasks import MPT
35.             self.algorithm = MPT(self)
36.         case 'Greedy':
37.             from Environment.Algorithms.Greedy import MaxCoreMPT
38.             self.algorithm = MaxCoreMPT(self)
39.         case 'MoldSqueezy':
40.             from Environment.Algorithms.MoldSqueezy import MoldSqueezy
41.             self.algorithm = MoldSqueezy(self)
42.         case "MoldFF":
43.             from Environment.Algorithms.Mold_plus_FF import MoldFF
44.             self.algorithm = MoldFF(self)
45.         case _:
46.             raise ValueError(f"'{alg}' is not a supported algorithm.")
47.
48.     @property
49.     def state(self):
50.         avg_jobs = [len(x.jobs) for x in self.machines]
51.         if not avg_jobs:
52.             return [len(self.machines), 0]
53.         return [len(self.machines), st.mean(avg_jobs)]
54.
55.     @property
56.     def load(self):
57.         return sum(job.dur for m in self.machines for job in m.jobs)
58.
59.     @property
60.     def state_bt(self):
61.         return sum(max(x.fin for x in m.jobs) - m.jobs[0].ar for m in self.machines)
62.
63.     def reset(self):
64.         self.jobs = []
65.         self.machines = []
66.         self.total_bt = 0
67.         self.total_machines = 0
68.         # EDGE COMPUTING
69.         self.max_concurrent_machines = 0
70.         self.jobs_rejected = 0
71.         self.busy_time_rejected = 0
72.
73.     def info(self):
74.         print("---------------------------------")
75.         for indx, m in enumerate(self.machines):
76.             print("=========================================")
77.             print(f"Machine: {indx}")
78.             m.info()
79.             print("=========================================")
80.         print("---------------------------------")
81.
82.     def add_machine(self, machine: machine):
83.         if self.is_edge and len(self.machines) + 1 > self.avail_servers:
84.             self.jobs_rejected += 1
85.             self.busy_time_rejected += machine.jobs[0].dur
86.             return -1
87.
88.         self.total_machines += 1
```

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

```
89.         machine.indx = self.total_machines
90.         self.machines.append(machine)
91.         self.max_concurrent_machines = max(self.max_concurrent_machines, len(self.machines))
92.
93.     def update_all(self, time):
94.         for m in self.machines:
95.             self.total_bt += m.update(time)
96.
97.         self.machines = [x for x in self.machines if not x.shut_down]
98.
99.     def fin(self):
100.         for m in self.machines:
101.             time = m.intervals[-1].fin
102.             self.total_bt += m.update(time)
103.
104.     def finish_no_update(self):
105.         cur_fin_bt = self.total_bt
106.         for m in self.machines:
107.             time = m.intervals[-1].fin
108.             cur_fin_bt += time - m.last_update
109.         return cur_fin_bt
110.
111.     def check_fit(self, machine, job, ret_leftover=False):
112.         leftover = 0
113.         cnt_intrmd_ivs = 0
114.         fits = True
115.
116.         for i in machine.intervals:
117.             if i.beg >= job.fin:
118.                 break
119.             if i.fin > job.ar:
120.                 leftover += self.cores - (job.req + i.cores)
121.                 cnt_intrmd_ivs += 1
122.                 if (job.req + i.cores) > self.cores:
123.                     fits = False
124.
125.         if ret_leftover:
126.             return fits, leftover / cnt_intrmd_ivs
127.         return fits
128.
129.     def get_max_req_interval(self, machine, job, ret_leftover=False):
130.         max_ival_req = 0
131.
132.         for i in machine.intervals:
133.             if i.beg >= job.fin:
134.                 break
135.             if i.fin > job.ar and i.cores > max_ival_req:
136.                 max_ival_req = i.cores
137.
138.         return max_ival_req
139.
140.     def run(self, jobs):
141.         for j in jobs:
142.             self.update_all(j.ar)
143.             self.algorithm.pack(j)
144.         self.fin()
145.         return self.total_bt
146.
147.     def run_many(self, jobs=[], fin=False):
```

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

```
148.        if fin:
149.            self.fin()
150.            return self.total_bt
151.
152.        for j in jobs:
153.            self.update_all(j.ar)
154.            self.algorithm.pack(j)
155.        return self.total_bt
```

scheduler

Η κλάση Scheduler έχει σχεδιαστεί για τη διαχείριση υπολογιστικών πόρων και τον προγραμματισμό του φόρτου εργασίας αποτελεσματικά. Η κλάση προετοιμάζεται με παραμέτρους όπως ο αριθμός υπολογιστικών πυρήνων (cores), ο αλγόριθμος προγραμματισμού (alg) και πρόσθετες επιλογές όπως edge computing configuration (is_edge) και οι διαθέσιμοι servers (avail_servers).

Αυτή η προετοιμασία παρέχει μια ευέλικτη ρύθμιση, επιτρέποντας στον scheduler να προσαρμοστεί σε διαφορετικές απαιτήσεις. Το property state επιστρέφει τον αριθμό των ενεργών μηχανημάτων και τον μέσο αριθμό εργασιών ανά μηχάνημα, ενώ το load υπολογίζει το συνολικό φόρτο εργασίας σε όλα τα μηχανήματα αθροίζοντας τις διάρκειες των ενεργών εργασιών. Μια άλλη ιδιότητα (property), το state_bt, καθορίζει τον συνολικό χρόνο απασχολήσεως αναλύοντας τα ενεργά διαστήματα για κάθε μηχανή. Αυτές οι ιδιότητες προσφέρουν ένα στιγμιότυπο της απόδοσης του συστήματος και της χρήσης πόρων ανά πάσα στιγμή.

Για τη διαχείριση των λειτουργιών του, ο scheduler παρέχει διάφορες βασικές μεθόδους. Η μέθοδος reset αρχικοποιεί εκ νέου τον scheduler, διαγράφοντας όλες τις εργασίες, τις μηχανές και τις σχετικές μετρήσεις. Η μέθοδος info εξάγει μια λεπτομερή περίληψη της τρέχουσας κατάστασης του χρονοπρογραμματιστή, συμπεριλαμβανομένης της κατάστασης κάθε μηχανής και των εργασιών του. Η μέθοδος add_machine προσθέτει ένα νέο μηχάνημα στον χρονοπρογραμματιστή, αλλά στο edge computing, επιβάλλει όρια χωρητικότητας απορρίπτοντας μηχανές που θα υπερέβαιναν τους διαθέσιμους διακομιστές. Αυτή η μέθοδος παρακολουθεί επίσης μετρήσεις όπως οι απορρίψεις εργασιών και τον μέγιστο αριθμό ταυτόχρονων μηχανών, εξασφαλίζοντας την “υπακοή” περιορισμών του edge-computing.

Η ενημέρωση της κατάστασης όλων των μηχανημάτων γίνεται με τη μέθοδο update_all, η οποία συγχρονίζει τις καταστάσεις του μηχανήματος με βάση τον παρεχόμενο χρόνο. Αυτό περιλαμβάνει τον υπολογισμό του makespan του κάθε μηχανήματος, την κατάργηση ολοκληρωμένων διεργασιών και την απόφαση εάν ένα μηχάνημα θα πρέπει να τερματιστεί. Τα μηχανήματα που δεν είναι πλέον ενεργά αφαιρούνται από το σύστημα. Ομοίως η fin μέθοδος ολοκληρώνει τη διαδικασία προγραμματισμού, ενημερώνοντας όλα τα μηχανήματα στα τελευταία τους διαστήματα, διασφαλίζοντας έναν ακριβή υπολογισμό του συνολικού χρόνου απασχόλησης. Η μέθοδος finish_no_update παρέχει μια εναλλακτική προσέγγιση με τον υπολογισμό του makespan χωρίς τροποποίηση των καταστάσεων του μηχανήματος, η οποία είναι χρήσιμη για ανάλυση της απόδοσης χωρίς αλλαγή της τρέχουσας διαμόρφωσης του scheduler.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

Το πρόγραμμα περιλαμβάνει μεθόδους για την προσαρμογή διεργασιών και την κατανομή πόρων. Η μέθοδος check_fit αξιολογεί εάν μια εργασία μπορεί να ανατεθεί σε ένα μηχάνημα χωρίς υπέρβαση της χωρητικότητάς του, επιστρέφοντας προαιρετικά τη μέση υπολειπόμενη χωρητικότητα για επικαλυπτόμενα διαστήματα. Η μέθοδος get_max_req_interval προσδιορίζει τη μέγιστη χρήση πόρων σε διαστήματα που επικαλύπτονται με μια δεδομένη εργασία, βοηθώντας στη βελτιστοποίηση της κατανομής πόρων. Αυτές οι μέθοδοι διασφαλίζουν αποτελεσματικό προγραμματισμό διενεργώντας λεπτομερείς ελέγχους σχετικά με τη διαθεσιμότητα και τη χρήση των πόρων.

Τέλος, η run μέθοδος επεξεργάζεται μια λίστα εργασιών, ενημερώνοντας τα μηχανήματα την ώρα άφιξης κάθε εργασίας και αναθέτοντας την πραγματική τοποθέτηση εργασίας στη μέθοδο πακέτου του επιλεγμένου αλγόριθμου. Μόλις όλες οι εργασίες υποβληθούν για επεξεργασία, η μέθοδος οριστικοποιεί το χρονοδιάγραμμα. Για workloads με σταδιακή αύξηση των δεδομένων τους η μέθοδος run_many προσφέρει παρόμοια λειτουργικότητα, επιτρέποντας στον προγραμματιστή να επεξεργάζεται εργασίες σε παρτίδες ή να οριστικοποιεί το χρονοδιάγραμμα όπως απαιτείται.

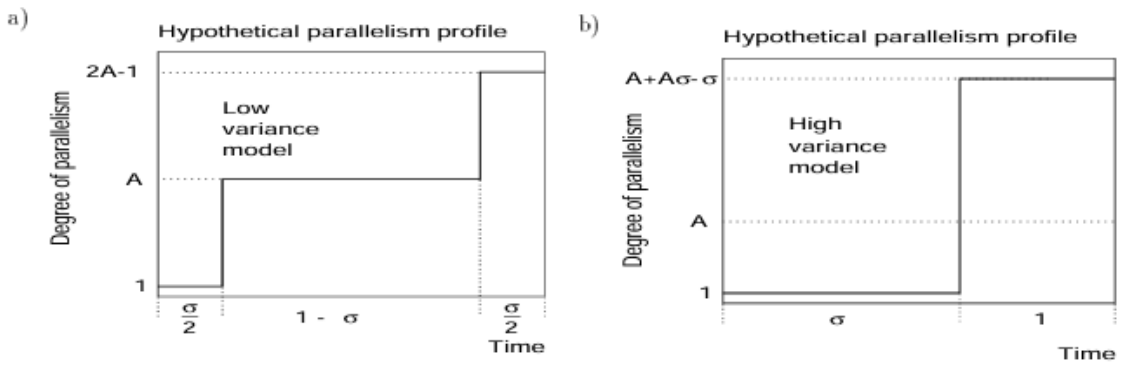
```
def duration_with_nLOW(job, A, n, sigma):
    """
    Calculate the new duration of a job given a specific number of cores (n)
    under a low-variance model.
    """
    initial_speedup = speedupLOW(n=job.req, A=A, sigma=sigma)
    new_speedup = speedupLOW(n=n, A=A, sigma=sigma)
    speedup_ratio = initial_speedup / new_speedup
    return job.dur * speedup_ratio


def duration_with_nHIGH(job, A, n, sigma):
    """
    Calculate the new duration of a job given a specific number of cores (n)
    under a high-variance model.
    """
    initial_speedup = speedupHIGH(n=job.req, A=A, sigma=sigma)
    new_speedup = speedupHIGH(n=n, A=A, sigma=sigma)
    speedup_ratio = initial_speedup / new_speedup
    return job.dur * speedup_ratio


def speedupLOW(n, A, sigma):
    """
    Calculate the speedup of a job under a low-variance model.
    """
    if 1 <= n <= A:
        return (A * n) / (A + (sigma / 2) * (n - 1))
    elif A <= n < 2 * A - 1:
        return (A * n) / (sigma * (A - 0.5) + n * (1 - sigma / 2))
    else:
        return A


def speedupHIGH(n, A, sigma):
    """
```

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

```
    Calculate the speedup of a job under a high-variance model.
    """
    if 1 <= n <= (A + (A * sigma) - sigma):
        return ((n * A) * (sigma + 1)) / (sigma * (n + A - 1) + A)
    elif n >= (A + (A * sigma) - sigma):
        return A


def optimal_coresLOW(A, sigma, max_cores):
    """
    Calculate the optimal number of cores for a job under a low-variance model.
    """
    n_star = (sigma * (A - 0.5)) / (1 - sigma / 2)
    return min(n_star, max_cores)  # Ensure we don't exceed the core capacity


def optimal_coresHIGH(A, sigma, max_cores):
    """
    Calculate the optimal number of cores for a job under a high-variance model.
    """
    n_star = ((A * (sigma + 1)) - sigma) / sigma
    return min(n_star, max_cores)  # Ensure we don't exceed the core capacity

```

moldable_task_model

Το αρχείο moldable_task_model παρέχει ένα σύνολο λειτουργιών με στόχο την βελτιστοποίηση της διάρκειας και απόδοσης των διεργασιών με βάση την κατανομή πόρων. Αυτό το επιτυγχάνει αξιοποιώντας μοντέλα επιτάχυνσης για χαμηλούς και υψηλούς φόρτους εργασίας, μαζί με αλγόριθμους για τον προσδιορισμό του βέλτιστου αριθμού υπολογιστικών πυρήνων για μια δεδομένη εργασία. Οι συναρτήσεις duration_with_nLOW και duration_with_nHIGH υπολογίζουν την προσαρμοσμένη διάρκεια μιας εργασίας όταν αλλάζει ο αριθμός των ανατεθειμένων πυρήνων. Οι υπολογισμοί βασίζονται στα μοντέλα επιτάχυνσης speedupLOW και speedupHIGH (βασισμένα στο paper του Allen B. Downey), τα οποία καθορίζουν την κλιμάκωση της απόδοσης των εργασιών σε συνθήκες χαμηλής και υψηλής διακύμανσης, αντίστοιχα. Σε αυτές τις λειτουργίες, η αρχική επιτάχυνση (με βάση τους ζητούμενους πυρήνες της εργασίας) συγκρίνεται με την επιτάχυνση που επιτυγχάνεται με τη νέα κατανομή πυρήνα. Η αναλογία των δύο επιταχύνσεων καθορίζει τον τρόπο με τον οποίο κλιμακώνεται η διάρκεια της εργασίας. Αυτή η προσαρμογή παρέχει μια εκτίμηση του χρόνου εκτέλεσης των tasks υπό διαφορετικές συνθήκες πόρων.



Προφίλ παραλληλισμού για το low και high variance (βιβλιογραφία 8)

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

Οι συναρτήσεις optimal_coresLOW και optimal_coresHIGH καθορίζουν τον βέλτιστο αριθμό πυρήνων που πρέπει να εκχωρηθούν σε μια εργασία, ανάλογα με τη διακύμανση του φόρτου εργασίας. Αυτές οι συναρτήσεις υπολογίζουν το "γόνατο" των αντίστοιχων καμπυλών επιτάχυνσης, το οποίο αντιπροσωπεύει το σημείο φθίνουσας απόδοσης. Στο optimal_coresLOW, ο υπολογισμός προσδιορίζει το βέλτιστο n_star για φόρτους εργασίας χαμηλής διακύμανσης, διασφαλίζοντας ότι η επιστρεφόμενη τιμή δεν υπερβαίνει τους μέγιστους διαθέσιμους πυρήνες (max_cores). Ομοίως, το optimal_coresHIGH υπολογίζει τη βέλτιστη κατανομή πυρήνα για φόρτους εργασίας υψηλής διακύμανσης, εφαρμόζοντας τον ίδιο περιορισμό άνω ορίου.

Συμπερασματικά, ο προσομοιωτής μας αναδεικνύει μια εξαιρετικά δομημένη και επεκτάσιμη αρχιτεκτονική, σχεδιασμένη να προσαρμόζεται σε διαφορετικές απαιτήσεις και περιβάλλοντα. Με τη χρήση επιμέρους κλάσεων και λειτουργιών επιτυγχάνεται μια λεπτομερής και δυναμική προσομοίωση της διαχείρισης φόρτου εργασίας και κατανομής πόρων.

Ο συνδυασμός των βασικών στοιχείων, όπως οι αλγόριθμοι χρονοπρογραμματισμού, η παρακολούθηση διαστημάτων, η ανάλυση δεδομένων και οι μέθοδοι βελτιστοποίησης, προσφέρει ευελιξία και ακρίβεια. Οι λειτουργίες αυτές όχι μόνο επιτρέπουν την προσομοίωση διαφορετικών σεναρίων, αλλά και παρέχουν τις κατάλληλες βάσεις για την ανάλυση της αποδοτικότητας και τη σύγκριση διαφορετικών προσεγγίσεων.

ΚΕΦΑΛΑΙΟ 7 EXPERIMENTAL EVALUATION

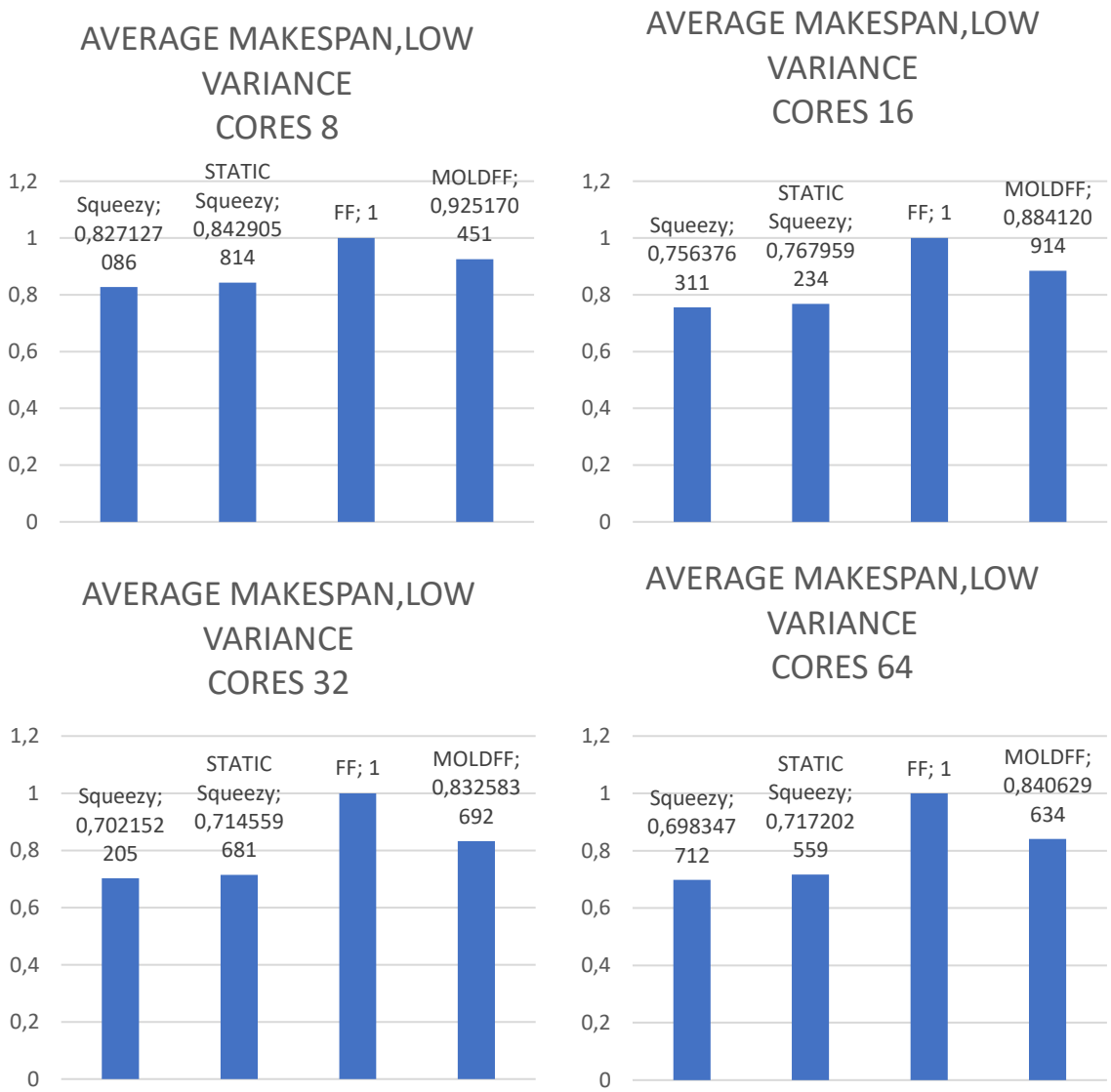
Αυτό το κεφάλαιο παρουσιάζει την πειραματική αξιολόγηση διαφόρων αλγορίθμων χρονοπρογραμματισμού για διεργασίες, εξετάζοντας στους moldable αλγόριθμους τα low και high variances τους. Οι αλγόριθμοι που αξιολογήθηκαν περιλαμβάνουν τους FirstFit (FF), MoldableFirstFit (MoldFF) για την άμεση παρατήρηση βελτιστοποίησης τον αποτελεσμάτων που μπορεί να προσφέρουν τα moldable tasks και του MoldSqueezy με στατικά thresholds (2x και /2), αλλά και με το βελτιστιποιόμενο (optimal) threshold μεταξύ του 1,0 και 10,0 . Η αξιολόγηση στοχεύει στον εντοπισμό του πιο αποτελεσματικού αλγόριθμου σε ένα εύρος μετρήσεων πυρήνων μηχανής (8, 16, 32, 64) και σεναρίων φόρτου εργασίας. Τα πειράματα επικεντρώνονται στην μέτρηση του makespan, που θα μας παρέχει πληροφορίες για τη σχετική απόδοση αυτών των αλγορίθμων. Αναλύοντας τα αποτελέσματα, στοχεύουμε να καθορίσουμε τις συνθήκες κάτω από τις οποίες υπερέχει κάθε αλγόριθμος και να παρέχουμε συστάσεις για την πρακτική εφαρμογή τους.

7.1 LOW VARIATION

Τα παραπάνω αποτελέσματα αναδεικνύουν πως ο αλγόριθμος MoldSqueezy με optimal threshold (για κάθε workload) ξεπερνάει σταθερά τους άλλους αλγόριθμους ανεξαρτήτως το αρχείο εργασίας και αναθετημένω αριθμο πηρήνων του server, επιτυχόντας μικρότερα makepan. Αυτή η ανώτερη απόδοση γίνεται χάρης στην ικανότητά του να προσαρμόζει τα thresholds, βρίσκοντας το "sweet spot" καθενός φορτίου εργασίας για την ανάθεση ορίων,

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

που ευθυγραμμίζει την κατανομή εργασιών. Σε σενάρια με υψηλότερο αριθμό πυρήνων, η ικανότητας αύτη γίνεται πιο ισχύρη στην μείωση του makespan.

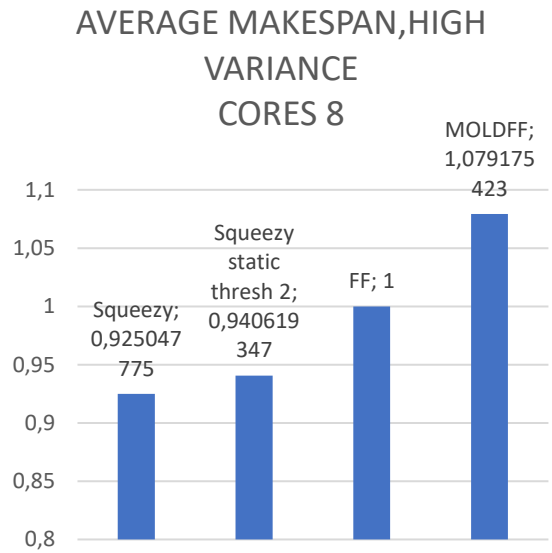


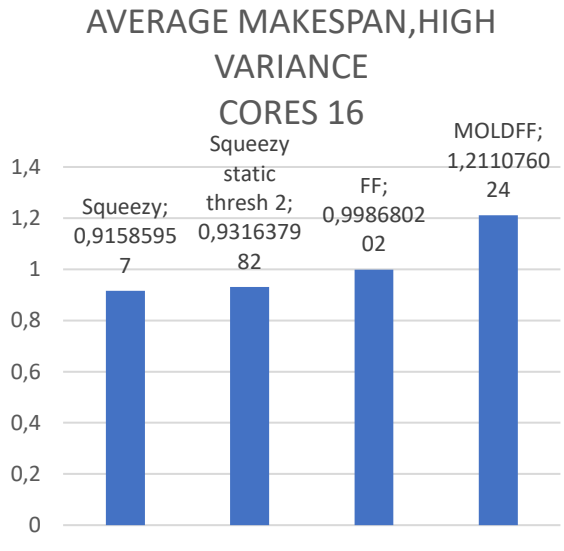
Κλειδώνοντας το threshold του MoldSqueezy στα 2, ενώ προσφέρει μέτριες βελτιώσεις σε σχέση με το FF και το MoldFF, ο αλγόριθμος δεν μπόρει να προσαρμοστεί σε συγκεκριμένα χαρακτηριστικά των workloads μας. Αποτυγχάνει να λάβει υπόψη τις διακυμάνσεις στη διάρκεια της εργασίας γεγονός που μπορεί να οδηγήσει σε μη optimal κατανομή πόρων και αυξημένους χρόνους αδράνειας. Αντίθετα, με την εύρεση του optimal threshold σε κάθε workload, ο αλγόριθμος μπορεί να "προσαρμοστεί" σε αυτές στις δυνατότητες κάθε φορτίου, βρίσκοντας πιο κατάλληλα και "ακριβές" όρια να θέσει. Αναλύοντσα τα γραφήματα, παρατηρούμε ότι καθώς αυξάνεται ο αριθμός των πυρήνων,το χάσμα απόδοσης μεταξύ του MoldSqueezy και των First Fit αλγορίθμων γίνεται όλο ένα και μεγαλύτερο, με των MoldFF να έχει μεγαλύτερο κατα μέσο όρο makespan στο πείραμα των 64 πηρήνων σε σχέση με όταν έχει 32. Αυτή η τάση δείχνει ότι τα υψηλότερα επίπεδα παραλληλισμού ενισχύουν τα πλεονεκτήματα του διαμορφώσιμου προγραμματισμού εργασιών με βελτιστοποιημένα thresholds. Το αυξανόμενο χάσμα πιθανότατα οφείλεται στην ικανότητα του MoldSqueezy να κατανέμει αποτελεσματικά

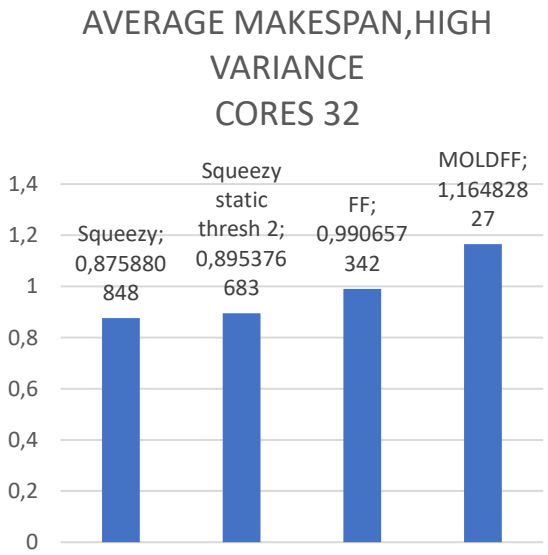
Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

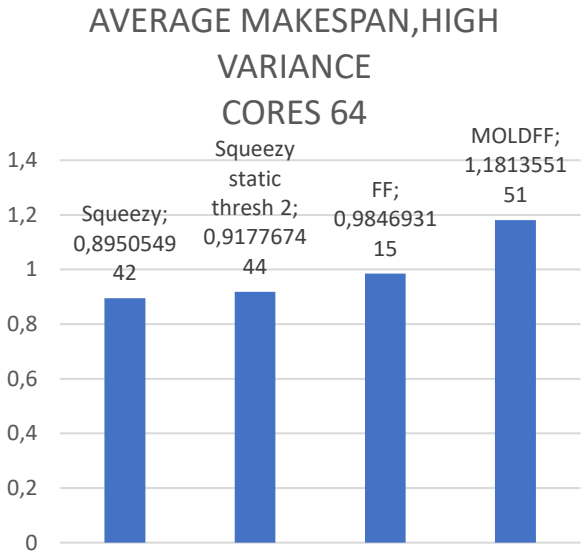
τον φόρτο εργασίας σε έναν αυξανόμενο αριθμό πυρήνων, μειώνοντας τα σημεία συμφόρησης και μεγιστοποιώντας τη χρήση των πόρων.

7.2 HIGH VARIATION









Όπως είδαμε και την low έκδοση, παρατηρούμε ότι τα αποτελέσματα δείχνουν πως ο MoldSqueezy ξεπερνά σταθερά τα FF και MoldFF σε όλα τα σενάρια. Το στατικό threshold *2 /2 αποδεικνύει εξαιρετικά αποτελεσματικό σε φόρτους εργασίας υψηλής διακύμανσης. Ενώ τα optimal thresholds παραμένουν ανταγωνιστικά εξισορροπώντας την ευαισθησία των εργασιών και την κατανομή πόρων, η ελαφρά υπο-απόδοση τους σε σύγκριση με τα στατικά κατώφλια υπογραμμίζει την απλότητα και την αποτελεσματικότητα των τελευταίων στον χειρισμό φόρτου εργασίας υψηλής διακύμανσης.

Το high variation μας αποδίδει χειρότερα αποτελέσματα για όλους τους αλγόριθμους σε σύγκριση με τα αποτελέσματα του low (Εκτός από τον FF. Ως ένας απλός μη moldable αλγόριθμος, παραμένει ανεπηρέαστος επειδή δεν βασίζεται στην προσαρμογή των μεγεθών

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

ή των ορίων των εργασιών, καθιστώντας το ανεπηρέαστο στις διακυμάνσεις του φόρτου εργασίας). Εισάγει πιο περίπλοκες πράξεις και υπολογισμούς που επηρεάζουν την αποδοτικότητα του προγραμματισμού σε όλους τους επηρεαζόμενους αλγόριθμους. Η εξίσωση speedupHIGH δίνει έμφαση σε ένα ευρύτερο φάσμα από φθίνουσες αποδόσεις, γεγονός που αυξάνει τα σημεία bottleneck στην εξάρτηση. Ο υπολογισμός optimal_coresHIGH βγάζει ως αποτέλεσμα μια λιγότερο ακριβή κατανομή πυρήνων λόγω αυτών των ευρύτερων παραλλαγών, με αποτέλεσμα μη optimal αποτελέσματα. Αντίθετα, η low εκδοχη επιτρέπει την αυστηρότερη ευθυγράμμιση των πόρων λόγω της πιο προβλέψιμης κλίμακας που καταγράφεται από τις συνάρτησης speedupLOW και optimal_coresLOW. Αυτή η προβλεψιμότητα μειώνει τα σημεία συμφόρησης και εξασφαλίζει πιο αποτελεσματικό scheduling. Για τον MoldSqueezy, τα optimal thresholds προσαρμόζονται καλά στα διαφορετικά μεγέθη και ποσότητα διεργασιών, αλλά η αυξημένη μεταβλητότητα προκαλεί την ευθυγράμμιση των πόρων, οδηγώντας συχνά σε αναποτελεσματικότητα στη χρήση των cores.

7.3 EXPERIMENT CONCLUSIONS

Αναλύοντας τα παραγόμενα αποτελέσματα, συμπεραίνουμε τα πλεονεκτήματα του MoldSqueezy στον προγραμματισμό moldable task υπό διάφορες συνθήκες φορτιού εργασίας. Δίνοντας του την ικανότητα να βρίσκει τα πιο βέλτιστα threshold για κάθε workload του επιτρέπει να επιτυγχάνει ανώτερες μετρήσεις απόδοσης και συνολικής αξίας, ιδιαίτερα σε σενάρια χαμηλής διακύμανσης προσφέροντας απλότητα και αποτελεσματικότητα. Σε μελλοντική έρευνα θα μπορούσαμε να βελτιστοποιήσουμε ακόμα περισσότερο τα αποτελέσματα του δίνοντας του ένα μεγαλύτερο range από thresholds, καθώς και να ξε-συνδέσουμε τα δύο όρια (αντί να έχουμε δηλαδή μια κοινή μεταβλητή για το άνω και κάτω όριο, ο αλγόριθμος να έβρισκε το optimal ανώτατο και optimal κατώτατο π.χ. διάρκεια<= max_dur*6.5 και διάρκεια>=min_dur/3).

Οι βασική αλγόριθμοι όπως ο FF και ο MoldFF, λόγο ότι είναι πιο απλή στην υλοποίηση τους, ξεπερνιούνται σε αυξανόμενο ρυθμό από τις moldable προσεγγίσεις όσο αυξάνεται ο αριθμός πυρήνων και η πολυπλοκότητα του φόρτου εργασίας. Στην high έκδοση κιόλας βλέπουμε πώς η πρόσθεση moldable tasks στον First Fit χειροτερεύει το τελικό makespan! Σε μελλοντική εργασία-έρευνα θα μπορούσαν να διερευνήσουν υβριδικές στρατηγικές που συνδυάζουν την προσαρμοστικότητα της αναζήτησης βέλτιστων ορίων με την αποτελεσματικότητα των στατικών προσεγγίσεων για περαιτέρω βελτίωση της απόδοσης.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

ΚΕΦΑΛΑΙΟ 8

8.1 ΣΥΜΠΕΡΑΣΜΑΤΑ

Κατά την έρευνα και ανάπτυξη του προσομοιωτή και αλγόριθμων μας, διερευνήσαμε τα αποτελέσματα του moldable scheduling στον παράλληλο υπολογισμό (parallel computing), συγκρίνοντας τον με τις στατικές μεθόδους χρονοπρογραμματισμού υπό διάφορα workloads. Εξετάζοντας διάφορες όριο-ανατεθειμένες στρατηγικές, με σκοπό την ταξινόμηση παρομοίων διεργασιών στα ίδια μηχανήματα, αξιολογήσαμε την επίδραση του δυναμικού έναντι του στατικού χρονοπρογραμματισμού στον συνολικό χρόνο εκτέλεσης του αλγορίθμου, σε διάφορες διαμορφώσεις πυρήνων και παραλλαγές του φόρτου εργασίας.

Η πειραματική μας προσέγγιση αποκάλυψε ότι ο διαμορφώσιμος χρονοπρογραμματισμός -ιδιαίτερα όταν χρησιμοποιούνται “βέλτιστα” καθορισμένα όρια- ξεπερνάει σταθερά τις στατικές μεθόδους. Τα ευρήματα υποδηλώνουν ότι καθώς αυξάνεται η διαθεσιμότητα των πυρήνων, η Squeezy προσέγγιση μας αποδίδει μεγαλύτερες μειώσεις στο makespan σε σύγκριση με στατικούς αλγορίθμους όπως ο First-Fit (FF) και την moldable εκδοχή του (MOLDFF). Αυτή η πρόοδος είναι ιδιαίτερα αισθητή σε σενάρια με φόρτους εργασίας υψηλής διαφοράς, όπου οι στατικές τεχνικές κατανομής δυσκολεύονται να προσαρμοστούν αποτελεσματικά στην ποικιλομορφία του workload μας.

Η διαφορά μεταξύ στατικών και δυναμικών ορίων αποδείχθηκε καθοριστική για τα συμπεράσματά μας. Τα στατικά όρια επιβάλλουν άκαμπτους περιορισμούς που ενδέχεται να μην αντιστοιχούν στα βέλτιστα επίπεδα παραλληλισμού του φόρτου εργασίας, με αποτέλεσμα την μη αποδοτική χρήση των πόρων. Αντίθετα, τα δυναμικά όρια παρέχουν αυξημένη ευελιξία, εξασφαλίζοντας ότι οι εργασίες ταξινομούνται “ομαδικά” σε μηχανές, και, διασφαλίζοντας ότι οι εργασίες λαμβάνουν βέλτιστο αριθμό πυρήνων με βάση τους δεδομένους περιορισμούς του συστήματος. Αυτή η προσαρμοστικότητα είναι απαραίτητη για τη μεγιστοποίηση της παράλληλης απόδοσης, ιδίως σε περιβάλλοντα που υπόκεινται σε μεταβλητό φόρτο εργασίας.

8.2 ΜΕΛΛΟΝΤΙΚΕΣ ΕΡΓΑΣΊΕΣ

Παρόλο που η παρούσα έρευνα προσφέρει σημαντικές γνώσεις σχετικά με την ανάλυση και αποτελεσματική βελτιστοποίηση του interval scheduling προβλήματος, υπάρχει πάντα το περιθώριο για περαιτέρω έρευνα και επιπλέον μελλοντική ανάλυση.

- **Διερεύνηση υβριδικών τεχνικών προγραμματισμού**: Η συγχώνευση των μεθόδων moldable και στατικών task θα μπορούσε να προσφέρει μεγαλύτερη προσαρμοστικότητα, ιδίως σε πολύ δυναμικά περιβάλλοντα.
- **Βελτίωση της δυναμικής επιλογής κατωφλίου**: Η δημιουργία αλγορίθμων που προσαρμόζουν ξεχωριστά τα κατώτατα και ανώτατα όρια σύμφωνα με τα πρότυπα φόρτου εργασίας σε πραγματικό χρόνο και για κάθε task, θα μπορούσε να ενισχύσει περαιτέρω την απόδοση του χρονοπρογραμματισμού.
- **Επέκταση σε ετερογενείς αρχιτεκτονικές**: Τα σύγχρονα υπολογιστικά συστήματα περιλαμβάνουν συχνά ετερογενείς μονάδες επεξεργασίας (π.χ. την GPU του συστήματος μας). Η επέκταση του moldable scheduling σε τέτοια περιβάλλοντα

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

θα μπορούσε να οδηγήσει σε πρόσθετες βελτιώσεις επιδόσεων, με ένα πολύ θετικό παράδειγμα να είναι τα μοντέρνα μοντέλα AI που αναπτύσσονται και εξελίσσονται πλέον καθημερινά.

Συνοψίζοντας, η πτυχιακή μας υπογραμμίζει τη σημασία των στρατηγικών προσαρμόσιμου χρονοπρογραμματισμού στον παράλληλο υπολογισμό. Ο moldable scheduling αλγόριθμος, ιδίως όταν συνδυάζεται με δυναμικό thresholding, προσφέρει μια πολλά υποσχόμενη μέθοδο για τη βελτιστοποίηση της εκτέλεσης εργασιών, την ελαχιστοποίηση του makespan και την ενίσχυση της αποδοτικότητας των πόρων σε περιβάλλοντα με πολλούς πυρήνες, και multi-core περιβάλλοντα.

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Α. Ευστρατιάδης "Θεμελιώδεις έννοιες βελτιστοποίησης και κλασικές μαθηματικές μέθοδοι"
2. P. Oikonomou, N. Tziritas, T. Loukopoulos, G. Theodoropoulos, M. Hanai and Samee U. Khan "Online Algorithms for the Interval Scheduling Problem in the Cloud: Affinity Pair Threshold Based Approaches"
3. Y. Li, X. Tang and W. Cai "Dynamic Bin Packing for On-Demand Cloud Resource Allocation"
4. Y. Mao, X. Chen and X. Li "Max–Min Task Scheduling Algorithm for Load Balance in Cloud Computing"
5. D. Ye, Danny Z. Chen & G. Zhang "Online scheduling of moldable parallel tasks"
6. W. Cirne and F. Berman "A model for moldable supercomputer jobs"
7. A. Pujiyanta and F. Novianto "Job Scheduling on Grid Computing Using First Fit, Best Fit, and Worst Fit"
8. Allen B. Downey "**A Model for Speedup of Parallel Programs**"
9. H. Sun, R. Elghazi, A. Gainaru, G. Aupy and P. Raghavan. "Scheduling Parallel Tasks under Multiple Resources: List Scheduling vs. Pack Scheduling"
10. X. Tang,Y. Li, R Ren, and W. Cai "On First Fit Bin Packing for Online Cloud Server Allocation"

Institutional Repository - Library & Information Centre - University of Thessaly
04/09/2026 05:28:59 EEST - 216.73.216.250