ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Αλγόριθμοι χρονοδρομολόγησης και τεχνικές οπτικοποίησης για τη διαχείριση ροής διαδικασιών στο υπολογιστικό νέφος.

Παππάς Νικόλαος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Νικόλαος Τζιρίτας Επίκουρος Καθηγητής

ΣΥΝΕΠΙΒΛΕΠΩΝ
(εφόσον υπάρχει)

Νικόλαος Τζιρίτας Επίκουρος Καθηγητής

Λαμία 21/07 έτος 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Αλγόριθμοι χρονοδρομολόγησης και τεχνικές οπτικοποίησης για τη διαχείριση ροής διαδικασιών στο υπολογιστικό νέφος.

Παππάς Νικόλαος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Νικόλαος Τζιρίτας Επίκουρος Καθηγητής

ΣΥΝΕΠΙΒΛΕΠΩΝ
(εφόσον υπάρχει)

Νικόλαος Τζιρίτας Επίκουρος Καθηγητής

Λαμία 21/07 έτος 2022

UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

Workflow scheduling algorithms and visualization techniques over the Cloud.

Pappas Nikolaos

FINAL THESIS

ADVISOR
Tziritas Nikolaos
Assistant Professor
Lamia 21/07 year 2022

«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: 21/07/2022

Ο – Η Δηλ.

(1) «Όποιος 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.»

ΕΥΧΑΡΙΣΤΙΕΣ

Έχοντας τελειώσει τη συγγραφή της πτυχιακής εργασίας μου θα ήθελα να ευχαριστήσω κάποια πρόσωπα που υπήρξαν δίπλα μου το τελευταίο διάστημα, αλλά και καθ' όλη τη διάρκεια της μέχρι τώρα πορείας μου. Αρχικά, θέλω να ευχαριστήσω τον επιβλέποντα καθηγητή της εργασίας μου, τον Επίκουρο Καθηγητή Νικόλαο Τζιρίτα ο οποίος με εμπιστεύτηκε και με στήριξε να βγάλω εις πέρας την παρούσα εργασία με ένα τόσο ενδιαφέρον θέμα όπως οι αλγόριθμοι χρονοδρομολόγησης. Επιπλέον ένα μεγάλο ευχαριστώ, στον καθηγητή του τμήματος Παναγιώτη Οικονόμου, αρχικά για την υπομονή του και την στήριξη του καθόλη την διάρκεια της εργασίας αλλά και για την ευκαιρία που μου έδωσε να δημοσιεύσουμε παραπλήσιο άρθρο καθώς και για την πολύτιμη βοήθεια του αναφορικά με τα τεχνικά κομμάτια της εργασίας αφού μέσα από αυτόν τον αγώνα είχα την δυνατότητα να εξελιχθώ προσωπικά αλλά και προγραμματιστικά. Ολοκληρώνοντας το στάδιο αυτό και κλείνοντας έναν κύκλο σπουδών νιώθω την ανάγκη επίσης να ευχαριστήσω όλους τους καθηγητές μου του Πανεπιστήμιου Θεσσαλίας για το χρόνο και τις γνώσεις που μου προσέφεραν τα τελευταία χρόνια. Τέλος θα ήθελα να ευχαριστήσω την οικογένεια μου και όλους τους φίλους και συγγενείς που στάθηκαν δίπλα μου σε δύσκολες στιγμές και πίστεψαν σε μένα δείχνοντας υπομονή και κατανόηση στηρίζοντας κάθε απόφαση μου.

ΠΕΡΙΛΗΨΗ

Στην παρούσα εργασία μελετάται ο προγραμματισμός σε κατανεμημένα υπολογιστικά περιβάλλοντα όπως πλέγματα και υπολογιστικά νέφη. Η έννοια της ροής διαδικασιών ή εργασιών υιοθετείται ως ένα παράδειγμα της ισχυρής ικανότητάς της να εκφράζει ένα ευρύ φάσμα εφαρμογών συμπεριλαμβανομένων επιστημονικών υπολογισμών, πολύ-επίπεδων Web εφαρμογών, μεγάλων προγραμμάτων επεξεργασίας δεδομένων. Με την ανάπτυξη της τεχνολογίας του υπολογιστικού νέφους, το πρόβλημα του προγραμματισμού ροής εργασιών στο υπολογιστικό νέφος γίνεται όλο και πιο σημαντικό ερευνητικό θέμα. Το σύστημα καλείται να βρει βέλτιστη λύση για πολλαπλές ροές εργασιών. Οι προκλήσεις των προβλημάτων που προαναφέρθηκαν πηγάζουν μεταξύ άλλων από θέματα όπως η γενικότερη NP-Complete φύση της χρονοδρομολόγησης εργασιών, η παροχή πόρων βάση προτεραιότητας της εργασίας, οι αλληλεξαρτήσεις μεταξύ εργασιών, οι πολλαπλές ροές με χρονικά περιθώρια, η δίκαια κατανομή πόρων μεταξύ διαφορετικών ροών, οι διακυμάνσεις απόδοσης στον χειρισμό αστοχιών. Κατά συνέπεια εξετάζονται επίσης μια σειρά μελετών, που είχαν ως επίκεντρο διάφορες πτυχές του προβλήματος και οι οποίες αναδείχθηκαν στη βιβλιογραφία. Επιπλέον γίνεται μια συγκριτική επισκόπηση αλγορίθμων προγραμματισμού μονής ροής εργασιών καθώς και αλγορίθμων πολλαπλών ροών. Παράλληλα με την διεξοδική έρευνα του προγραμματισμού ροής εργασιών γίνονται προτάσεις συνδυαστικών αλγορίθμων βάση της βιβλιογραφίας αλλά και νέων πιο καινοτόμων ιδεών, με σκοπό την μείωση του συνολικού χρόνου εκτέλεσης των ροών. Ακόμα με την δημιουργία διαφόρων προσαρμοσμένων εργαλείων οπτικοποίησης και αποσφαλμάτωσης γίνεται ανάλυση και εγκυροποίηση των αποτελεσμάτων. Τέλος επισημαίνονται κάποιες ερευνητικές κατευθύνσεις για μελλοντική έρευνα.

Λέξεις-κλειδιά: Ροή εργασιών, Υπολογιστικό νέφος, Πολλαπλές ροές εργασιών, Χρονοδρομολόγηση ροών εργασιών, Οπτικοποίηση, Αποσφαλμάτωση.

ABSTRACT

The present work is studying programming in distributed computing environments such as grids and computing clouds. The concept of workflow is adopted as an example of its powerful ability to express a wide range of applications including scientific computers, multi-level Web applications, large data processing programs. With the development of cloud computing technology, the problem of cloud computing workflow scheduling becomes even more important research topic. The system is challenged to find the optimal solution for multiple workflow scheduling. The challenges of the problems mentioned above stem from, among other things, from the NP-Complete nature of task scheduling, resource allocation based on task priority, task interdependencies, multiple workflows, equitable resource allocation between different workflows, performance fluctuations and failure handling. Consequently, a series of studies that have emerged in the literature are also examined, focusing on various aspects of the problems. In addition, a comparative view of single-workflow programming algorithms as well as multiple workflow scheduling algorithms are studied. Along with the extensive research of workflow scheduling, combinatorial algorithms that are based on the literature and more innovative proposals are made. With the main focus being to reduce the total execution time of the workflows. Finally, with the creation of many customized visualization and debugging tools, the data are analyzed and some research directions for future research are highlighted.

Keywords: Workflow, Cloud computing, Multiple workflows, Workflow scheduling, Visualization, Debugging

Table of Contents

ΠΕΡΙΛΗΨΗ I
ABSTRACT III

ΕΙΣΑΓΩΓΗ 2

ΒΙΒΛΙΟΓΡΑΦΙΚΗ ΕΠΙΣΚΟΠΗΣΗ 5

Μοντελοποιηση του προβληματος 2.1 5
Μοντελοποιηση του προβληματος σε περιβαλλον υπολογιστικου νεφους 2.2 7
Επισκοπηση αλγοριθμων που μελετηθηκαν για μονες ροες εργασιων 2.3 8
Επισκοπηση αλγοριθμων που μελετηθηκαν για πολλαπλες ροες εργασιων 2.4 9
Αναφορα στα εργαλεια που αναπτυχθηκαν 2.5 11

ΠΕΡΙΣΥΛΛΟΓΗ ΚΑΙ ΔΗΜΙΟΥΡΓΙΑ ΔΕΔΟΜΕΝΩΝ 13

Custom generated δεδομενα 3.1 14
Realistic δεδομενα 3.2 17
Pegasus 3.2.α 17
WfCommons 3.2.β 17
Κατηγοριες ρεαλιστικων δεδομενων 3.2.γ 18
Περισυλλογη δεδομενων 3.3 22
Εγκυροποιηση και τροποποιηση δεδομενων 3.4 24
Τροποποιηση δεδομενων 3.5 24

ΑΛΓΟΡΙΘΜΟΙ ΜΟΝΗΣ ΡΟΗΣ ΕΡΓΑΣΙΩΝ 27

HEFT 4.1 27
CPOP 4.2 32
Holes Scheduling 4.3 34
Ruin and Recreate 4.4 37
Εφαρμογες του Ruin and Recreate απο την επιστημονικη κοινοτητα 4.4.α 38
Εφαρμογες του Ruin and Recreate με μονη ροη εργασιων 4.4.β 40
Στατιστικα ευρηματα σχετικα με τους αλγοριθμους 4.5 44
Η επιρροη του πλατος των ροων στην αποδοτικοτητα των αλγοριθμων 4.5.α 45
Η επιρροη των υπολοιπων ορισματων αναφορικα με την αποδοτικοτητα των αλγοριθμων 4.5.β 50
Εφαρμογη του Ruin and Recreate στην υπαρχουσα λυση 4.5.γ 52

ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΩΝ ΡΟΩΝ ΕΡΓΑΣΙΩΝ 54

Compositions 5.1 55

C1 5.1.Α ... 56
C2 5.1.Β ... 56
C3 5.1.Γ ... 57
MULTIPLE WORKFLOW SCHEDULING 5.2 ... 58
MWCP 5.3 ... 59
ΔΙΑΦΟΡΕΤΙΚΑ ΧΡΟΝΙΚΑ ΟΡΙΣΜΑΤΑ 5.3.Α ... 60
ΠΑΡΑΔΕΙΓΜΑ ΕΦΑΡΜΟΓΗΣ ΚΑΙ Η ΕΦΑΡΜΟΓΗ ΔΙΑΦΟΡΩΝ ΤΕΧΝΙΚΩΝ BIN PACKING 5.3.Β ... 62
CTF 5.4 ... 65
ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ 5.5 ... 67

ΕΡΓΑΛΕΙΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΤΕΧΝΙΚΕΣ ΟΠΤΙΚΟΠΟΙΗΣΗΣ ... 74

ΑΝΑΠΤΥΞΗ ΚΩΔΙΚΑ ΓΙΑ ΠΕΡΙΣΥΛΛΟΓΗ ΔΕΔΟΜΕΝΩΝ 6.1 ... 74
ΑΠΟΣΦΑΛΜΑΤΩΣΗ ΧΡΟΝΟΔΙΑΓΡΑΜΜΑΤΩΝ 6.2 ... 75
ΑΠΟΣΦΑΛΜΑΤΩΣΗ ΣΕ ΕΠΙΠΕΔΟ ΕΡΓΑΣΙΑΣ 6.2.Α ... 75
ΑΠΟΣΦΑΛΜΑΤΩΣΗ ΣΕ ΕΠΙΠΕΔΟ ΥΠΟΛΟΓΙΣΤΙΚΟΥ ΠΟΡΟΥ 6.2.Β ... 75
ΑΠΟΣΦΑΛΜΑΤΩΣΗ ΣΕ ΕΠΙΠΕΔΟ ΡΟΗΣ ΕΡΓΑΣΙΩΝ 6.2.Γ ... 76
ΑΠΟΣΦΑΛΜΑΤΩΣΗ ΣΕ ΕΠΙΠΕΔΟ ΧΡΟΝΟΔΙΑΓΡΑΜΜΑΤΟΣ 6.2.Δ ... 76
ΟΠΤΙΚΟΠΟΙΗΣΗ ΚΑΙ ΑΠΟΘΗΚΕΥΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ 6.3 ... 77

ΣΥΜΠΕΡΑΣΜΑΤΑ ... 80

ΒΙΒΛΙΟΓΡΑΦΙΑ ... 82

Εισαγωγή

Η ανάγκη για υπολογιστική ισχύ και για την βέλτιστη αξιοποίηση της από τις εταιρίες, τις επιστημονικές κοινότητες, τους πιο εξειδικευμένους χρήστες, και όχι μόνο, συνεχώς αυξάνεται. Επιστημονικές κοινότητες για παράδειγμα από ποικίλους κλάδους αναζητούν όλο ένα και μεγαλύτερη υπολογιστική ισχύ για διάφορες εφαρμογές όπως την περισυλλογή δεδομένων, διεξαγωγή πειραμάτων, επεξεργασία ή και μοντελοποίηση μεγάλων όγκου δεδομένων. Ακόμα το μείζον ζήτημα για την βέλτιστη αξιοποίηση της ισχύς αυτής αυξάνεται ραγδαία ειδικότερα όταν γίνεται αναφορά για ροές εργασιών που τρέχουν σε υπολογιστικά νέφη. Μάλιστα δεν είναι λίγα τα ερευνητικά κέντρα τα οποία τρέχουν πολλαπλές ροές εργασιών ταυτόχρονα σε καθημερινή βάση. Επιπλέον πολλές από τις πολύπλοκες εργασίες που διεξάγονται είναι βαθιά αλληλεξαρτώμενες μεταξύ τους. Ενώ ο τεράστιος όγκος δεδομένων, οι μεγάλες πολυπλοκότητες των εργασιών αυτών, καταλήγουν συνολικά να έχουν τεράστιους χρόνους εκτέλεσης.

Από το τελευταίο γεγονός που αναφέρθηκε πηγάζει ίσως και το πιο κρίσιμο πρόβλημα βελτιστοποίησης το οποίο απασχολεί την επιστημονική κοινότητα. Το οποίο έχει να κάνει με την προσπάθεια μείωσης του συνολικού χρόνου εκτέλεσης όλων των εργασιών μαζί. Καθώς για τέτοιου είδους επιστημονικές ροές εργασιών κρίνεται αναγκαίο να γίνεται χρήση πολλαπλών υπολογιστικών πόρων ταυτόχρονα. Χρειάζεται να ανατεθούν δηλαδή πόροι από συστήματα όπως πλέγματα και υπολογιστικά σύννεφα των οποίων η υπολογιστική ισχύ μακράν επισκιάζει μία πιο συμβατική λύση όπως ένας επεξεργαστής/υπολογιστής. Μία ακόμα καθοριστική ιδιότητα για μία επιστημονική ροή είναι ότι διαχειρίζεται τη ροή δεδομένων. Οι εργασίες μπορεί να είναι οτιδήποτε, από σύντομες σειριακές εργασίες έως πολύ μεγάλες παράλληλες εργασίες που περιβάλλονται από μεγάλο αριθμό μικρών, σειριακών εργασιών που χρησιμοποιούνται για προ και μετά την επεξεργασία αυτών. Οι υπολογιστικοί πόροι που θα τρέξουν οι εργασίες βρίσκονται σε κάποιον προμηθευτή υπολογιστικού νέφους όπως τον AWS (πχ. Amazon Web Services), τοπικά σε κάποιο πλέγμα ή ακόμα και σε κλειστά κατανεμημένα συστήματα. Από αυτό προκύπτει το πρόβλημα της βέλτιστης ανάθεσης μιας εργασίας/πειράματος σε έναν υπολογιστικό πόρο. Η φύση του προβλήματος δυσκολεύει όταν από μία εργασία σε κάποια άλλη χρειάζεται να γίνει μεταφορά δεδομένων ή ακόμα χειρότερα να γίνει επικοινωνία με κάποιο εξωτερικό του δικτύου υπολογιστικό πόρο. Με συνδυασμό το ότι υπάρχουν αλληλεξαρτήσεις ανάμεσα στις εργασίες που επιβάλλουν συγκεκριμένους περιορισμούς, καταλήγει σαν πρόβλημα βελτιστοποίησης πολυπλοκότητας NP-Complete. Σημαντικό να αναφερθεί πως το πρόβλημα και κατ’ επέκταση οι λύσεις που προτάσσονται δεν εστιάζουν απαραίτητα μόνο σε επιστημονικές ροές εργασιών, αλλά σε γενικότερες ροές εργασιών/διαδικασιών που μπορεί να προκύψουν από άλλους ποικίλους τρόπους.

Όσο αναφορά το πρόβλημα του προγραμματισμού πολλαπλών ροών εργασίας, ένα σύνολο ροών εργασίας πρέπει να προγραμματιστεί ταυτόχρονα στους διαθέσιμους πόρους του συστήματος. Αυτό εκτός ότι αυξάνει τον βαθμό πολυπλοκότητας αυξάνονται και οι πιθανές εκδοχές και λύσεις του προβλήματος. Σε συνδυασμό ότι οι ροές εργασίας παρουσιάζουν διαφορετικά χαρακτηριστικά, όπως τυπολογική δομή, μέγεθος και καμιά φορά δραματικές απαιτήσεις σε ότι αφορά τις ανάγκες υπολογισμού-επικοινωνίας. Ενώ ένα μεγάλο θέμα αποτελεί και το γεγονός

ότι μπορούν να έχουν διαφορετικούς ή και αντικρουόμενους στόχους όσον αφορά την βελτιστοποίηση. Τα παραπάνω έχουν ως αποτέλεσμα τον προγραμματισμό αποφάσεων υψηλής πολυπλοκότητας που με τη σειρά τους μπορεί να επηρεάσει αρνητικά την ποιότητα των λύσεων. Αυτό έχει ως αποτέλεσμα να θεωρείται και να έχει αποδειχθεί ως πρόβλημα κατηγορίας NP-Hard.

Ακόμα ο τρόπος με τον οποίο γίνεται μια κατανομή θέσεων εργασίας μπορεί να εξαρτάται όχι μόνο από τις ιδιότητες και τους περιορισμούς τους, αλλά και από τη φύση της υποδομής του συστήματος που μπορεί να υπόκειται σε απρόβλεπτο φόρτο εργασίας που δημιουργείται από ανεξάρτητους χρήστες σε ένα κατανεμημένο περιβάλλον. Ενώ ο προγραμματισμός ροών εργασιών για τα πλέγματα έχει αποτελέσει αντικείμενο έρευνας εδώ και πολλά χρόνια, υπάρχει μια τάση προς πιο σύνθετες ροές εργασιών καθώς το πλέγμα έχει γίνει συνηθισμένο περιβάλλον παραγωγής στον επιστημονικό τομέα. Οι περισσότερες από τις μελέτες έχουν ασχοληθεί με τον προγραμματισμό ενός μόνο DAG (άκυκλος κατευθυνόμενος γράφος) σε έναν μόνο υπολογιστή. Λίγες μελέτες έχουν εξετάσει τον προγραμματισμό πολλαπλών DAG. Ομοίως, ο προγραμματισμός ροής εργασιών έχει λήφθηκε επίσης υπόψη για πλέγματα κατά το παρελθόν καθώς έχουν προταθεί και αρκετοί αλγόριθμοι για διαφορετικούς τύπους Πλέγματος.

Υπάρχει μία σχετική έλλειψη όσο αναφορά τον προγραμματισμό πολλαπλών ροών εργασιών σε περιβάλλοντα όπως τα υπολογιστικά νέφη. Γενικά, το cloud computing είναι ένα μοντέλο επιχείρησης κατά παραγγελία [1] υπολογισμού που παρέχει εύκολη πρόσβαση σε κοινόχρηστα διαμορφώσιμους πόρους στο διαδίκτυο. Η εμφάνιση της τεχνικής της ανάθεσης εργασιών σε υπολογιστικά νέφη κατάφερε να αποδώσει έναν δυνατό με υψηλό ποσοστό απόδοσης και αξιόπιστο υπολογισμό κατά παραγγελία για κάποιους εξωτερικούς πελάτες στο διαδίκτυο. Επί του παρόντος, τα μοντέλα υπηρεσιών, όπως το λογισμικό ως υπηρεσία, πλατφόρμα ως υπηρεσία και υποδομή ως υπηρεσία είναι τα κύρια μοντέλα που έχουν αναπτυχθεί για αυτήν την τεχνική [2]. Η υπηρεσία βασίζεται επίσης στις αλληλεπιδράσεις μεταξύ πελατών και των παροχών του νέφους. Το κύριο συστατικό της αρχιτεκτονικής υπηρεσιών νεφών είναι το σύστημα διαχείρισης cloud (CMS) που περιλαμβάνουν ουρά αιτημάτων, σύστημα διαχείρισης πόρων (RMS) κ.λπ. Το CMS στην πραγματικότητα ορίζει το απαιτούμενα προγράμματα και αντίστοιχους πόρους δεδομένων σύμφωνα με διάφορους τύπους υπηρεσιών . Οι ερευνητικές μελέτες έχουν χρησιμοποιήσει τα υπολογιστικά πλέγματα από διαφορετικούς τομείς της επιστήμης για την εκτέλεση πολύπλοκων επιστημονικών εφαρμογών. Άλλα όλο και περισσότερες επιστημονικές εφαρμογές υπολογιστών έχουν μεταφερθεί στο υπολογιστικό νέφος. Με την αύξηση του φόρτου εργασίας και τη δυσκολία των εργασιών που υποβάλλονται τέθηκε το εξής ερώτημα. «Πώς να ολοκληρωθούν αυτές τις εργασίες αποτελεσματικά και γρήγορα με περιορισμένους πόρους από το υπολογιστικό νέφος?». Το σημείο μιας προσέγγισης προγραμματισμού εργασιών είναι ίσως ο εντοπισμός μιας αντιστάθμισης μεταξύ των αναγκών των χρηστών και των πόρων που χρησιμοποιούνται από τους χρήστες. Ωστόσο, όπως προαναφέρθηκε οι εργασίες που υποβάλλονται από διαφορετικούς χρήστες ενδέχεται να έχουν διαφορετικές ανάγκες υπολογιστικός χρόνος, χώρους μνήμης, κίνηση δεδομένων, χρόνους απόκρισης, και λοιπά. Επιπλέον, παρουσιάζεται ένα πρόβλημα με τον αυξανόμενο φόρτο εργασίας των εργασιών που υποβάλλονται από καταναλωτές στο νέφος και οι πόροι του νέφους ενδέχεται να μην επαρκούν για να ολοκληρωθούν τις εργασίες εγκαίρως.

Όπως αναφέρθηκε παραπάνω, ο προγραμματισμός εργασιών θεωρείται μια πολύ αναγκαία τεχνική στο υπολογιστικό νέφος. Σύμφωνα με το διαθέσιμα ερευνητικά δεδομένα, οι περισσότεροι από τους υπάρχοντες μηχανισμούς επικεντρώνονται στη δημοπρασία της κατανομής των πόρων του νέφους και όχι στην επακόλουθη εκτέλεση των εργασιών από τους κατανεμημένους πόρους του νέφους [3].

Στην παρούσα εργασία προτάσσονται τρόποι επιλύσεις του προγραμματισμού μονής ροής εργασιών καθώς και πολλαπλών ροών εργασίας με βέλτιστη κατανομή πόρων με περιβάλλοντα όπως τα πλέγματα και τα υπολογιστικά νέφη. Μέσω διαφόρων προσαρμοστικών παραλλαγών κάποιον θεμελιωδών αλγορίθμων που έχουν προταθεί από την βιβλιογραφία με συνδυασμό κάποιον αλγορίθμων βελτιστοποίησης. Οι αλγόριθμοι βελτιστοποίησης υπερτερούν των συμβατικών αλγορίθμων. Ως εκ τούτου, αυτή η εργασία άντλησε τον συνδυασμό κάποιων από αυτούς συγκρίνοντας την αποτελεσματικότητα τους.

Για την ευκολότερη ανάλυση, επαλήθευση και αποσφαλμάτωση τόσο των αποτελεσμάτων όσο και των δεδομένων δημιουργήθηκαν προσαρμοσμένα εργαλεία οπτικοποίησης το οποίο αποτελείται από δύο βασικά διακριτά σκέλη καθώς και εργαλεία αποσφαλμάτωσης και εργαλεία παραγωγής δεδομένων. Πρακτικά πρόκειται για ένα σύνολο εργαλείων που αναπτύχθηκαν τα οποία αποτέλεσαν καθοριστικό ρόλο στην κατάλληλη ανάπτυξη και στην ανάλυση των αλγορίθμων που μελετήθηκαν αλλά και προτάθηκαν. Υπήρξε ακόμα ένα εργαλείο προσομοίωσης το οποίο είχε πρωταρχικό ρόλο όσο αναφορά των προγραμματισμό πολλαπλών ροών εργασιών σε διάφορα κατανεμημένα περιβάλλοντα. Όσο αφορά τα εργαλεία οπτικοποίησης μπορούν να κατηγοριοποιηθούν σε τρεις κατηγορίες η πρώτη έχει να κάνει με την οπτικοποίηση της γενικότερης μορφής μίας ροής εργασιών, η δεύτερη έχει να κάνει με την μελέτη των χρονοδιαγραμμάτων που παράγονται από τον προγραμματισμό ροών εργασιών, και τέλος η τρίτη με την ανάλυση των αποτελεσμάτων που παράγουν οι αλγόριθμοι χρονοδρομολόγησης. Επίσης θα γίνει αναφορά των τεχνολογιών που χρησιμοποιήθηκαν για την ανάπτυξη των συγκεκριμένων εργαλείων.

Βιβλιογραφική Επισκόπηση

Στο παρόν κεφάλαιο αρχικά γίνεται αναφορά στην μοντελοποίηση του προβλήματος και μία εκτενής αναφορά στον τρόπο επιλογής των δεδομένων καθώς και των τρόπο δημιουργίας αυτών που χρησιμοποιήθηκαν. Ακόμα γίνεται μία μικρή επισκόπηση αναφορικά με τις προτάσεις επίλυσης του προβλήματος που η βιβλιογραφία πρότεινε. Καθώς και τις προτάσεις τις οποίες προτείνει η ίδια η εργασία. Αυτό εκτείνεται και στον προγραμματισμό μονής ροής εργασιών αλλά και σε προγραμματισμό πολλαπλών ροών εργασιών.

Όσο αφορά για τις μελέτες που εξετάστηκαν λόγω της δημοτικότητας του προβλήματος εκατοντάδες μελέτες και αλγόριθμοι αναπτύχθηκαν με σκοπό την βελτιστοποίηση της χρονοδρομολόγησης μίας ροής εργασιών. Μέσα από αυτές κάποιες ξεχώρισαν και είχαν θεμελιώδη ρόλο στην ανάπτυξη μετέπειτα αλγορίθμων. Επομένως έγινε επιλογή ανάπτυξης συγκεκριμένων αλγορίθμων που αποδείχθηκαν αποτελεσματικοί και αναγνωρίστηκαν από την βιβλιογραφία. Ακόμα επιλέχθηκαν και μελετήθηκαν αλγόριθμοι οι οποίοι είχαν σαν τεχνικό περιβάλλον ένα εικονικό κατανεμημένο σύστημα το οποίο να είναι παρεμφερές ενός υπολογιστικού νέφους προσομοιώνοντας ουσιαστικά περιβάλλοντα και σενάρια τα οποία είναι ρεαλιστικά. Αυτό επιτρέπει μία πιο αξιόπιστη όψη αυτών των αλγορίθμων. Αυτό έδωσε το ερέθισμα να υιοθετηθούν κάποια στοιχεία τους από μεθόδους που προτάθηκαν και από την παρούσα εργασία.

Μοντελοποίηση του προβλήματος 2.1

Ένα σύνολο εφαρμογών μπορεί να αποτυπωθεί ως μία ροή διαδικασιών/εργασιών. Στον κλάδο της πληροφορικής και όχι μόνο η αγγλική ορολογία είναι «Workflow». Μεμονωμένες εργασίες/διαδικασίες χαρακτηρίζονται ως «Tasks» ή «Jobs» και οι υπολογιστικοί πόροι αναφέρονται ως «Machines» ή «Processors». Μια ροή εργασίας περιγράφει τις εξαρτήσεις μεταξύ των εργασιών και η δομή της επιβάλλεται να είναι αυτή ενός κατευθυνόμενου άκυκλου γράφου (DAG) [4] , όπου οι κόμβοι είναι οι εργασίες και οι ακμές υποδηλώνουν τις αλληλεξαρτήσεις μεταξύ αυτών. Όσο αφορά το πρόβλημα του προγραμματισμού μίας ροής εργασίας, δίνεται μεμονωμένα μία ροή εργασιών W. Όπως αναφέρθηκε ήδη η ροή έγκειται να είναι σε μορφή κατευθυνόμενου άκυκλου γράφου όπου $W = (T, E)$ όπου T είναι το σύνολο των εργασιών και E είναι το σύνολο των ακμών μεταξύ των εργασιών. Κάθε εργασία t φέρει μια υπολογιστική απαίτηση g, δηλαδή, αριθμό εντολών MIPS που πρέπει να εκτελεστούν για την ολοκλήρωσή της. Κάθε άκρη e_z αντιπροσωπεύει εξαρτήσεις δεδομένων μεταξύ της i-ης εργασίας του $w(t)$ και t_z. Η εκτέλεση μιας εργασίας εξαρτάται από τις ακόλουθες συνθήκες: α) όλες οι προηγούμενες εργασίες που είναι γονείς της ($pred(t_i)$) πρέπει να έχουν ολοκληρωθεί και β) όλα τα δεδομένα από τους γονείς $pred(t_i)$ πρέπει να μεταφερθούν προς τη σύνδεση t_i. Τα δεδομένα που πρέπει να μεταδοθούν μεταξύ t_i και t_{iz} είναι μεγέθους d_{iz} μονάδων δεδομένων, παραδείγματος χάριν σε MBs.

Το τεχνικό περιβάλλον που προγραμματίστηκε για να τρέξουν οι ροές εργασιών αποτελείται από ένα σύνολο πόρων $M, (1 \leq i \leq |M|)$. Κάθε πόρος χαρακτηρίζεται από παραμέτρους ισχύος επεξεργασίας. Ποιο συγκεκριμένα ένας υπολογιστικός πόρος mi έχει έναν ακέραιο αριθμό από επεξεργαστές U, έναν ακέραιο αριθμό S που υποδεικνύει την ταχύτητα του πυρήνα, μία μεταβλητή N που υποδεικνύει τις ταχύτητες του δικτύου στο οποίο βρίσκεται ο υπολογιστικός πόρος και τέλος μία μεταβλητή M που δείχνει την υπάρχουσα μνήμη RAM που έχει στην διάθεση του. Όλα μαζί δηλώνουν έναν αριθμό πιθανών εντολών MIPS που μπορούν να εκτελεστούν ανά μονάδα χρόνου (δηλαδή, ένα δευτερόλεπτο) από τον m_i. Αυτό ενσωματώνει επίσης την ταχύτητα μνήμης, το μέγεθος του δίσκου και ούτω καθεξής. Ο χρόνος εκτέλεσης που απαιτείται για την ολοκλήρωση της εργασίας t_i στον πόρο mz υπολογίζεται ως $miz = (\frac{gi}{nz} * sz)$. Για την απλούστευση του προβλήματος στην περίπτωση της χρόνο-δρομολόγησης μίας ροής εργασιών η επικοινωνία μεταξύ των πόρων συμβαίνει με συγκεκριμένη ταχύτητα. Έστω bxy ο ρυθμός μεταφοράς μεταξύ mx και my και aij ο όγκος δεδομένων σε MBs που πρέπει να μεταφερθεί από την μία εργασία a_i στην άλλη a_j. Η καθυστέρηση (χρονικό κόστος) για τη μεταφορά του συνόλου δεδομένων από την μία εργασία στην άλλη εξαρτιέται πλήρως από την ταχύτητα δικτύου στους υπολογιστικούς πόρους και αν υποτεθεί ότι εργασία a_i και a_j. είναι φιλοξενούμενες στον ίδιο υπολογιστικό πόρο τότε θεωρούμε το χρονικό κόστος μεταφοράς μηδενικό. Για κάθε εργασία, οι τελευταίες τιμές υπολογίζονται αναδρομικά ως $EST_i^z = max_{tz \in \text{pred}(t_i)} \{AFT_z + R_{iz}\}$ και $EFT_{zi} = u_{iz} + EST_i^z$. Όταν ο προγραμματιστής αποφασίζει τον πιο ωφέλιμο πόρο για να φιλοξενήσει την εργασία T_i, οι τιμές EST_i^z και EST_i^z είναι ίσες με τους παραπάνω ορισμούς και μπορεί να υπολογιστεί ο συνολικός χρόνος εκτέλεσης μιας ροής εργασίας w (makespan) καθώς $M_i = AFT_i$ όπου t_i είναι η πιο αργή εργασία εξόδου του w_i. Η γενική άτυπη δήλωση του προβλήματος στο έγγραφο είναι η εξής: Βρείτε μια εφικτή αντιστοίχιση μεταξύ των εργασιών του κάθε w και των πόρων στο M, έτσι ώστε το συνολικό εύρος απόδοσης να ελαχιστοποιείται και ισχύουν οι ακόλουθοι περιορισμοί:

(i) Ένας κόμβος δεν μπορεί να εκτελέσει ταυτόχρονα περισσότερες από μία εργασίες.
(ii) Μια εργασία δεν μπορεί να ανατεθεί σε περισσότερους από έναν κόμβους.
(iii) Μία εργασία παιδί πρέπει να περιμένει τα τελειώσουν όλες οι εργασίες γονείς της για να μπορέσει να ξεκινήσει.
(iv) Η εργασία παιδί πρέπει να περιμένει και την μεταφορά του όγκου δεδομένων της εργασίας γονέα για να μπορέσει να ξεκινήσει.

Η διαφορές όταν γίνεται αναφορά σε πολλαπλές ροές εργασιών ανέρχονται κυρίως στο τεχνικό σύστημα που λαμβάνει τις εκάστοτε ροές αλλά και στον τρόπο μοντελοποίησης αυτών. Θεωρείτε ότι ένα σύνολο W_i= (T_i, E_i) όπου T_i είναι το σύνολο των εργασιών και E_i είναι το σύνολο των ακμών. Κάθε εργασία t_{ij} φέρει έναν υπολογισμό απαίτηση g_{ij} και όπως και προηγούμενος κάθε άκρο e_{ijz} αντιπροσωπεύει εξαρτήσεις δεδομένων μεταξύ της j-ης εργασίας του w_i (t_{ij}) και t_{iz}.

Μοντελοποίηση του προβλήματος σε περιβάλλον υπολογιστικού νέφους 2.2

Ένα χρονοδιάγραμμα το οποίο έχει τεθεί να τρέξει στον υπολογιστικό νέφος έρχεται και με μία σειρά νέων προκλήσεων. Αρχικά υπάρχει η αβεβαιότητα του εάν θα τρέξει ένας υπολογιστικός πόρος όπως έχει ανατεθεί να τρέξει. Είναι πολύ πιθανό λόγω μηχανικής βλάβης ή λόγω εξωτερικών παραγόντων να μπορεί να φέρει εις πέρας την εκτέλεση της εργασίας που του έχει ανατεθεί. Επομένως υπάρχει ένα ποσοστό που ορίζει για το εάν θα αποτύχει ο υπολογιστικός πόρος. Το ποσοστό αυτό ειδικά όταν γίνεται αναφορά σε μεγάλα υπολογιστικά νέφη όπως AWS (Amazon Web Services), Google, Azure [5] και λοιπά είναι εξαιρετικά μικρό. Παρόλα αυτά αποτελεί μία πιθανότητα που πρέπει να παρθεί υπόψιν.

Σε μία τέτοια περίπτωση υπάρχουν διάφοροι τρόποι αντιμετώπισης του προβλήματος. Μία ιδέα είναι να εκτελεστεί η εργασία που απέτυχε να τρέξει κατάλληλα ξανά στον ίδιο υπολογιστικό. Μία πιο έξυπνη λύση θα ήταν να γίνει προσπάθεια ανάθεσης της εργασίας στον υπολογιστικό πόρο που καλύπτει καλύτερα το χρονικό όρισμα που έχει θέση ο χρήστης. Τέλος αν θεωρηθεί ο υπολογιστικός πόρος ακατάλληλος ή απλά δεν είναι προσβάσιμος για το υπόλοιπο χρονοδιάγραμμα μπορεί η εργασία να ανατεθεί στον επόμενο καλύτερο πόρο πάλι βάση του χρονικού ορίσματος που έχει τεθεί.

Υπάρχουν επίσης περιπτώσεις στις οποίες οι εργασίες αποτυχαίνουν να εκτελεστούν κατάλληλα καθώς υπάρχει κάποιου είδους προθεσμία. Σε μία τέτοια περίπτωση, όπως θα και δειχθούν και από παραδείγματα από την επιστημονική κοινότητα αργότερα, μπορεί να αλλάξει ο γενικότερος τρόπος χρόνο-δρομολόγησης των εργασιών και να μπουν σε προτεραιότητα εργασίες οι οποίες έχουν σχετικά μικρό χρόνο προθεσμίας. Εργασίες οι οποίες δεν έχουν έρθει εις πέρας από τον χρόνο που ορίζει η προθεσμία τους θεωρούνται ότι απέτυχαν και εάν επηρεάζουν και το υπόλοιπο χρονοδιάγραμμα (με το να έχουν εργασίες παιδιά) ολόκληρη η ροή θεωρείται ότι απέτυχε να εκτελεστεί εντός των χρονικών ορίων.

Ακόμα όσο αναφορά το πρόβλημα της χρόνο-δρομολόγησης στο υπολογιστικό νέφος παρατηρήθηκε το εξής. Πολλοί αλγόριθμοι χρόνο-δρομολόγησης ενώ μπορεί να είχαν ως αποτέλεσμα ένα αξιόλογο χρονοδιάγραμμα πολλές χρονικές στιγμές εντός των υπολογιστικών πόρων έμεναν ανεκμετάλλευτες. Υπήρχαν δηλαδή χρονικές στιγμές στις οποίες οι υπολογιστικοί πόροι έμεναν αδρανείς. Το γεγονός αυτό ειδικά σε περιβάλλον όπως το υπολογιστικό νέφος αποτελεί μείζον ζήτημα. Αυτό γιατί υπάρχουν μοντέλα χρέωσης όπως το Pay-As-You-Go [6] το οποίο χρεώνει τον χρήστη με τον χρόνο που καταλαμβάνει κάποιον υπολογιστικό πόρο. Αν και τα συγκεκριμένα μοντέλα συνηθίζεται να χρεώνουν με την ώρα, μπορεί κάποιος να κάνει την εύλογη συσχέτιση πως η αδράνεια των υπολογιστικών πόρων είναι πρακτικά χρήματα τα οποία χάνει ο χρήστης.

Επισκόπηση αλγορίθμων που μελετήθηκαν για μονές ροές εργασιών 2.3

Στην μελέτη των H. Topcuoglu, S. Hariri and Min-You Wu, με τίτλο "Performance-effective and low-complexity task scheduling for heterogeneous computing," αναπτύσσονται δύο καινοτόμοι αλγόριθμοι, για την μέχρι τότε βιβλιογραφία, με ονομασία HEFT (Heterogeneous Earliest-Finish-Time) και CPOP (Critical-Path-on-a-Processor). Η παρούσα μελέτη δημιουργεί και αναλύει τις δυνατότητες και τις εφαρμογές και των δύο αλγορίθμων αλλά εστιάζει στην απόδοση του αλγορίθμου του HEFT. [7] Παρακάτω γίνεται μία μικρή αναφορά σε κάθε αλγόριθμο που μελετήθηκε και αναλύεται λεπτομερώς αργότερα.

Ο HEFT σαν αλγόριθμος ξεχώρισε ανάμεσα στην μέχρι τότε βιβλιογραφία καθώς με σχετικά μικρή πολυπλοκότητα καταφέρνει να βρίσκει έναν πιθανό υπολογιστικό πόρο για μία εργασία στον οποίο η εκτέλεση της θα είναι βέλτιστη για εκείνη την χρονική στιγμή. Απαραίτητο να αναφερθεί φυσικά είναι ότι ο αλγόριθμος λαμβάνει υπόψιν τους περιορισμούς ανάμεσα στις εργασίες και βάση αυτών βρίσκει κάθε φορά το κατάλληλο υπολογιστικό πόρο. Ο αλγόριθμος επίσης είχε ένα στάδιο το οποίο θα μπορούσε κανείς να χαρακτηρίσει και ως προετοιμασίας. Αυτό έχει να κάνει με την ανάθεση προτεραιοτήτων σε κάθε εργασία.

Ενώ μελετάται πιο αναλυτικά η αποδοτικότητα του αλγορίθμου του HEFT, μεταξύ άλλων γίνεται και αναφορά στον αλγόριθμο CPOP αλλά και ανάπτυξη αυτού. Ενώ παράλληλα γίνεται και χρήση κάποιον κομματιών του σε μετέπειτα αλγόριθμους που παρουσιάστηκαν στην βιβλιογραφία. Πιο συγκεκριμένα γίνεται χρήση το κομμάτι της προετοιμασίας του αλγορίθμου το οποίο έχει να κάνει με την εύρεση του κρίσιμου μονοπατιού.

Εκτενής αναφορά, εξέταση και εφαρμογή επίσης έγινε σε έναν αλγόριθμο ο οποίος παρόλο που ίσως αργότερα αναπτύχθηκε και θεωρήθηκε ως ολόκληρη κατηγορία αλγορίθμων έθεσε μία νέα οπτική στα προβλήματα βελτιστοποίησης. Ο αλγόριθμος του Ruin and Recreate [8] πρακτικά είναι μία προσέγγιση που όπως θα φανεί και από την παρούσα εργασία είχε εκπληκτικά αποτελέσματα. Πρόκειται για έναν αλγόριθμο που ανήκε στην κατηγορία των μετά-ευρετικών. Προτάθηκε από τους Gerhard Schrimpf, Johannes Schneider, Hermann Stamm-Wilbrandt, Gunter Dueck. Ο αλγόριθμος εφαρμόζεται σε μία ήδη υπάρχων λύση ενός προβλήματος, εξού και η κατηγορία μετά-ευρετικός, βασίζεται στην ιδέα της καταστροφής ενός ποσοστού της λύσης και στην αναδόμηση της με διάφορους τρόπους, με σκοπό να επιτευχθεί ένα καλύτερο αποτέλεσμα. Όσο αναφορά την εργασία έγινε εφαρμογή και του ίδιου του αλγορίθμου του Ruin and Recreate αλλά προσαρμοσμένων εκδοχών αυτού.

Η πρώτη συνδυαστική προσπάθεια επίλυσης του προβλήματος έγινε με μία νέα τεχνική που ονομάστηκε Holes Scheduling (χρονοδρομολόγηση τρυπών). Η τεχνική πρακτικά έχει να κάνει με την εκμετάλλευση των χρονικών τρυπών που παράγονται κατά τον προγραμματισμό μίας ροής εργασιών με την χρήση τεχνικών bin packing. Ποιο συγκεκριμένα οι τεχνικές bin packing που εξετάστηκαν ήταν οι εξής:

- Best-Fit
- Worst-Fit

- First-Fit
- Quickest-Fit

Με την τελευταία να πρόκειται για πιο πολύ τρόπο επιλογής κατάλληλης θέσης μίας εργασίας παρά bin packing τεχνική. Αργότερα θα γίνει εκτενής αναφορά στις παραπάνω και θα εξηγηθεί και η ιδιαιτερότητα της τελευταίας τεχνικής.

Επισκόπηση αλγορίθμων που μελετήθηκαν για πολλαπλές ροές εργασιών 2.4

Μία προσέγγιση που μελετήθηκε είναι η ιδέα του να συνδυαστούν όλες οι ροές εργασιών σε μία με το σκεπτικό ότι σε μονή ροή εργασιών η πολυπλοκότητα μειώνεται και οι αλγόριθμοι που μπορούν να εφαρμοστούν μπορούν εξίσου να φέρουν αξιόλογα αποτελέσματα. [9] Από τους τέσσερις αλγορίθμους που προτάθηκαν εξετάστηκαν και οι τέσσερις.

- Common Entry Common Exit Node (C1)

Η πρώτη ιδέα ήταν αυτή που δημιουργεί το σύνθετο γράφημα. Αφού ενοποιηθούν όλοι ο κόμβοι τρέχει τον αλγόριθμο του HEFT στην πλέον ενοποιημένη ροή εργασιών. Η ενοποίηση αυτή γίνεται καθιστώντας τους κόμβους εισόδου όλων των ροών εργασιών άμεσους διαδόχους ενός νέου κόμβου εισόδου και όλους τους κόμβους εξόδου των ροών εργασιών άμεσους προγόνους ενός νέου κόμβου εξόδου. Αυτοί οι δύο επιπλέον κόμβοι (δηλ. νέα είσοδος και νέος κόμβος εξόδου) δεν έχουν υπολογισμό και καμία επικοινωνία μεταξύ αυτών και άλλων κόμβων.

- Level-Based Ordering (C2)

Αυτή η προσέγγιση δημιουργεί ένα σύνθετο γράφημα με τον ίδιο τρόπο όπως αναφέρθηκε στο 2.2.α. Ωστόσο, το νέο γράφημα ομαδοποιείται σε επίπεδα και κατ' επέκταση ο προγραμματισμός πραγματοποιείται σε επίπεδα. Σαφώς, σε κάθε επίπεδο, υπάρχουν μόνο ανεξάρτητες εργασίες που πρέπει να προγραμματιστούν. Αυτά μπορούν να προγραμματιστούν χρησιμοποιώντας οποιονδήποτε αλγόριθμο για τον προγραμματισμό ανεξάρτητων εργασιών για την απλοποίηση για την γρήγορη δημιουργία παραδειγμάτων για μελέτη επιλέχθηκε ο αλγόριθμος του HEFT.

- Alternating DAGs (C3)

Η συγκεκριμένη προσέγγιση δημιουργεί ξανά ένα σύνθετο γράφημα ακριβώς με τον ίδιο τρόπο όπως αναφέρθηκε και στο 2.2.α. Η γενικότερη ιδέα της μελέτης αυτής βασίζεται πάνω την ενοποίηση των ροών σε μία. Μετά την ενοποίηση ο αλγόριθμος εξετάζει τις εργασίες από κάθε ροή διαδοχικά εξετάζοντας κάθε μία εργασία από διαφορετική ροή. Δημιουργεί δηλαδή έναν κύκλο από τις ροές και κάθε φορά ο δείκτης δείχνει στην επόμενη ροή από την οποίο διαλέγει και μία εργασία. Αυτό επιτρέπει την πιο δίκαια κατανομή πόρων μεταξύ των ροών εργασιών.

- Ranking-Based Composition DAGs (C4)

Εάν και δεν θα εξεταστεί ο αλγόριθμος αργότερα αξίζει να αναφερθεί.
Σαν C4 οι συγγραφείς του άρθρου αναφέρονται στον Ranking-Based Composition. Η ιδέα της προσέγγισης αυτής είναι ότι λαμβάνεται υπόψιν η δομή της κάθε ροής εργασίας και ο χρόνος εκτέλεσης της κάθε εργασίας ατομικά, όταν ενώνονται οι εργασίες εξόδου. Έτσι μία εργασία εξόδου μίας “κοντής” ροής εργασιών ενώνεται με μία της οποίας το μακρύτερο μονοπάτι από την εργασία εισόδου της είναι περίπου το ίδιο με το μεγαλύτερο μονοπάτι της κοντής ροής εργασιών. Ως αποτέλεσμα βγαίνει ένα ενοποιημένο γράφημα στο οποίο είναι πολύ πιο εύκολο να τρέξει κάποιος αλγόριθμος όπως του HEFT και να μπορέσει να δώσει δίκαιες προτεραιότητες σε κάθε εργασία.

Ακόμα μία σειρά αλγορίθμων που προτάθηκε από τους (Georgios L. Stavrinides και Helen D. Karatza, 2011) [10]. Σε αυτό το άρθρο γίνεται διερεύνηση του προγραμματισμού πολλαπλών γραφημάτων εργασιών με προθεσμίες από άκρο σε άκρο σε μία ετερογενή κατανομή. Χρησιμοποιείται σύστημα σε πραγματικό χρόνο, που χρησιμοποιεί ευρετικό προγραμματισμό λίστας, όπου η φάση επιλογής εργασιών βασίζεται μεταξύ συγκεκριμένων ορισμάτων. Ένα από τα επίκεντρα της παραπάνω μελέτης ήταν η δίκαια κατανομή των πόρων μεταξύ των ροών εργασιών που έρχονται σε ένα κατανεμημένο σύστημα. Έγινε λοιπόν προσπάθεια δημιουργίας κάποιων μεθόδων οι οποίες σε πραγματικό χρόνο θα λάμβαναν υπόψη τα χρονικά περιθώρια εκτέλεσης που μπορεί να είχε θέσει ένας χρήστης για την ροή του και θα χρονοδρομολογούσαν τις εργασίες με δίκαιο τρόπο. Κάποιες από τις πολιτικές προτεραιότητας που αναπτύχθηκαν είναι οι εξής:

- EDF

Η λογική είναι ότι υπάρχει ένα κεντρικό σύστημα στο οποίο έρχονται εργασίες σε πραγματικό χρόνο. Έπειτα οι εργασίες ταξινομούνται βάση της πολιτικής EDF (Earliest Deadline First) και λαμβάνει υπόψη μόνο τις προθεσμίες των θέσεων εργασίας για την ιεράρχηση εργασιών. Πρακτικά μία εργασία έχει προτεραιότητα έναντι κάποιας άλλης όταν έχει μικρότερο χρονικό περιθώριο μέχρι να εκτελεστή. Αυτό το χρονικό περιθώριο ορίζεται από την ροή στην οποίο ανήκει η εργασία.

- HLF

Στην συγκεκριμένη μέθοδο γίνεται χρήση του ίδιου κεντρικού συστήματος που προαναφέρθηκε και στην ενότητα 2.2.δ. Πρόκειται για ένα από τους απλούστερους αλγόριθμους προγραμματισμού λίστας. Ο Highest Level First with Estimated Times (HLFET), ή πιο απλά, η πολιτική υψηλότερου επιπέδου πρώτου (HLF). Η πολιτική HLF (Highest Level First) που λαμβάνει υπόψη μόνο τα επίπεδα των εργασιών, αγνοώντας τυχόν περιορισμούς από χρονικά περιθώρια που μπορεί να έχει θέση κάποιος χρήστης.

- LSTF

Η πολιτική LSTF (Least Space-Time First) εφαρμόζεται πάλι σε ένα κεντρικό σύστημα το οποίο προγραμματίζει της εργασίες οι οποίες έρχονται σε πραγματικό χρόνο. Η συγκεκριμένη πολιτική λαμβάνει υπόψιν τόσο την προτεραιότητα όσο και τους χρονικούς περιορισμούς μεταξύ των εργασιών και των καθηκόντων τους.

Ωστόσο, το κόστος επικοινωνίας μεταξύ των εργασιών σε από μία ροή εργασιών δεν λαμβάνεται υπόψη.

Μελετήθηκαν επίσης ιδέες οι οποίες είναι εμπνευσμένες από την βιβλιογραφία. Μία από αυτές είναι η ιδέα του CTF (Critical Tasks First). Εμπνευσμένο από την ιδέα του Critical Path (από τον αλγόριθμο του CPOP) στην παρούσα εργασία αναπτύχθηκε μία μέθοδος η οποία στρέφει στο επίκεντρο στις εργασίες τις ίδιες. Αυτό συμβαίνει γιατί κατά την ανάλυση των ροών που έχουν τεθεί να προγραμματιστούν, υπάρχουν εργασίες οι οποίες η μέσος όρος εκτέλεσης τους μπορεί να καταλαμβάνει ένα πολύ μεγάλο ποσοστό συγκριτικά με τις υπόλοιπες. Το πως επηρεάζει αυτό το τελικό χρόνο-διάγραμμα και πως αντιμετωπίστηκε θα γίνει ανάλυση αργότερα στα κεντρικά κεφάλαια που αφορούν τον προγραμματισμό πολλαπλών ροών εργασιών. Ο CTF (Critical Tasks First) όπως προαναφέρθηκε πρόκειται για έναν αλγόριθμο ευρετικό ο οποίος έχει ως σκοπό να ανακαλύψει τυχόν εργασίες οι οποίες επηρεάζουν το τελικό χρονοδιάγραμμα σε βαθμό που η βέλτιστη τοποθέτηση αυτών θεωρείται αναγκαία. Όταν και αν βρεθούν αυτές η εργασίες εντός των ροών γίνεται προγραμματισμών αυτών όσο τον δυνατόν πιο γρήγορα και στον πιο δυνατό υπολογιστικό πόρο. Αυτό γίνεται με σκοπό να ολοκληρωθεί η εκάστοτε κρίσιμη εργασία και να μπορέσουν οι υπόλοιπες να καλύψουν τα πελώρια κενά που θα αφήσει στο σύστημα. Ο λόγος που αυτές η εργασίες είναι αναγκαία να χρόνο-δρομολογούνται όσων τον δυνατόν γρηγορότερα γίνεται σαφές αργότερα στα επόμενα κεφάλαια.

Με αφορμή κάποιες από τις μεθόδους που προτάθηκαν στην βιβλιογραφία προτάθηκαν νέες καινοτόμες προτάσεις όπως η ιδέα του αλγορίθμου MWCP (Multiple Workflow Complementary Packing algorithm) [11] ο οποίος εκμεταλλεύεται την διαφορετική μορφολογία της κάθε ροής με σκοπό να γεμίσει όσα περισσότερα χρονικά κενά υπάρχουν στο σύστημα και να μειώσει τον συνολικό χρόνο που χρειάζεται να τρέξουν όλες οι ροές μαζί. Πρακτικά ο αλγόριθμος κοιτάζει να βρει ροές οι οποίες είναι συμπληρωματικές η μία με την άλλη με σκοπό να μην αφήσει κενά και όποιον τυχόν κενό δημιουργηθεί στην πορεία να το καλύψει καταλλήλως με την χρήση των bin-packing τεχνικών. Κάποιες από αυτές όπως αναφέρθηκαν προηγουμένων συνδυάζουν και άλλες μεθόδους χρόνο-δρομολόγησης. Ο αλγόριθμος επίσης συνδυάζει τον προγραμματισμό λιστών με μεθοδολογίες και τεχνικές bin-packing και προσπαθεί να βρει συμπληρωματικά στοιχεία μεταξύ των ροών εργασιών τα οποία υποδεικνύουν το πόσο αποδοτικό θα ήταν αν προγραμματιστούν αυτές οι ροές μαζί. Αφού βρει αυτά τα στοιχεία δημιουργεί ζευγάρια ροών τα οποία ορίζει ότι θα χρόνο-δρομολογηθούν μαζί. Έπειτα στην κάθε ροή εντός των ζευγαριών δίνεται ένα όρισμα το οποίο ορίζει τον τρόπο με τον οποίο η εργασία θα επιλέξει κάποιον υπολογιστικό πόρο. Δεν στέκεται δηλαδή μόνο στον θεμελιώδη τρόπου του HEFT το Earliest Time Finish.

Αναφορά στα εργαλεία που αναπτύχθηκαν 2.5

Με αφορμή κάποιες από τις μεθόδους που προτάθηκαν στην βιβλιογραφία προτάθηκαν νέες καινοτόμες προτάσεις όπως η ιδέα του αλγορίθμου MWCP (Multiple Workflow Complementary Packing algorithm) ο οποίος εκμεταλλεύεται την διαφορετική μορφολογία της κάθε ροής με σκοπό να γεμίσει όσα περισσότερα

χρονικά κενά υπάρχουν στο σύστημα και να μειώσει τον συνολικό χρόνο που χρειάζεται να τρέξουν όλες οι ροές μαζί. Πρακτικά ο αλγόριθμος κοιτάζει να βρει ροές οι οποίες είναι συμπληρωματικές η μία με την άλλη με σκοπό να μην αφήσει κενά και όποιον τυχόν κενό δημιουργηθεί στην πορεία να το καλύψει καταλλήλως με την χρήση των bin-packing τεχνικών. Κάποιες από αυτές όπως αναφέρθηκαν προηγουμένων συνδυάζουν και άλλες μεθόδους χρόνο-δρομολόγησης. Ο αλγόριθμος επίσης συνδυάζει τον προγραμματισμό λιστών με μεθοδολογίες και τεχνικές bin-packing και προσπαθεί να βρει συμπληρωματικά στοιχεία μεταξύ των ροών εργασιών τα οποία υποδεικνύουν το πόσο αποδοτικό θα ήταν αν προγραμματιστούν αυτές οι ροές μαζί. Αφού βρει αυτά τα στοιχεία δημιουργεί ζευγάρια ροών τα οποία ορίζει πως θα χρονοδρομολογηθούν μαζί.

Περισυλλογή και δημιουργία δεδομένων

Η αναζήτηση, η περισυλλογή και η δημιουργία των δεδομένων χωρίστηκε σε στάδια. Αναζήτηση δεδομένων, περισυλλογή/δημιουργία, εγκυροποίηση και τροποποίηση. Μία αρχική αναζήτηση έγινε με στόχο να βρεθούν δεδομένα τα οποία να μπορούν καλύπτουν μία μεγάλη γκάμα πιθανών περιπτώσεων.

Ένα αξιόπιστος αλγόριθμος που κάλυπτε αυτή την ανάγκη δημιουργίας ροών εργασιών προτάθηκε από το ίδιο άρθρο του αλγορίθμου HEFT. Στο άρθρο γίνεται αναφορά για μία γεννήτρια DAGs η οποία τα παράγει βάσει διαφόρων ορισμάτων τα οποία ορίζει ο χρήστης με αποτέλεσμα να δημιουργεί 56 χιλιάδες παραλλαγές γράφων και καλύπτοντας κάθε πιθανό edge case που μπορεί να πέσει ο αλγόριθμος. Αυτό επέτρεψε τη δημιουργία ροών οι οποίες καλύπτουν πλήρως τα απαιτούμενα ζητούμενα για να θεωρηθεί ένας γράφος ότι έχει μορφή ροής εργασιών. Άρα λοιπόν κάθε ροή γράφου που παράχθηκε από τα εργαλεία που δημιουργήθηκαν είχε τα εξής χαρακτηριστικά:

- Άκυκλος
- Κατευθυνόμενος
- Με βάρη

Με βάση αυτή την πρόταση δημιουργήθηκε τροποποιήθηκε ένα εργαλείο το οποίο παρομοιάζει σε μεγάλο βαθμό την λειτουργεία του παραπάνω αλγόριθμου. Η ακριβής λειτουργία του και τα ορίσματα που ορίζουν πως θα παραχθούν οι γράφοι θα αναλυθούν παρακάτω.

Ακόμα η αναζήτηση είχε άλλον έναν γνώμονα ο οποίος εν τέλη επηρέασε ακόμη και την λειτουργία κάποιων αλγορίθμων που προτάθηκαν. Έγινε λοιπόν προσπάθεια εύρεσης δεδομένων τα οποία να προσομοιάζουν σε πραγματικά δεδομένα. Δεδομένα τα οποία να έχουν παραχθεί από επιστημονικές ροές εργασιών οι οποίες είτε έχουν τρέξει σε κάποιο κατανεμημένο σύστημα είτε παρομοιάζουν ροές οι οποίες ενδείκνυται να τρέξουν σε ένα. Το πως ορίζεται αν μία επιστημονική ροή θεωρείται αρκετά ρεαλιστική για να θεωρηθεί ότι ανήκε σε έναν επιστημονικό κλάδο συγκεκριμένα έχει να κάνει κυρίως με την μορφή αυτής. Διάφοροι επιστημονικοί κλάδοι τείνουν να έχουν ροές εργασιών οι οποίες έχουν συγκεκριμένη μορφή ροών εργασιών.

Τα δεδομένα λοιπόν μπορούν να κατηγοριοποιήθηκαν σε Custom generated και Realistic. Τα κομμάτια της εγκυροποίησης και της τροποποίησης αυτών διαφέρουν από κατηγορία σε κατηγορία. Επομένως γίνεται αναφορά σε αυτά σε ξεχωριστό υποκεφάλαιο.

Custom generated δεδομένα 3.1

Στην συγκεκριμένη κατηγορία μπαίνουνε όλες οι ροές εργασιών οι οποίες έχουν παραχθεί βάσει ενός τροποποιημένου προγράμματος το οποίο παράγει γράφους βάσει συγκεκριμένων μεταβλητών. Πρακτικά τροποποιήθηκε ώστε να παράγει σαν αποτέλεσμα DAG τα οποία ποικίλουν σε δομή και μέγεθος μεταξύ τους. Τα συγκεκριμένα δεδομένα έχουν παραχθεί με την βοήθεια ενός αλγορίθμου ο οποίος για να δημιουργήσει ένα DAG λαμβάνει υπόψιν κάποιους παράγοντες.

Ποιο συγκεκριμένα οι παράγοντες αυτοί οι οποίοι διαφοροποιούν το ένα DAG από το άλλο είναι οι εξής:

- n: Αριθμός κόμβων υπολογισμού στο DAG (δηλαδή, "εργασίες" εφαρμογής)
- mindata: Ελάχιστο μέγεθος δεδομένων που υποβάλλονται σε επεξεργασία από μια εργασία
- maxdata: Μέγιστο μέγεθος δεδομένων που υποβάλλονται σε επεξεργασία από μια εργασία
- minalpha: Ελάχιστη τιμή για επιπλέον παράμετρο (π.χ. παράμετρος νόμου Amdahl)
- maxalpha: Ελάχιστη τιμή για επιπλέον παράμετρο (π.χ. παράμετρος νόμου Amdahl)
- fat: Πλάτος του DAG, που είναι ο μέγιστος αριθμός εργασιών που μπορούν να βρίσκονται στο ίδιο επίπεδο (level). Μια μικρή τιμή θα οδηγήσει σε ένα λεπτό DAG (π.χ. αλυσίδα) με χαμηλό παραλληλισμό εργασιών, ενώ μεγάλη τιμή προκαλεί ένα φαρδύ DAG με πολλές εργασίες ανά επίπεδο.
- density: Καθορίζει τον αριθμό των εξαρτήσεων μεταξύ δύο διαδοχικών επιπέδων του DAG.
- regular: Τακτικότητα της κατανομής των εργασιών μεταξύ των διαφορετικών επιπέδων του DAG.
- ccr: Λόγος επικοινωνίας προς υπολογισμό. Η παράμετρος αυτή στην πραγματικότητα απλώς κωδικοποιεί την πολυπλοκότητα του υπολογισμού μιας εργασίας ανάλογα με τον αριθμό των στοιχείων στο σύνολο δεδομένων. Αυτός ο αριθμός στοιχείων εξαρτάται από τον όγκο των δεδομένων που επεξεργάζεται μια εργασία. Ανάλογα τον αριθμό που δώσει ο χρήστης σε αυτή την μεταβλητή μπορεί να βγει ένα DAG το οποίο να είναι πιο πολύ υπολογιστικά βαρύ και ανάμεσα στις εργασίες να μην υπάρχει μεγάλο κόστος επικοινωνίας, δηλαδή μεταφοράς δεδομένων. Φυσικά ανάλογα τον αριθμό μπορεί να επιτευχθεί και το αντίστροφο.
- jump: Μέγιστος αριθμός επιπέδων που εκτείνονται από επικοινωνίες μεταξύ εργασιών. Αυτό επιτρέπει τη δημιουργία DAG με διαφορετικές διαδρομές εκτέλεσης και μήκη. Η συγκεκριμένη μεταβλητή επηρεάζει επίσης και τα πιθανά παιδιά που μπορεί να έχει ένας κόμβος.

Βάση των παραπάνω παραμέτρων και έχοντας μία λίστα για την κάθε παράμετρο:

- minalpha = 20
- maxalpha = 50
- n = [20, 50, 100, 200, 300, 400, 500, 600]
- fat = [0.2, 0.4, 0.6, 0.8]
- density = [0.2, 0.6, 0.8]
- regularity = [0.2, 0.6, 0.8]
- jump = [1, 2, 4, 5]

Με κάθε πιθανό συνδυασμό των παραπάνω παράχθηκαν 1152 διαφορετικά DAG. Αν και δεν αποτελεί αριθμό ο οποίος πλησιάζει αυτόν που προτάθηκε από το άρθρο του αλγορίθμου του HEFT, είναι σημαντικό να σημειωθεί ότι αφού έγινε η εγκυροποίηση αυτών των δεδομένων όσων αφορά την ορθότητα τους και τις περιπτώσεις edge cases που μπορεί να παραχθούν (π.χ προσθέσεις βαρών μεταξύ ακμών λόγω δυαδικού συστήματος δεν είχαν απόλυτη ισότητα) έγινε επιλογή δημιουργίας δεδομένων τα οποία κρίθηκαν ότι θα έχουν μεγαλύτερη πρακτική εφαρμογή με τους αλγορίθμους που τα χρησιμοποίησαν. Για παράδειγμα ένας συνδυασμών μεταβλητών με αυτές τις τιμές:

- n 5
- fat 0.1
- density 0.8
- regular 0.8
- jump 1

Επίσης να αναφερθεί ότι ο λόγος που δεν λαμβάνονται υπόψιν οι μεταβλητές maxalpha, minalpha είναι επειδή δεν επηρεάζουν την τελική μορφή του γράφου ενώ παράλληλα δεν γίνεται χρήση αυτών καθώς τροποποιούνται τα δεδομένα αργότερα. Επομένως οι τιμές τους είναι πιο πολύ ενδεικτικές.

Οι παραπάνω τιμές θα έχουν ως αποτέλεσμα μία ροή εργασιών της μορφής pipeline. Η συγκεκριμένη μορφή λόγω των περιορισμών του προβλήματος της χρόνο-δρομολόγησης της μονής ροής προγραμματισμού πρακτικά δεν έχει ουσιαστική εφαρμογή. Καθώς δεν γίνεται να ανατεθεί διαφορετική σειρά χρόνο-δρομολόγησης στις εργασίες που παράγονται. Σε τέτοιου είδους μορφές ροών δηλαδή υπάρχει μία και μόνο πιθανή λύση χρόνο-δρομολόγησης. Αυτό γίνεται πιο εμφανές και από σχήμα που παρουσιάζεται στην παρακάτω εικόνα.

Εικόνα 3.1. Στιγμιότυπο από τον παραχθείσα γράφο βάση των ορισμάτων που δόθηκαν παραπάνω

Εκ των 1152 διαφορετικών γράφων που είχαν παραχθεί έγινε επιλογή αυτών που κρίθηκαν πιο αντιπροσωπευτικοί για να γίνει μελέτη στην αποτελεσματικότητα των αλγορίθμων. Από αυτούς επιλέχθηκαν οι 252 με τους υπόλοιπους να μην προσφέρουν αρκετά μεγάλη διαφοροποίηση για να θεωρηθούν χρήσιμοι.

Τέλος να αναφερθεί ότι η επιλεκτικότητα αυτή οφείλεται στην πολυπλοκότητα των αλγορίθμων οι οποίοι έτρεξαν πάνω στα δεδομένα. Αυτό σε συνδυασμό με μία αργή εκ φύσεως γλώσσα προγραμματισμού όπως η Python [12] έχουν ως αποτέλεσμα μεγάλους χρόνους εκτέλεσης και η σωστή αναζήτηση, ο αριθμός και η προετοιμασία των δεδομένων έπαιξαν καθοριστικό ρόλο στους χρόνους αυτούς.

Realistic δεδομένα 3.2

Πρόκειται για δεδομένα τα οποία έχουν δημιουργηθεί είτε από πραγματικές ροές εργασιών που έχουν τρέξει ήδη ή προσομοιώνουν πιθανές ροές εργασιών που μπορεί να προκύψουν βάσει του είδους των επιστημονικών πειραμάτων που θα θέσει ο χρήστης. Τα συγκεκριμένα δεδομένα ξεχωρίζουν λόγω της δομής τους η οποία ορίζεται από τον εκάστοτε τύπο πειράματος, και απαγορεύεται να παρεκκλίνουν από την “συνταγή” αυτή. Για παράδειγμα μία ροή εργασιών για να θεωρηθεί ότι αποτελείται από εργασίες σεισμολογίας οφείλει να έχει συγκεκριμένη δομή η οποία είναι ανάλογη. Τα δεδομένα αυτά περισυλέχθηκαν και δημιουργήθηκαν από δύο projects/εργαλεία τα οποία είναι ευρέως αναγνωρισμένα στην επιστημονική κοινότητα. Το project Pegasus [13] και το framework σε Python WfCommons [14].

Με την χρήση των παραπάνω εργαλείων δημιουργήθηκαν ροές εργασιών οι οποίες είχαν συγκεκριμένη δομή. Η δομή τους όπως προαναφέρθηκε έχει να κάνει με την κατηγορία της ίδιας της επιστημονικής ροής. Μία ροή εργασιών βιοπληροφορικής θα έχει διαφορετική δόμηση από μία η οποία ασχολείται με ωκεανολογία.

Pegasus 3.2.α

Το έργο Pegasus περιλαμβάνει ένα σύνολο τεχνολογιών που βοηθούν τις εφαρμογές που βασίζονται στη ροή εργασίας να εκτελούνται σε πολλά διαφορετικά περιβάλλοντα, όπως desktops, πλέγματα, νέφη, campus clusters [15]. Γεφυρώνει πρακτικά τον επιστημονικό τομέα και το περιβάλλον εκτέλεσης αντιστοιχίζοντας αυτόματα περιγραφές ροής εργασίας υψηλού επιπέδου σε κατανεμημένους υπολογιστικούς πόρους. Το Pegasus δίνει τη δυνατότητα στους επιστήμονες να κατασκευάζουν ροές εργασίας με αφηρημένο τρόπο χωρίς να ανησυχούν για τις λεπτομέρειες του υποκείμενου περιβάλλοντος εκτέλεσης. Έχει χρησιμοποιηθεί σε διάφορους επιστημονικούς τομείς όπως η βιοπληροφορική, η επιστήμη των σεισμών, η φυσική των βαρυτικών κυμάτων, η αστρονομία, η επιστήμη των ωκεανών και άλλους. Ακόμα το Pegasus διέθεσε ένα σύνολο ρεαλιστικών ροών εργασίας από διάφορες επιστημονικές εφαρμογές. Αυτές οι ροές εργασιών είναι διαθέσιμες σε μορφή γράφου DAG σαν αρχεία τύπου XML [16] (DAX), σε διαφορετικά μεγέθη (δηλαδή, αριθμός εργασιών). Ένα αρχείο DAX χαρακτηρίζει λεπτομερώς τη δομή, τα δεδομένα και τις υπολογιστικές απαιτήσεις για μια συγκεκριμένη ροή εργασίας.

WfCommons 3.2.β

Το framework WfCommons είναι αναπτυγμένο στην προγραμματιστική γλώσσα Python [17] και βοηθά στην διευκόλυνση της επιστημονικής έρευνας και ανάπτυξης ροής εργασιών παρέχοντας θεμελιώδη εργαλεία για την ανάλυση περιπτώσεων

εκτέλεσης ροής εργασιών, τη δημιουργία συνταγών ροής εργασίας και τη δημιουργία συνθετικών, αλλά ρεαλιστικών περιπτώσεων ροής εργασίας που μπορούν να χρησιμοποιηθούν για την ανάπτυξη νέων τεχνικών, αλγορίθμων και συστημάτων που μπορούν να ξεπεράσουν τις προκλήσεις της αποτελεσματικής και εύρωστης εκτέλεσης ολοένα και μεγαλύτερων ροών εργασιών σε όλο και περίπλοκες κατανεμημένες υποδομές.

Από κάθε κατηγορία δημιουργήθηκαν 12 διαφορετικά μεγέθη ροών εργασιών με την βοήθεια του εργαλείου WfCommons. Πιο συγκριμένα τα μεγέθη ήταν τα εξής:

[10, 14, 20, 30, 50, 100, 133, 200, 300, 400, 500, 1000]

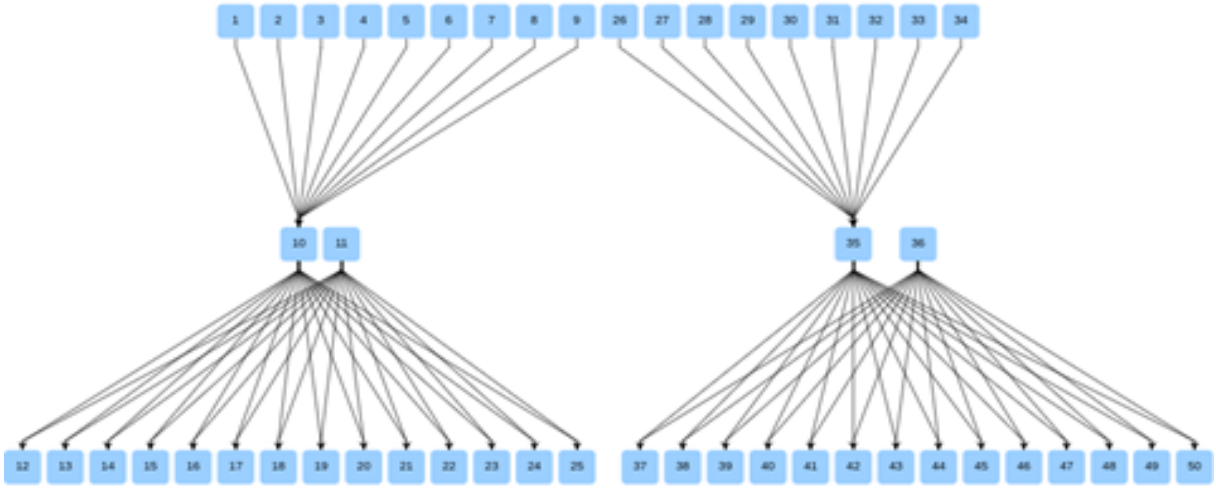
Από αυτά δημιουργήθηκαν 8 παραλλαγές επομένως 576 ροές εργασιών που δημιουργήθηκαν από πραγματικά/ρεαλιστικά δεδομένα. Εκ των οποίων τα μεγέθη τα οποία χρησιμοποιήθηκαν περισσότερο ήταν ανάμεσα σε [10, 50, 100, 300, 500, 1000].

Κατηγορίες ρεαλιστικών δεδομένων 3.2.γ

Από τα παραπάνω εργαλεία και τις ροές εργασιών που παρήχθησαν έγινε επιλογή κάποιων ροών εργασιών από συγκεκριμένες κατηγορίες και οι οποίες είχαν μεγάλη δημοτικότητα ανάμεσα στην επιστημονική κοινότητα. Κάποιες από τις κατηγορίες που χρησιμοποιήθηκαν είναι οι εξής:

- 1000Genome [18]

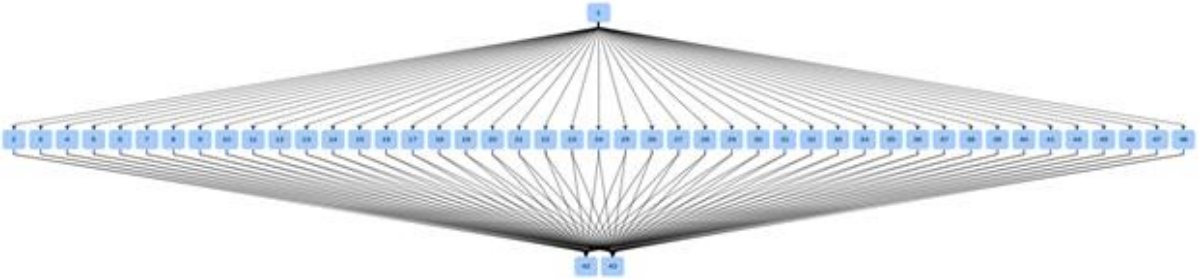
Το 1000 Genome Project δημιούργησε έναν κατάλογο κοινής ανθρώπινης γενετικής παραλλαγής, χρησιμοποιώντας δείγματα με ανοιχτή συναίνεση. Οι πόροι δεδομένων αναφοράς που δημιουργούνται από το έργο εξακολουθούν να χρησιμοποιούνται σε μεγάλο βαθμό από την κοινότητα της βιοϊατρικής επιστήμης.



Εικόνα 3.1. Το παραπάνω στιγμιότυπο απεικονίζει μία ροή εργασιών τύπου 1000Genome που δημιουργήθηκε με προσαρμοσμένο εργαλείο οπτικοποίησης.

- BLAST [19]

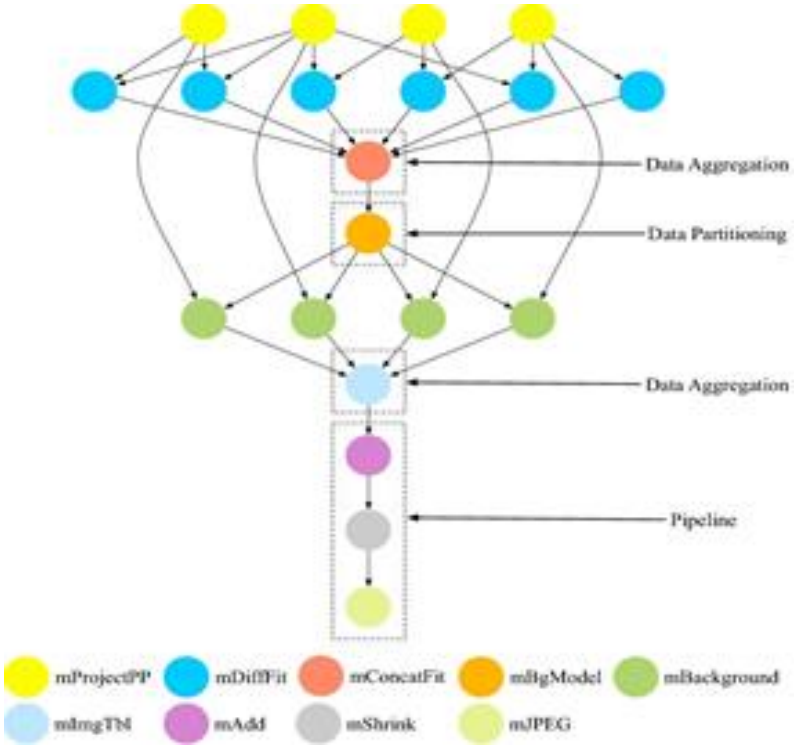
Στη βιοπληροφορική, το BLAST είναι ένας αλγόριθμος και πρόγραμμα για τη σύγκριση πληροφοριών πρωτογενούς βιολογικής αλληλουχίας, όπως οι αλληλουχίες αμινοξέων των πρωτεϊνών, και αν και ψάχνει μόνο τα πιο σημαντικά μοτίβα στις ακολουθίες πρόκειται για έναν αλγόριθμο με μεγάλη πολυπλοκότητα και τεράστιο όγκο που πρέπει να επεξεργαστεί.



Εικόνα 3.2. Το παραπάνω στιγμιότυπο απεικονίζει μία ροή εργασιών τύπου BLAST που δημιουργήθηκε με προσαρμοσμένο εργαλείο οπτικοποίησης.

- Montage [20]

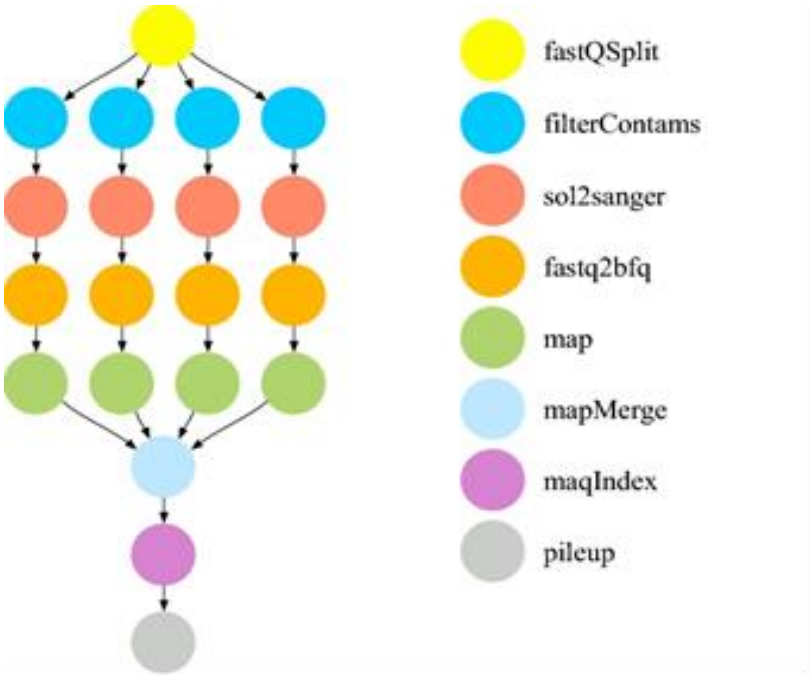
Η εφαρμογή Montage δημιουργήθηκε από τη NASA/IPAC και ενώνει μαζί πολλαπλές εικόνες για δημιουργία προσαρμοσμένων ψηφιδωτών του ουρανού. Παρέχει επιστημονικά ψηφιακά μωσαϊκά του ουρανού στην κοινότητα που αποτελείται τόσο από επαγγελματίες όσο και από ερασιτέχνες αστρονόμους.



Εικόνα 3.3. Το παραπάνω στιγμιότυπο απεικονίζει μία ροή εργασιών τύπου Montage.

- Epigenomics [21]

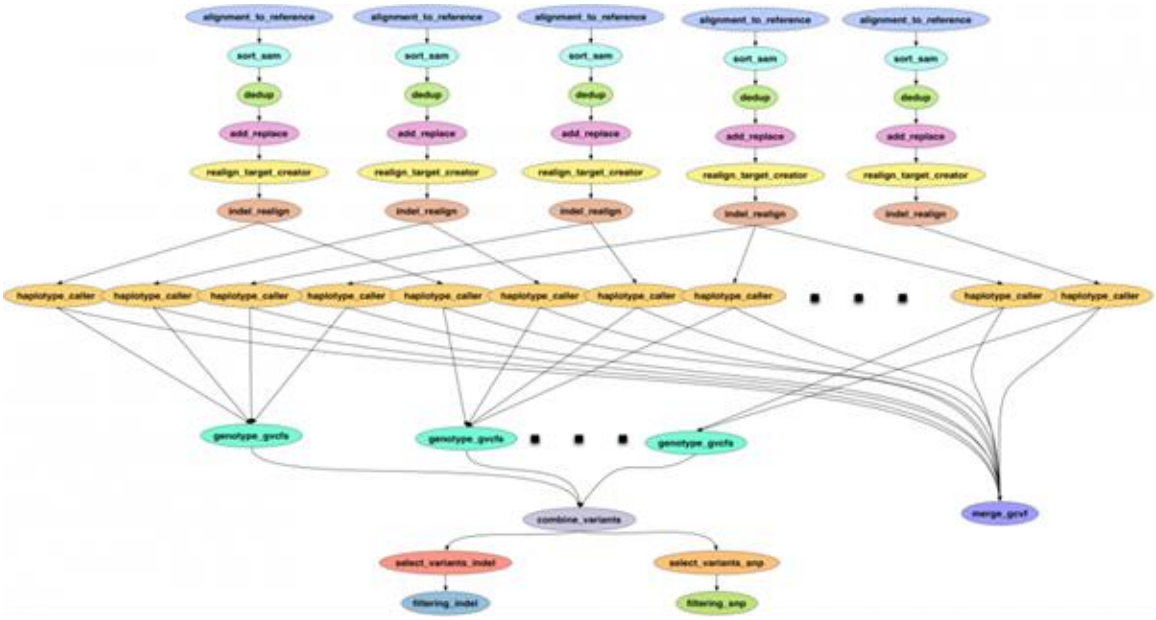
Η ροή εργασιών της επιγονιδιωματικής δημιουργήθηκε από το κέντρο επιγονιδιώματος USC.



Εικόνα 3.4. Το παραπάνω στιγμιότυπο απεικονίζει μία ροή εργασιών τύπου Epigenomics.

- SoyKB [22]

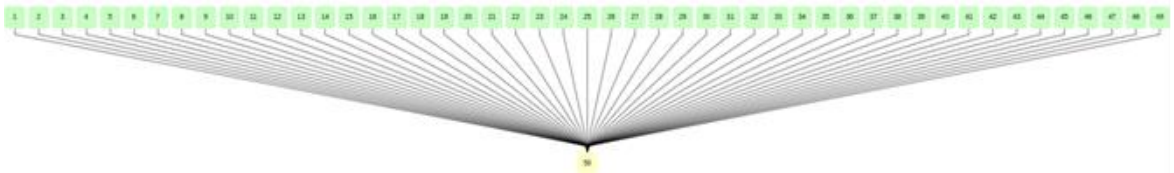
Δεδομένα αναδιάταξης NGS αντιπροσωπεύουν μια πηγή πληροφοριών και μπορούν να οδηγήσουν σε σημαντικές ανακαλύψεις όταν πρόκειται για εξόρυξη γονοτυπικών δεδομένων. Τα δεδομένα και τα αποτελέσματα είναι προσβάσιμα μέσω της Βάσης Σόγιας (SoyKB).



Εικόνα 3.5. Το παραπάνω στιγμιότυπο απεικονίζει μία ροή εργασιών τύπου SoyKB.

- Seismology [23]

Η σεισμολογία γνωρίζει μια πρόσφατη άνθηση, η οποία οφείλεται εν μέρει από μεγάλους όγκους διαθέσιμων δεδομένων. Ωστόσο, υπάρχουν σημαντικά εμπόδια στην επεξεργασία μεγάλων όγκων δεδομένων, όπως μεγάλοι χρόνοι ανάκτησης από αποθήκες δεδομένων, πολύπλοκη διαχείριση δεδομένων και περιορισμένοι υπολογιστικοί πόροι.

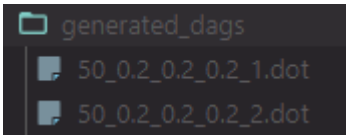


Εικόνα 3.6. Το παραπάνω στιγμιότυπο απεικονίζει μία ροή εργασιών τύπου Seismology που δημιουργήθηκε με προσαρμοσμένο εργαλείο οπτικοποίησης.

Περισυλλογή δεδομένων 3.3

Όπως προαναφέρθηκε από τα παραχθείσα Custom generated αρχεία αυτά τα οποία κρίθηκαν πιο αντιπροσωπευτικά ήταν σε αριθμό 252. Τα συγκεκριμένα αρχεία έχουν επέκταση ονόματος αρχείου .dot. Το DOT είναι μια γλώσσα περιγραφής γραφήματος. Γράφτηκε κώδικας ο οποίος αναγνωρίζει την επέκταση αυτή και δημιουργεί τον γράφο και τις ακμές του βασισμένο στα στοιχεία που έχουν παραχθεί εντός του εκάστοτε αρχείο με επέκταση .dot.

Κάθε αρχείο φέρει μία ονομασία ξεχωριστή. Από το όνομα του αρχείο μπορεί κανείς να καταλάβει και τα ορίσματα που δόθηκαν για την δημιουργία αυτού.



Εικόνα 3.7. Στιγμιότυπο από τα παραχθείσα αρχεία που δείχνει τον περιβάλλοντα φάκελο και τον τρόπο ονομασία τους.

Αν και από την εικόνα 3.7 παρατηρείται ότι δεν είναι εύκολο κανείς να διακρίνει εκ πρώτης άποψης πιο όρισμα αντιστοιχεί σε ποιο. Κρίθηκε όμως πιο σωστό να αποτυπωθούν με τον παραπάνω τρόπο καθώς επιτρέπει έναν πιο εύκολο τρόπο αναγνώρισης συγκεκριμένων στοιχείων που έχουν μεγαλύτερη επιρροή στον τελικό γράφο. Χωρίς να δημιουργούνται δυσανάγνωστα μεγάλες γραμμές ονομάτων. Παραδείγματος χάρη ο πρώτος αριθμός υποδεικνύει τον αριθμό των κόμβων, ενώ ο τελευταίος τα πιθανά παιδιά/ακμές που ενδέχεται να έχει ένας κόμβος. Τέλος τα ορίσματα έχουν αποτυπωθεί με την σειρά που εισήχθησαν και στο εργαλείο παραγωγή τους.

Παρομοίως για την κατηγορία των ρεαλιστικών δεδομένων αφού δημιουργηθούν με την βιβλιοθήκη WfCommons της γλώσσας Python, αργότερα το πρόγραμμα διαβάζει τα παραχθέντα αρχεία τα οποία έχουν επέκταση ονόματος .json και βάση αυτών δημιουργεί τους ανάλογους γράφους. Τα συγκεκριμένα αρχεία κρύβουν μία ιδιαιτερότητα σε ότι αφορά της ακμές τους. Τα βάρη των ακμών δεν δίνονται «απλόχερα». Πιο συγκεκριμένα μια εργασία έχει την εξής μορφή εντός των αρχείων:

- **name**: Όνομα εργασίας. Επίσης λειτουργεί και ως αναγνωριστικό καθώς αποτελεί μοναδικό εντός μίας ροής.

- **type**: Έχει να κάνει με τον τύπο της διεργασίας που εκτελεί η εργασία, η συγκεκριμένη μεταβλητή αγνοείται καθώς δεν προσφέρει κάποια αξιόλογη πληροφορία για την μελέτη αυτής εργασίας.

- **runtime**: Τον τυπικό χρόνο που χρειάζεται για να τρέξει η διεργασία η μονάδα μέτρησης σύμφωνα με την WfCommons είναι τα seconds.

- **parents**: Πίνακας στοιχείων όπου αναγράφονται ονομαστικά τα ονόματα των εργασιών που έχουν ακμές προς στην συγκεκριμένη εργασία.

- **children**: Πίνακας στοιχείων όπου αναγράφονται ονομαστικά τα ονόματα των εργασιών που δείχνουν οι ακμές της συγκεκριμένης εργασίας.

- **files**: Πίνακας στοιχείων που περιλαμβάνει κάθε αρχείο που εισέρχεται και εξέρχεται της εργασίας.

Εικόνα 3.8. Στιγμιότυπο από τα παραχθείσα αρχεία που δείχνει τον μορφή ενός στοιχείου τύπου .json της βιβλιοθήκης WfCommons.

Ο συγκεκριμένος πίνακας **files** αποτελεί και το πιο προβληματικό όσον αφορά προς την περισυλλογή των συγκεκριμένων τύπων δεδομένων. Αυτό συμβαίνει γιατί παρατηρώντας και από την εικόνα 3.8 ο πίνακας απλά υποδεικνύει ένα random hash ως όνομα των αρχείων καθώς και το μέγεθος τους σε KBs. Ο τρόπος με τον οποίο αντιμετωπίστηκε το παραπάνω πρόβλημα ήταν να ανατρέξει ο αλγόριθμος για κάθε όνομα αρχείου που υπάρχει μέσα στον πίνακα και να ελέγξει αν υπάρχει το αντίστοιχο αρχείο στον αντίστοιχο πίνακα που υπάρχει εντός των εργασιών που είναι είτε παιδιά της εργασίας είτε γονείς. Για να μειωθεί λίγο η πολυπλοκότητα και κατ' επέκταση και ο χρόνος εκτέλεσης αυτής της διαδικασίας ο αλγόριθμος κοιτάει την μεταβλητή **link** βάση αυτής ορίζει εάν θα γίνει έλεγχος για το αρχείο εντός των εργασιών παιδιών οι των εργασιών του γονέα και μόνο. Όταν η παραπάνω μεταβλητή έχει την τιμή input τότε γίνεται έλεγχος στις εργασίες γονείς καθώς πρόκειται για εισερχόμενο αρχείο. Αντίστοιχα όταν έχει την τιμή output γίνεται έλεγχος στις εργασίες παιδιά.

Όσο αφορά τους υπολογιστικούς πόρους του συστήματος, δίνονται από τον προγραμματιστή αν αυτός επιθυμεί. Συγκεκριμένα το προκαθορισμένο σύστημα αποτελείται από 4 υπολογιστικούς πόρους οι οποίοι διαφέρουν μεταξύ τους μόνο στον αριθμό των επεξεργαστών. Ακόμα σημαντικό είναι να σημειωθεί ότι βρίσκονται κάτω από το ίδιο δίκτυο με ταχύτητα μεταφοράς αρχείων στα 500Mbs. Αργότερα θα υπάρξουν πολλές παραλλαγές συστημάτων στις οποίες θα διαφοροποιούνται οι δυναμικές των υπολογιστικών πόρων καθώς και τα δίκτυα επικοινωνίας μεταξύ τους.

Εγκυροποίηση και τροποποίηση δεδομένων 3.4

Όπως φάνηκε και από το κεφάλαιο της περισυλλογής των δεδομένων, τα δεδομένα υποχρεώνονται να έχουν πολύ συγκεκριμένες μορφές όσο αφορά τους γράφους που αποτυπώνουν. Το κάθε εργαλείο χρειάστηκε διαφορετικούς τρόπους εγκυροποίησης των δεδομένων που είχε δημιουργήσει. Πιο συγκεκριμένα το εργαλείο που δημιούργησε τα Custom generated .dot αρχεία βάση των ορισμάτων που του δίνονται ενδέχεται να δημιουργήσει συνθέσεις γράφων που δεν συνδέονται όλοι ο κόμβοι ή να δημιουργήσει γράφους στους οποίους να υπάρχουν κόμβοι οι οποίοι μπορεί να δείχνουν στον εαυτό τους.

Για να αντιμετωπιστεί αυτό έγινε έλεγχος εγκυρότητας κάθε αρχείου που χρησιμοποιήθηκε από το συγκεκριμένο εργαλείο για τα εξής χαρακτηριστικά: α) Οι γράφοι να είναι όντως άκυκλοι β) Κάθε κόμβος πάνω στον γράφο έχει τουλάχιστον έναν γονέα ή έναν κόμβο παιδί.

Όσο αφορά τα αρχεία από το εργαλείο WfCommons η μορφή του αρχείου έχει να κάνει αυστηρά με τα πρότυπα και τους κανονισμούς που έχει θέσει η βιβλιοθήκη. Επομένως δεν υπήρξε λόγος ελέγχου όσο αφορά την μορφή που έχει ένας γράφος. Υπήρξε όμως μία περίπτωση στην οποία χρειάστηκε να ανοίξει και ένα issue στο github στο επίσημο repo της βιβλιοθήκης. Η βιβλιοθήκη όταν δημιουργούσε ροές εργασιών τύπου Montage τα βάρη των ακμών αυτών δεν ήταν πάντα σωστά. Συγκεκριμένα η αναμενόμενη συμπεριφορά ήταν το άθροισμα των τιμών των αρχείων σε μία εργασία Α να είναι το ίδιο με το άθροισμα των αρχείων αυτών που δέχεται και μόνο μία εργασία Β. Μετά όμως από εξονυχιστικό έλεγχο και των προγραμματιστών της βιβλιοθήκης δεν βρέθηκε τρόπος αναπαραγωγής του σφάλματος. Θεωρήθηκε λοιπόν ότι ίσως ήταν συνδυαστικό το πρόβλημα και το συντριπτικό ποσοστό των γράφων που είχαν μορφή τύπου Montage αφέθηκαν εκτός. [24]

Τροποποίηση δεδομένων 3.5

Η τροποποίηση των δεδομένων παραμένει κοινή και για τους δύο τύπους δεδομένων με την σημαντική διαφορά ότι κατά την ανάγνωση των αρχείων τύπου .dot τα βάρη των ακμών παράγονται στο runtime του προγράμματος. Μία πιο αποδοτική και παράλληλα πιο σωστή λύση θα ήταν αυτά τα βάρη να αποτυπωθούν στα αρχεία μέσα χωρίς να χρειάζεται να παράγονται κάθε φορά.

Η κύρια τροποποίηση που λαμβάνει χώρα είναι η μετατροπή των αρχείων σε δομές οι οποίες είναι κατάλληλες για επεξεργασία από τους αλγορίθμους χρόνο-δρομολόγησης. Η οποία χωρίζεται σε στάδια:

- Μετατροπή κόμβων γράφου σε ανάλογες κλάσεις.

Ο χρήστης μπορεί να ορίσει μία θέση αρχείου από την οποία θα παρθεί ο εκάστοτε γράφος. Κατά την ανάγνωση του γράφου δημιουργούνται οι ανάλογες δομές. Κάθε

κόμβος στο σύστημα μεταφράζεται ως ένα αντικείμενο τύπου Task. Καθώς και κάθε ακμή σε αντικείμενο τύπου Edge. Μέσα σε κάθε αντικείμενο Task εμπεριέχονται όλες οι πιθανές ακμές που έχει είτε είναι από τους γονείς του είτε προς τα παιδιά του. Αυτό επιτρέπει μία πολύ εύχρηστη διάβαση από τον έναν κόμβο σε έναν άλλον. Επίσης βάση των ακμών που δημιουργήθηκαν ανατίθεται σε κάθε κόμβο δύο μεταβλητές τύπου boolean η οποίες υποδεικνύουν για το αν ένας κόμβος δεν έχει κόμβους παιδιά (exit-node) και για το αν ένας κόμβος δεν έχει κόμβους γονείς (entry-node). Τέλος γίνεται έλεγχος για κάθε κόμβο για το εάν είναι συνδεδεμένος γενικότερα στον υπόλοιπο γράφο. Εάν δεν είναι το πρόγραμμα δείχνει το ανάλογο error message. Αυτό γίνεται γιατί οι γράφοι με τους οποίους πραγματεύεται υποχρεωτικά πρέπει έχουν μία συγκεκριμένη μορφή.

- Δημιουργία Dummy Nodes. [25]

Για την διευκόλυνση της λειτουργίας των αλγορίθμων καθώς και για την πιο εύχρηστη αποσφαλμάτωση, για όλους τους κόμβους που είναι entry-node δημιουργείται μία ακμή η οποία είναι κατευθυνόμενη στους entry κόμβους από έναν κόμβο με ονομασία ‘Dummy-In’. Πρόκειται για κόμβο με βοηθητική ιδιότητα. Το υπολογιστικό κόστος του είναι ίσων με 0 καθώς και κάθε ακμή από αυτών σε κάποιον άλλον κόμβο έχει μηδενικό βάρος. Η κύρια αιτία δημιουργίας του είναι ότι από αυτόν υπάρχει εύκολη διάβαση σε όλους στους entry-nodes. Αντίστοιχα δημιουργείται ένας ‘Dummy-Out’ κόμβος στον οποίο όλοι οι κόμβοι exit έχουν ακμές που κατευθύνονται προς αυτούς.

- Υπολογισμός του ccr (λόγος επικοινωνίας προς υπολογισμό). [26]

Καθώς στα ρεαλιστικά δεδομένα δεν υπάρχει η ένδειξη ccr, υπολογίζεται ως ο λόγος του μέσου επικοινωνιακού κόστους όλων των κόμβων προς τον μέσο όρο όλου του υπολογιστικού όρου.

- Δημιουργία αντικειμένου Workflow.

Ο τελικός γράφος παίρνει την μορφή ενός αντικειμένου τύπου Workflow. Το συγκεκριμένο αντικείμενο είναι υπεύθυνο για μεταβλητές και μεθόδους που σχετίζονται με μία ροή εργασιών. Κάποιες από τις μεταβλητές που βρίσκονται εντός του αντικειμένου είναι, το σύνολο των κόμβων/εργασιών, ο συνολικός χρόνος εκτέλεσης της ροής, ο τύπος της ροής, το όνομα της, ένα μοναδικό αναγνωριστικό, το σημείο από το οποίο επιλέχθηκε το αρχείο και άλλες. Ενώ κάποιες από τις μεθόδους είναι μεταξύ άλλων, η μέθοδος που υπολογίζει το ccr, η μέθοδος που βρίσκει και αναθέτει τα επίπεδα σε όλες τις εργασίες (level order), η μέθοδος που βρίσκει όλες τις εργασίες που είναι σε ετοιμότητα και τους επιτρέπεται να ξεκινήσουν βάση των ορισμάτων που έχουν προαναφερθεί, διάφορες στατικές μέθοδοι που διευκολύνουν στην παραγωγή συγκεκριμένων παραδειγμάτων ροών εργασιών, διάφορες μέθοδοι που σχετίζονται με τροποποιήσεις στην δομή μίας ροής καθώς μπορεί να αποτελεί προϋπόθεση για μετέπειτα αλγόριθμους που θα χρησιμοποιηθούν αργότερα, και τέλος μέθοδοι οι οποίες έχουν να κάνουν με διαφορετικούς τρόπους διάβασης της ροής.

```python
if tasks is None:
    # 1. Parse the workflow tasks
    if file_path.endswith(".dot"):
        self.tasks = self.read_dag()
    else:
        self.tasks = get_tasks_from_json_file(file_path, id_, self.machines[0].network_kbps)

    # 2. add dummy nodes
    if add_dummies:
        self.__add_dummy_nodes()

#3. Calc ccr which means also the avg_comp and avg_com costs
self.ccr = self.calc_ccr()
```

Εικόνα 3.9. Στιγμιότυπο από κώδικα Python που δείχνει τα βήματα που γίνονται εντός της κλάσης Workflow κατά την ανάγνωση μίας ροής εργασιών από ένα αρχείο.

Αλγόριθμοι μονής ροής εργασιών

Στο παρόν κεφάλαιο θα γίνει ανάλυση των αλγορίθμων που προτάθηκαν τόσο από την βιβλιογραφία όσο και από την παρούσα εργασία. Ποιο συγκεκριμένα θα γίνει αναφορά στους αλγόριθμους που προαναφέρθηκαν όπως: HEFT, CPOP, Holes scheduling, Ruin and Recreate. Οι αλγόριθμοι που θα παρουσιαστούν παρακάτω θα ενδέχεται να έχουν κάποιες τροποποιήσεις όχι σε ότι αφορά στο αποτέλεσμα που παράγουν αλλά σε κάποια διαδικαστικά κομμάτια που εντάχθηκαν η άλλαξαν για να διευκολυνθεί η ροή του προγράμματος. Ακόμα θα εξεταστούν περιπτώσεις στις οποίες κάποιος αλγόριθμος ίσως αποδίδει καλύτερα συγκριτικά με τους υπόλοιπους και τα θετικά του να εφαρμοστεί κάποιος αλγόριθμος σε συγκεκριμένες περιπτώσεις. Ακόμα αργότερα θα γίνει σύγκριση αυτών όσο αφορά την αποτελεσματικότητα τους σε ένα κατανεμημένο περιβάλλον. Ενώ θα υπάρξει και οπτικοποίηση κάποιων στατιστικών αποτελεσμάτων για τους αλγορίθμους που έτρεξαν.

HEFT 4.1

Ο Heterogeneous Early Finish Time (ή HEFT) είναι μια ευρετική μέθοδος για τον προγραμματισμό ενός συνόλου εξαρτημένων εργασιών σε ένα δίκτυο ετερογενών εργαζομένων λαμβάνοντας υπόψη τον χρόνο επικοινωνίας. Για εισόδους, ο HEFT λαμβάνει μία ροή εργασιών, που αντιπροσωπεύεται ως κατευθυνόμενος άκυκλος γράφος, ένα σύνολο εργαζομένων (ή οι υπολογιστικοί πόροι στην προκειμένη περίπτωση), τους χρόνους εκτέλεσης κάθε εργασίας σε κάθε εργαζόμενο και τους χρόνους για την κοινοποίηση των αποτελεσμάτων από κάθε εργασία σε καθένα από τα παιδιά της μεταξύ κάθε ζεύγους εργατών. Προέρχεται από αλγόριθμους προγραμματισμού λιστών. Επομένως εκ φύσεως ο αλγόριθμος είναι άπληστος.

Ο HEFT είναι ένας αλγόριθμος με μεγάλη δημοτικότητα και έχει λάβει τον σεβασμό μεταξύ των ευρετικών αλγορίθμων για το πρόβλημα της χρόνο-δρομολόγησης. Ωστόσο, σε περίπλοκες καταστάσεις μπορεί εύκολα να αποτύχει να βρει τον βέλτιστο προγραμματισμό. Ο HEFT είναι ουσιαστικά ένας άπληστος αλγόριθμος και ανίκανος να κάνει βραχυπρόθεσμες θυσίες για μακροπρόθεσμα οφέλη. Μια τροποποίηση του αλγορίθμου που κοιτάζει μπροστά για την καλύτερη εκτίμηση της ποιότητας μιας απόφασης προγραμματισμού μπορεί να χρησιμοποιηθεί για την ανταλλαγή του χρόνου εκτέλεσης με την απόδοση προγραμματισμού. [27]

Ο αλγόριθμος μπορεί να χωριστεί σε δύο βασικά διακριτά βήματα:

1. Task Prioritizing Phase
2. Processor Selection Phase

Task Selection

Στην πρώτη φάση δίνεται προτεραιότητα σε κάθε εργασία. Η προτεραιότητα κάθε εργασίας n_i συνήθως ορίζεται ως η "ανοδική κατάταξη" της ή αλλιώς upward-rank, η

οποία ορίζεται αναδρομικά ως εξής:

$$rank_u(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)} (\overline{c_{i,j}} + rank_u(n_j)),$$

Εικόνα 4.1. Στιγμιότυπο μαθηματικού τύπου για το πως ορίζεται το upward-rank από το άρθρο των Topcuoglu, Haluk; Hariri, Salim; Wu, M. (2002).

Όπου το n_i αντιπροσωπεύει την εργασία, $\overline{w_i}$ είναι ένα μέσο κόστος υπολογισμού της εργασίας i μεταξύ όλων των υπολογιστικών πόρων, $succ(n_i)$ είναι το σύνολο όλων των εργασιών που εξαρτώνται άμεσα από την εργασία n_i , και $\overline{c_{i,j}}$ είναι το μέσο κόστος επικοινωνίας των δεδομένων που μεταφέρονται μεταξύ των εργασιών n_i και n_jμεταξύ όλων των ζευγών υπολογιστικών πόρων. Σημείωση ότι ο υπολογισμός του $rank_u(n_i)$ εξαρτάται από τον υπολογισμό της κατάταξης όλων των παιδιών του. Η ανοδική κατάταξη προορίζεται να αντιπροσωπεύει την αναμενόμενη απόσταση οποιασδήποτε εργασίας από το τέλος του υπολογισμού. Για μέσες ποσότητες όπως το $\overline{w_i}$ διαφορετικοί μέσοι όροι ενδέχεται να παρέχουν διαφορετικά αποτελέσματα. [28]
Η παραπάνω συνάρτηση σε κώδικα Python αποτυπώθηκε ως εξής:

```python
def avg(arr):
    return sum(arr) / len(arr)


def calculate_upward_ranks(tasks):
    def calc_upward(curr_task):
        # Base condition once your set
        # is empty just return the computation cost
        if curr_task.is_exit:
            curr_task.up_rank = avg(curr_task.costs)
            return curr_task.up_rank

        curr_task.up_rank = avg(curr_task.costs) + max(e.weight + calc_upward(e.node) for e in curr_task.children_edges)
        return curr_task.up_rank

    calc_upward(tasks[0])
    return tasks
```

Εικόνα 4.2. Στιγμιότυπο κώδικα Python που υποδεικνύει έναν τρόπο εφαρμογής της συνάρτησης του αλγορίθμου του HEFT που ορίζει το upward-rank μίας εργασίας.

Αρχικά ορίζεται μια βοηθητική συνάρτηση **avg** η οποία υπολογίζει την μέση τιμή από έναν πίνακα αριθμών. Αργότερα καλείται μία αναδρομική συνάρτηση μέσα από την calculate_upward_ranks κρατώντας έτσι το scope της εντός την

προαναφερθέντας συνάρτησης. Σαν τερματικό βήμα της αναδρομικής συνάρτησης υπάρχει το εξής: ‘Αν η εργασία που έχει πάρει αυτή την στιγμή σαν παράμετρο η συνάρτηση είναι exit-task (δηλαδή δεν έχει παιδιά) τότε το up_rank της συγκεκριμένης εργασίας είναι ίσο με τον μέσο όρο του υπολογιστικού κόστους ανάμεσα σε όλους τους πόρους. Όπου costs είναι πίνακας ο οποίος δείχνει το πόσο χρόνο χρειάζεται μία εργασία να τρέξει στο εκάστοτε μηχάνημα. Έπειτα για να υπολογιστεί το up_rank, καλείται αναδρομικά πάλι η συνάρτηση calc_upward για κάθε παιδί της συγκεκριμένης εργασίας, προσθέτοντας παράλληλα το βάρος της ακμής τους και το μέσο όρο του υπολογιστικού κόστους ανάμεσα σε όλους τους πόρους.

Αφού υπολογιστούν τα up-ranks των εργασιών γίνεται ταξινόμηση αυτών με βάση το up-rank με φθίνουσα σειρά. Έπειτα υπάρχει ένας βασικός βρόχος όπου γίνεται επιλογή μίας εργασίας. Η εργασία που επιλέγεται κάθε φορά είναι αυτή με το μεγαλύτερο up-rank. Σε περίπτωση ισοτιμίας επιλέγεται τυχαία μία από της δύο.

Machine Selection

Στη δεύτερη φάση ανατίθενται καθήκοντα στους υπολογιστικούς πόρους. Τώρα που όλες οι εργασίες έχουν προτεραιότητα, γίνεται εξέταση και προγραμματίζεται η καθεμία, ξεκινώντας με την υψηλότερη προτεραιότητα. Η εργασία με την υψηλότερη προτεραιότητα για την οποία έχουν ολοκληρωθεί όλες οι εξαρτώμενες εργασίες προγραμματίζεται στον υπολογιστικό πόρο που θα έχει ως αποτέλεσμα τον νωρίτερο χρόνο τερματισμού αυτής της εργασίας. Αυτός ο χρόνος τερματισμού εξαρτάται από το χρόνο επικοινωνίας για την αποστολή όλων των απαραίτητων εισόδων στον πόρο, τον υπολογιστικό χρόνο της εργασίας στον πόρο και τον χρόνο που αυτός ο επεξεργαστής γίνεται διαθέσιμος (μπορεί να είναι απασχολημένος με άλλη εργασία). Ο HEFT χρησιμοποιεί μια πολιτική που βασίζεται σε εισαγωγή που καλύπτει επαρκώς μεγάλα κενά μεταξύ των ήδη προγραμματισμένων εργασιών.

Πιο συγκεκριμένα αφού έχει επιλεχθεί μία εργασία ο αλγόριθμος έπειτα προσπαθεί να βρει το κατάλληλο υπολογιστικό πόρο στον οποίο θα τρέξει η εργασία. Αυτό γίνεται βάση του ορίσματος EFT (Earliest Finish Time). Το οποίο μεταφράζεται σαν διαδικασία ως εξής: ‘Να γίνει επιλογή του υπολογιστικού πόρου στον οποίο εάν τρέξει η εκάστοτε εργασία να ενδέχεται να τελειώσει το συντομότερο συγκριτικά με το να έτρεχε σε οποιοδήποτε άλλο υπολογιστικό πόρο.’

Μία λανθασμένη εντύπωση που μπορεί να δημιουργηθεί βάση του παραπάνω ορισμού είναι να γίνεται κυρίως επιλογή του πιο γρήγορου υπολογιστικού πόρου. Αυτό όμως παύει να ισχύει όταν λαμβάνουμε υπόψη το επικοινωνιακό κόστος από τον έναν πόρο στον άλλον και το γεγονός ότι παρόλο που μία εργασία έχει αργότερη γενικά εκτέλεση σε έναν πόρο Α σε σχέση με έναν Β, ο Α μπορεί να επιτρέπει στην εργασία να τελειώνει την χρονική στιγμή 20 με χρόνο εκτέλεσης της εργασίας μέσα στον Α να ισούται με 15 ενώ στον πόρο Β παρόλο που μπορεί να έχει χρόνο εκτέλεσης 8 να τελειώνει την χρονική στιγμή 25. Από τους δύο πόρους ο αλγόριθμος βασισμένος στο όρισμα EFT θα επέλεγε τον υπολογιστικό πόρο Α καθώς επιτρέπει την ταχύτερη εκτέλεση της εργασίας γενικότερα.

Το πως υπολογίζεται η χρονική στιγμή στην οποία αρχίζει και η χρονική στιγμή στην οποία τελειώνει την εκτέλεση της μία εργασία μέσα σε έναν πόρο υπολογίζεται βάση συγκεκριμένων προϋποθέσεων. Αρχικά μία εργασία όπως έχει οριστεί παραπάνω θα πρέπει να περιμένει όλους τους γονείς να τελειώσουν την εκτέλεση τους αλλά και να μεταφέρουν και τα απαραίτητα δεδομένα σε αυτήν για να μπορέσει να ξεκινήσει. Επειδή όμως από όλες τις εργασίες γονείς μία θα καθυστερήσει το ξεκίνημα περισσότερο από τις υπόλοιπες αρκεί να κρατήσουμε αυτή σαν τον αργότερο γονέα για να υπολογίσουμε το ξεκίνημα της. Η συγκεκριμένη διαδικασία δεν αποτελεί μέρος του αλγορίθμου αλλά εντάχθηκε από την παρούσα εργασία με σκοπό να μειωθεί η συνολική πολυπλοκότητα.

```python
# Running this function means we already have
# as a fact that all the parent tasks are done
# before we get in here and check for the slowest parent
def compute_execution_time(task, m_id, potensial_start_time):
    """Find the execution time of a task on a machine based on given start_time.

    Parameters
    ----------
    task : Task
        The task we want to find the execution time for.
    m_id : int
        Machine id.
    potensial_start_time : float
        This can be either the current time of a machine or the start time of a hole in a machine.

    Returns
    -------
    Tuple[float, float]
        The potential start time and end time of the task on the machine.
    """
    # If between the child and the parent the machine doesn't change
    # then the communication cost is 0.
    if task.is_entry:
        return [potensial_start_time, potensial_start_time + task.costs[m_id]]
    elif task.slowest_parent['parent_task'].machine_id == m_id:
        communication_time = 0
    else:
        # The com_time was calculated at the parsing phase.
        communication_time = task.slowest_parent['communication_time']

    #  parent_end + communication_time + cost_of_task_in_this_machine
    # We pick the one that is bigger so we can ensure the child task starts
    # after the parent.
    start = max(task.slowest_parent['parent_task'].end + communication_time, potensial_start_time)
    end = start + task.costs[m_id]
    return [start, end]
```

Εικόνα 4.3. Στιγμιότυπο κώδικα Python που υποδεικνύει έναν τρόπο εφαρμογής της συνάρτησης του αλγορίθμου του HEFT που επιλέγει κατάλληλο υπολογιστικό πόρο για την εργασία.

Έχοντας σαν βοηθητικό την τεκμηρίωση που βρίσκεται και σαν σχόλιο στην συνάρτηση παρατηρεί κανείς την παράμετρο potensial_start_time, η οποία είναι ο τρέχων χρόνος που έχει ο υπολογιστικός πόρος. Ο χρόνος αυτός πρακτικά ισούται

με το τέλος τις τελευταίας εργασίας που έχει προγραμματιστεί να τρέξει στον συγκεκριμένο υπολογιστικό πόρο αυτό.

Ακόμα αν η εργασία είναι entry-task (δεν έχει εργασίες γονείς) τότε η πιθανή αρχή της θα είναι το ήδη υπάρχων potensial_start_time του πόρου και σαν πιθανό τέλος θα είναι η αρχή της συν το υπολογιστικό κόστος που θα χρειαστεί στον πόρο που εξετάζουμε. Αυτό ισχύει καθώς σαν entry-task δεν έχει κάποιον γονέα από τον οποίο θα εξαρτιέται επομένως μπορεί να ξεκινήσει όσο τον δυνατόν νωρίτερα το οποίο ορίζεται από την μεταβλητή potensial_start_time. Αν όμως η εργασία έχει γονείς εργασίες και κατ' επέκταση δεν είναι entry-task τότε υπάρχει ένας έλεγχος για το αν ο γονέας βρίσκεται στον ίδιο υπολογιστικό πόρο με την εργασία που εξετάζεται αυτή την στιγμή. Αν η εργασία γονέας είναι στον ίδιο υπολογιστικό πόρο τότε ουσιαστικά δεν υπάρχει επικοινωνιακό κόστος καθώς ο γονέας δεν χρειάζεται να μεταφέρει δεδομένα.

Αλλιώς αν η εργασία γονέας βρίσκεται σε διαφορετικό υπολογιστικό πόρο θα πρέπει να ληφθεί υπόψιν και όταν θα γίνει υπολογισμός για το ξεκίνημα της εργασίας ενδέχεται να επηρεαστεί η αρχή της, καθώς θα περιμένει τον γονέα να στείλει τα δεδομένα.

Τέλος βάση όλων των παραπάνω σαν πιθανή αρχή της εργασίας μπορεί να οριστεί, το τέλος της αργότερης εργασίας γονέα συν τον χρόνο για την μεταφορά δεδομένων που μπορεί να χρειαστεί αν βρίσκονται σε διαφορετικούς πόρους οι δύο εργασίες. Αν όμως ο πιθανός χρόνος που παράγεται βάση της παραπάνω εξίσωσης είναι μικρότερος από τον τρέχων ή από τον πιθανόν τρέχων χρόνο εντός του υπολογιστικού πόρου, τότε επιλέγεται ο δεύτερος εκ των δύο.

Αφού γίνει υπολογισμός των πιθανών χρόνων μίας εργασίας επιλέγεται ο καλύτερος υπολογιστικός πόρος βάση του ορίσματος που έχει δηλωθεί. Στην προκειμένη περίπτωση επειδή γίνεται αναφορά στον αλγόριθμο HEFT το όρισμα αυτό θα ισούται με EFT (Earliest Finish Time) δηλαδή θα γίνει επιλογή να γίνει χρόνο-δρομολόγηση της εργασίας στον υπολογιστικό πόρο ο οποίος θα μπορέσει να δημιουργήσει τον γρηγορότερο πιθανό χρόνο εκτέλεσης εργασίας συμπεριλαμβανομένου και των υπολοίπων που έχουν χρόνο-δρομολογηθεί μέχρι εκείνη την χρονική στιγμή.

Σε ψευδοκώδικα η διαδικασία θα ήταν η εξής:

Υπολογισμός up-rank κάθε εργασίας.
Ταξινόμηση των εργασιών βάση του up-rank τους.
while υπάρχουν unscheduled tasks στην λίστα:
    task = Επιλογή της πρώτης εργασίας
    for each machine:
        Υπολογισμός πιθανών χρόνων εκκίνησης και τερματισμού εργασίας αν είχε επιλεχθεί να χρόνο-δρομολογηθεί στο συγκεκριμένο πόρο.
        task.machine_id = Ανάθεση πόρου σε εργασία που εξυπηρετεί βέλτιστα το όρισμα EFT.
endwhile

CPOP 4.2

Ο αλγόριθμος CPOP (Critical-Path-on-a-Processor) ανήκει στην κατηγορία, όπως και ο HEFT, heuristic (ευρετικός). Ενώ οι ομοιότητες των δύο είναι εμφανής διαφοροποιείται του HEFT. Κυρίως εξαιτίας τις προετοιμασίας που λαμβάνει χώρο πριν την έναρξη της χρόνο-δρομολόγησης για την επιλογή της κατάλληλης σειράς χρόνο-δρομολόγησης εργασιών. Καθώς όμως και λόγω του διαφορετικού ορίσματος που λαμβάνει υπόψιν για να κάνει την προτεραιότητα μίας εργασίας. Προτάθηκε στο ίδιο άρθρο με τον HEFT.

Ο αλγόριθμος μπορεί να χωριστεί επίσης σε δύο διακριτά βήματα με τον HEFT.

1. Task Prioritizing Phase
2. Processor Selection Phase

Η ιδέα πίσω από τον CPOP βασίζεται στο γεγονός ότι: 'Αν γίνει ανάθεση ενός μονοπατιού εργασιών το οποίο ενδεικτικά επηρεάζει περισσότερο την χρόνο-δρομολόγηση σε σχέση με τα υπόλοιπα, στο πιο δυνατό υπολογιστικό πόρο αυτόματα μειώνεται ο πιθανός χρόνος εκτέλεσης αυτών ενώ το κόστος επικοινωνίας μεταξύ τους εκμηδενίζεται'. Το μονοπάτι αυτό στο άρθρο ονομάζεται Critical Path. Εξ αυτού και η ονομασία CPOP.

Task Selection

Όπως προαναφέρθηκε ο CPOP λειτουργεί γύρω από την ιδέα του κρίσιμου μονοπατιού. Για την κατασκευή του κρίσιμου μονοπατιού (Critical Path) το άρθρο ορίζει μία μεταβλητή με ονομασία priority για κάθε εργασία. Η μεταβλητή αυτή είναι συνδυασμός άλλων δύο ορισμάτων μίας εργασίας. Του upward-rank και του downward-rank της. Όπως και με τον HEFT γίνεται υπολογισμός του upward-rank των εργασιών με τον ίδιο τρόπο και με παρόμοιο λογική απλά με αντίστροφη διάβαση αυτή την φορά βρίσκεται η μεταβλητή downward-rank.

$$rank_d(n_i) = \max_{n_j \in pred(n_i)} \left\{ rank_d(n_j) + \overline{w_j} + \overline{c_{j,i}} \right\},$$

Εικόνα 4.4. Στιγμιότυπο μαθηματικού τύπου για το πως ορίζεται το downward-rank από το άρθρο των Topcuoglu, Haluk; Hariri, Salim; Wu, M. (2002).

Μία διαφορά που αξίζει να σημειωθεί είναι η μεταβλητή $pred(ni)$ η οποία είναι το σύνολο όλων των εργασιών που εξαρτιέται η εργασία ni.

Αφού βρεθούν τα upward-rank και downward-rank όλων των εργασιών αρκεί να γίνει πρόσθεση των δύο για να βρεθεί η μεταβλητή priority μίας εργασίας. Με βάση την παραπάνω μεταβλητή γίνεται ξεκινάει ένας βρόχος από ένα entry-task της ροής. Όπου θα υπάρχει μία λίστα η οποία θα ανανεώνεται κάθε φορά αν κριθεί ότι μία εργασία ανήκει στο κρίσιμο μονοπάτι. Πρώτο στοιχείο μέσα στην λίστα αποτελεί το entry-task όπου ξεκίνησε ο βρόχος.

Για να ανήκει μία εργασία στο κρίσιμο μονοπάτι οφείλει να έχει ίδιο priority με την αρχική εργασία. Αν δεν υπάρχει ένα παιδί και μόνο ένα παιδί με ίδιο priority σαν του υπόλοιπου κρίσιμου μονοπατιού τότε έχει γίνει λάθος είτε κατά το παρσάρισμα των εργασιών είτε κάποιο υπολογιστικό λάθος όσο αφορά τον υπολογισμό του down-rank ή του up-rank. Μόλις ο αλγόριθμος βρει ένα exit-task σημαίνει ότι το κρίσιμο μονοπάτι έχει ολοκληρωθεί.

```
# Find an entry task
entry_task = self.find_an_entry_task()
# Start from an entry task and run until you find an exit task.
# Then the critical path would be ready.
while temp_task.is_exit is False:
    for child_edge in temp_task.children_edges:
        diff: float = entry_priority - child_edge.node.priority
        if abs(diff) < 0.90:
            temp_task = child_edge.node
            critical_path.append(child_edge.node)
            break
exit_task = temp_task
```

Εικόνα 4.5. Στιγμιότυπο κώδικα Python που υποδεικνύει έναν πιθανό τρόπο δημιουργίας του κρίσιμου μονοπατιού του αλγορίθμου CPOP.

Μία παρατήρηση πιθανή είναι γιατί δεν γίνεται έλεγχος ισότητας ανάμεσα στις δύο προτεραιότητες ανάμεσα στις δύο εργασίες. Ο λόγος είναι γιατί όπως αναφέρθηκε και στο κεφάλαιο Τροποποίηση δεδομένων 3.5, επειδή υπάρχει μία κατηγορία δεδομένων που παραχθείσα μορφή τους απαιτεί προσθέσεις και συγκρίσεις για να βρεθεί το βάρος των ακμών πολλές φορές κατέληγε το πρόγραμμα να έχει απώλεια ακρίβειας λόγου του δυαδικού συστήματος. Αν γίνει εξαίρεση αυτής της ιδιαιτερότητας ο αλγόριθμος δεν έχει τροποποιηθεί αλλιώς.

Τέλος για την επιλογή μίας εργασίας επιλέγεται αυτή η οποία έχει μεγαλύτερο priority. Αυτό έχει σαν αποτέλεσμα να επιλέγονται κάθε φορά οι εργασίες που έχουν συνήθως την μεγαλύτερη επίδραση στο χρόνο-διάγραμμα όσο αναφορά τον χρόνο εκτέλεσης όλης της ροής.

Machine Selection

Αφού έχει δημιουργηθεί το κρίσιμο μονοπάτι γίνεται αναζήτηση για τον υπολογιστικό πόρο ο οποίος θα καταφέρει να μειώσει τον συνολικό χρόνο εκτέλεσης των εργασιών που βρίσκονται εντός του κρίσιμου μονοπατιού. Αν η εργασία που εξετάζεται πρόκειται για εργασία η οποία βρίσκεται εντός του κρίσιμου μονοπατιού, δηλαδή έχει max-priority, τότε γίνεται χρόνο-δρομολόγηση αυτής στον υπολογιστικό πόρο που έχει επιλεχθεί να τρέξει αυτές τις εργασίες. Στην περίπτωση που η επιλεχθείσα εργασία με το μεγαλύτερο priority δεν ανήκε εντός του κρίσιμου μονοπατιού γίνεται επιλογή του υπολογιστικού πόρου ο οποίος θα φέρει το καλύτερο αποτέλεσμα σχετικά με το χρονικό όρισμα που έχει δοθεί. Στον αλγόριθμο του CPOP γίνεται επιλογή του υπολογιστικού πόρου ο οποίος θα εκπληρώσει καλύτερα το EFT όπως στον αλγόριθμο του HEFT. Η διαδικασία λοιπόν και η μέθοδος επιλογής ενός

υπολογιστικού πόρου παραμένει η ίδια με αυτή του HEFT με την μόνη διαφορά ότι όταν η εργασία ανήκει στο κρίσιμο μονοπάτι θα πρέπει να χρόνο-δρομολογηθεί στον κρίσιμο υπολογιστικό πόρο που έχει αναθέσει ο αλγόριθμος για το κρίσιμο μονοπάτι.

Holes Scheduling 4.3

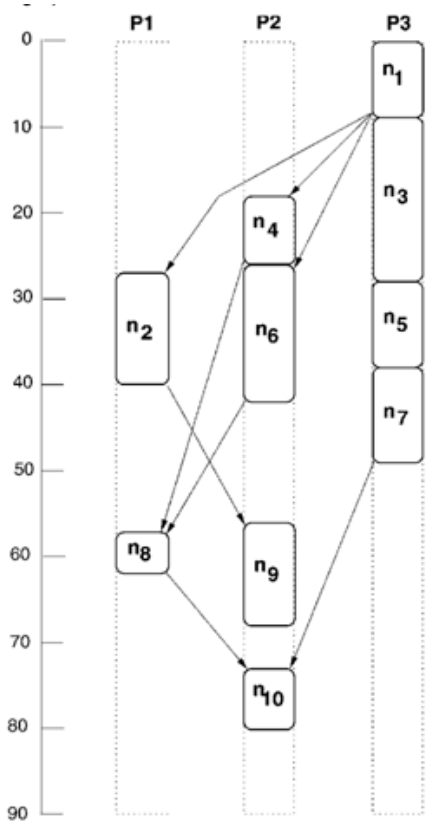
Για την επίλυση του παραπάνω προβλήματος μελετήθηκε μία λύση η οποία συνδυάζει τον θεμελιώδη αλγόριθμο του bin packing με τον αλγόριθμο του HEFT. Ο αλγόριθμος του bin packing έχει μελετηθεί εκτενώς. Μέσω της μελέτης αυτής έχει αποδειχτεί επανειλημμένα ότι έχει εφαρμογές σε πολλές περιπτώσεις και σε ποικίλες μορφές προβλημάτων. Ενώ παράλληλα έχει θεμελιώδη θεωρητική σημασία καθώς έθεσε το πρώιμο έδαφος για πολλές από τις κλασικές προσεγγίσεις για την ανάλυση της απόδοσης των αλγορίθμων προσέγγισης.

Επομένως βάση αυτής της ιδέας, έγινε μία νέα διατύπωση του προβλήματος. Πλέον τα διαστήματα που είναι αδρανεί σε έναν υπολογιστικό πόρο θεωρούνται χρονικά κενά. Ενώ κάθε φορά που γεμίζει ένα κενό ανεβαίνουν οι πιθανότητες να μειώνεται το τελικό χρονοδιάγραμμα. Αρχικά μελετήθηκαν τα κύρια αίτια που δημιουργούν χρονικά κενά.

Τα αίτια στα οποία οφείλεται η δημιουργία αυτών των χρονικών τρυπών διαφέρουν. Ένα πιθανό αίτιο συχνά μπορεί να έχει να κάνει με ένα πιθανό σφάλμα σε κάποιον υπολογιστικό πόρο. Εάν η εργασία που του είχε ανατεθεί αλλάξει υπολογιστικό πόρο για την εκτέλεση της τότε αυτός θα μείνει αναξιοποίητος. Ένα άλλο και πιο σύνηθες αίτιο είναι ότι ο χρόνος επικοινωνίας από έναν υπολογιστικό πόρο σε κάποιον άλλον αφήνει κενά ανάμεσα στην τελευταία εργασία που έχει ανατεθεί να τρέξει και σε αυτή που πρόκειται να τρέξει και περιμένει δεδομένα που προέρχονται από άλλον πόρο. Τέλος και με παρόμοια λογική με το προηγούμενο αίτιο, εάν μία εργασία αναγκαστεί να αλλάξει πόρο για να τρέξει πιο γρήγορα και η προηγούμενη της είχε μεγάλο χρόνο εκτέλεσης δημιουργούνται μεγάλα χρονικά κενά στον εκάστοτε πόρο στον οποίο θα κληθεί να τρέξει η εργασία.

Κύριος σκοπός της προτεινόμενης μεθόδου είναι το γέμισμα αυτών των τρυπών. Κάθε πιθανό κενό που γεμίζει πρακτικά μεταφράζεται σε κερδισμένο χρόνο, σε μείωση του συνολικού τελικού χρόνου εκτέλεσης της ροής και κατ' επέκταση και σε οικονομία χρημάτων για τον τελικό χρήστη. Σημαντικό είναι σημειωθεί πως η συγκεκριμένη μέθοδος παίρνει σαν δεδομένο πως η ροή εργασιών είναι γνωστή εκ των προτέρων. Αυτό σημαίνει ο αλγόριθμος γνωρίζει για κάθε εργασία εκ των προτέρων και δεν εμφανίζονται εργασίες μετέπειτα. Αυτό επιτρέπει στον αλγόριθμο να καλύπτει τα χρονικά κενά που δημιουργούνται κατά την διάρκεια της σχεδίασης του χρονοδιαγράμματος.

Σε έναν ευρετικό αλγόριθμο όπως του HEFT μπορεί κανείς να δει τα χρονικά κενά. Τα χρονικά κενά που αφήνει ο αλγόριθμος σε βάθος χρόνου αποδεικνύονται αρκετά προβληματικά. Αυτό γιατί όπως προαναφέρθηκε κενά σαν αυτά στο υπολογιστικό νέφος μπορούν να μεταφραστούν ως παραπάνω χρεώσεις για τον χρήστη.



Εικόνα 4.6. Στιγμιότυπο χρονοδιαγράμματος μίας ροής εργασιών από το άρθρο των Topcuoglu, Haluk; Hariri, Salim; Wu, M. (2002).

Αναφορικά με την εικόνα 4.6 μπορούν να γίνουν οι εξής παρατηρήσεις:
Είναι εμφανές ότι υπάρχουν χρονικά κενά. Κάποια από αυτά για παράδειγμα είναι το χρονικό κενό 0-27 στον υπολογιστικό πόρο P1, το χρονικό κενό 43-56 στον P2 και το χρονικό κενό 40-54 επίσης στον P1. Σημαντικό είναι να αναφερθούν τα εξής: Καταρχάς μπορεί να θεωρεί προφανές το γεγονός ότι οποιοδήποτε χρονικό διάστημα μετά το τέλος και της τελευταίας εργασίας δεν υπολογίζεται ως αδρανής χρόνος για τους υπολογιστικούς πόρους. Ακόμα η τελευταία χρήση ενός υπολογιστικού πόρου μπορεί να θεωρηθεί ως και το τέλος της ενοικίασης του. Τέλος υπάρχουν περιπτώσεις στις οποίες δεν γίνεται να γίνει αποφυγή των χρονικών τρυπών. Η ποιο συνήθης περίπτωση και αυτή που επηρεάζει σε συντριπτικό ποσοστό το εάν μπορούν να γεμίσουν τα χρονικά κενά είναι η αλληλεξάρτηση που υπάρχει ανάμεσα στις εργασίες μίας ίδιας ροής. Αυτό γίνεται γιατί όπως και στο παράδειγμα η αξιοποίηση των πόρων δυστυχώς υπομονεύεται εξαιτίας των αλληλεξαρτήσεων μεταξύ των εργασιών. Για παράδειγμα εάν δεν τελειώσει η εργασία n1 η n4 και n2 δεν μπορούν να τρέξουν και επειδή τα παιδιά αυτών περιμένουν την εκτέλεση τους μένουν χρονικά κενά στο χρονοδιάγραμμα τα οποία δεν είναι εφικτό να καλύπτονται.

Ο λόγος που γίνεται αναφορά για εργασίες που ανήκουν στην ίδια ροή και μόνο είναι γιατί αργότερα όταν γίνει αναφορά σε πολλαπλές ροές αυτό το πρόβλημα παύει να ισχύει.

Ο αλγόριθμος που αναπτύχθηκε λοιπόν αποτελείται από έναν βασικό βρόχο ο οποίος λειτουργεί με κριτήρια ταξινόμησης και ανάθεσης προτεραιοτήτων παρόμοια αυτά του HEFT. Εκεί που διαφοροποιείται του HEFT όμως είναι να τα εξής σημεία:

1. Πριν την ανάθεση μίας εργασίας σε κάποιον υπολογιστικό πόρο γίνεται έλεγχος κάθε υπολογιστικού πόρου για τυχόν χρονικά κενά εντός στο ήδη υπάρχων θεωρητικό χρονοδιάγραμμα που έχει ο πόρος. Σημαντικό επίσης να ειπωθεί πως ο κάθε υπολογιστικός πόρος γνωρίζει μόνο για τις εργασίες που του έχουν ανατεθεί και σε ποιες χρονικές στιγμές έχουν ανατεθεί να τρέξουν κάθε μία από αυτές.

2. Με την εισαγωγή μίας εργασίας σε ένας υπολογιστικό πόρο ο αλγόριθμος λαμβάνει υπόψιν του το γεγονός αυτό και ανατρέχει στο εν λόγω πόρο και βρίσκει τα νέα χρονικά κενά που δημιουργούνται με κάθε εισαγωγή. Αυτό συμπεριλαμβάνει και την πιθανότητα να δημιουργηθούν νέα κενά από την εισαγωγή μίας εργασίας σε ένα ήδη υπάρχων κενό.

3. Ο αλγόριθμος στις περισσότερες περιπτώσεις έχει σαν κύριο γνώμονα να μειώσει το σύνολο των τρυπών. Δηλαδή ανάμεσα σε έναν υπολογιστικό πόρο ο οποίος έχει κενά και σε κάποιον που δεν έχει, η εργασία θα κληθεί να τρέξει σε αυτόν που έχει χρονικά κενά και ας μην είναι η εκτέλεση της σε αυτόν τόσο ιδανική. Αυτό συμβαίνει γιατί ουσιαστικά κάθε γέμισμα ενός χρονικού κενού μπορεί να μεταφραστεί και ως μικρότερος συνολικός χρόνος εκτέλεσης για ολόκληρη την ροή εργασιών. Δημιουργήθηκαν όμως περιπτώσεις όμως, που ανάλογα τα ορίσματα που έχει δώσει ο προγραμματιστής, μπορεί αντί να γίνεται προτεραιότητα ένα κενό να γίνεται επιλογή του πόρου στον οποίο η εργασία που εξετάζεται να έχει γρηγορότερο χρόνο εκτέλεσης ασχέτως σε ποιον πόρο και χωρίς να εξετάζεται εάν τελικά γεμίζει κάποιο χρονικό κενό.

4. Η απόδοση του αλγόριθμου σε σχέση με τον HEFT μετριέται με τέσσερις μεταβλητές. Η μία είναι το συνολικό κόστος εκτέλεσης της ροής εργασιών, το πόσα κενά καλύφθηκαν, πόσος καθαρός χρόνος κερδήθηκε με κάθε εισαγωγή μίας εργασίας σε έναν υπολογιστικό πόρο και τέλος τι ποσοστό αδράνειας έχει κάθε υπολογιστικός πόρος. Σημαντικό είναι να ειπωθεί πως ο χρόνος που κερδήθηκε και ο αριθμός των χρονικών τρυπών που καλύφθηκαν λαμβάνονται υπόψιν βάση των ορισμάτων που δόθηκαν.

Οι τρόποι που γίνεται επιλογή ενός χρονικού κενού βασίζονται σε θεμελιώδης τεχνικές bin packing και είναι οι εξής:

- First-Fit
  Ο αλγόριθμος επιλέγει τον πρώτο διαθέσιμο πιθανό χρονικό κενό που υπάρχει σε έναν υπολογιστικό πόρο και η ανάθεση μίας εργασίας σε αυτόν δεν παραβιάζει κάποιους από τους κανόνες χρονοδρομολόγησης. Η ιδέα και οι περιπτώσεις που λειτουργεί η συγκεκριμένη μέθοδος είναι ότι δεν χρειάζεται να επιλεχθεί το ιδανικό χρονικό κενό κάθε φορά. Αυτό επιτρέπει σε κάποιες περιπτώσεις να υπάρχουν επιπτώσεις στο χρονοδιάγραμμα οι οποίες μπορεί να αποδειχθούν θετικές μακροπρόθεσμα.

- Best-Fit
  Ο αλγόριθμος επιλέγει ανάμεσα σε ένα σύνολο χρονικών κενών το κενό το οποίο εάν χρονοδρομολογηθεί η εργασία θα αφήσει το μικρότερο κενό πίσω στον υπολογιστικό πόρο. Με την συγκεκριμένη μέθοδο μία εργασία καλύπτει κάποιο κενό όσο πιο βέλτιστα γίνεται. Αυτό επιτρέπει σε κάθε εργασία να μπορεί βρει το κατάλληλο χώρο να ανατεθεί με αποτέλεσμα να υπάρχουν συνολικά περισσότερα χρονικά κενά καλυμμένα.
- Worst-Fit
  Ο αλγόριθμος επιλέγει ανάμεσα σε ένα σύνολο χρονικών κενών το κενό το οποίο εάν χρόνο-δρομολογηθεί η εργασία θα αφήσει το μεγαλύτερο κενό πίσω στον υπολογιστικό πόρο. Με μία πρώτη σκέψη μπορεί να μην ακουστεί ιδανική σαν μέθοδος όμως στην προκειμένη περίπτωση επιτρέπει να δημιουργούνται καταστάσεις στις οποίες το συνολικό χρονοδιάγραμμα μειώνεται. Μία από αυτές είναι ότι υπάρχει η πιθανότητα με μία ανάθεση εργασίας να δημιουργηθούν παραπάνω χρονικά κενά σαφώς μικρότερες της πρώτης αλλά επιτρέποντας σε παραπάνω εργασίες να χρονοδρομολογηθούν σε αυτές. Το καλό με την συγκεκριμένη μέθοδο είναι ότι με κάθε χρόνο-δρομολόγηση μίας εργασίας σε ένα χρονικό κενό παραμένει πάντα μεγάλο ποσοστό του ήδη υπάρχων κενού για συμπληρωθεί ίσως αργότερα.
- Quickest-Fit
  Εάν και δεν αποτελεί ακριβώς πολιτική bin packing είναι μία επιλογή που μπορεί να ορίσει ο προγραμματιστής σαν όρισμα και αποδείχθηκε ότι αποφέρει αξιόλογα αποτελέσματα. Στην συγκεκριμένη μέθοδο δεν έχουν προτεραιότητα ανάθεσης τα χρονικά κενά. Αλλά αυτό που συμβαίνει είναι ότι επιλέγεται ο υπολογιστικός πόρος ο οποίος μπορεί να φέρει το πέρας μίας εργασίας το συντομότερο. Αυτό σημαίνει ότι ο αλγόριθμος επιλέγει ανάμεσα σε ένα σύνολο τρυπών και ένα σύνολο υπολογιστικών πόρων.

Ruin and Recreate 4.4

Κάτι στο οποίο έγινε εκτενής μελέτη ήταν η εφαρμογή ενός μεταευρετικού αλγορίθμου με το όνομα Ruin and Recreate. Ο Ruin and Recreate είναι ένας αλγόριθμος που προτάθηκε το 2000 [8] . Πρόκειται για έναν αλγόριθμο βελτιστοποίησης ο οποίος λειτουργεί σε πάνω σε μία ήδη υπάρχουσα λύση. Συγκεκριμένα ο ruin and recreate παίρνει τις λύσεις των προβλημάτων τις καταστρέφει εν μέρει και τις ξαναχτίζει / τις ξαναδημιουργεί στη συνέχεια. Η συγκεκριμένη μέθοδος θεωρείται κατάλληλη για σύνθετα προβλήματα βελτιστοποίησης (“ασυνεχή” προβλήματα με δυσεύρετες αποδεκτές λύσεις, προβλήματα με πολύπλοκους στόχους ή πολλούς περιορισμούς όπως χρόνο-δρομολόγηση μίας ροής εργασιών). Το βασικό στοιχείο της ιδέας αυτής είναι η απόκτηση νέων λύσεων βελτιστοποίησης με γνώμονα την εξασφάλιση της μη καταπάτησης όλων των πολύπλοκων περιορισμών που έχουν τεθεί από το πρόβλημα μας. Ο ruin-and-recreate είναι ένας επαναλαμβανόμενος αλγόριθμος ο οποίος χωρίζεται σε δύο βασικά τμήματα. Το πρώτο τμήμα είναι αυτό του “ruin” στο συγκεκριμένο κομμάτι οφείλουμε να σπάσουμε ένα μέρος της υπάρχουσας λύσης. Το δεύτερο κομμάτι είναι το “recreate” στο οποίο ανακατασκευάζει την λύση με κριτήρια τα οποία θα μας οδηγήσουν σε κάτι καινούργιο. Τέλος ο αλγόριθμος σταματάει με βάση ενός κριτηρίου αποδοχής που θέτουμε όπως για παράδειγμα

“Threshold Annealing” ή τεχνικές “Greedy”. Για να γράψουμε μία ρουτίνα βελτιστοποίησης βάση του ruin-and-recreate, πρέπει να σκεφτούμε το είδος και το μέγεθος των βημάτων καταστροφής της λύσης. Ακόμα θα πρέπει να δούμε για το πως θα αναδημιουργηθούν τα κατεστραμμένα μέρη της λύσης. Τέλος με ποια λογική θα θεωρούμε αποδεκτή μία νόμιμη λύση? Μία ιδέα θα ήταν να δεχόμαστε μόνο καλύτερες λύσεις (“Greedy acceptance”) ή να πρωχορήσουμε βάση των Simulated Annealing, Threshold Accepting, Great Deluge τεχνικών. Με την βοήθεια του ruin-and-recreate μπορούμε να έχουμε μεγαλύτερο εύρος τιμών λόγω των μεγάλων κινήσεων που κάνει ο αλγόριθμος. Τεχνικές που μοιράζονται τα χαρακτηριστικά του ruin-and-recreate έχουν σαν αποτέλεσμα η μία πιθανή λύση με την άλλη να διαφέρουν δραματικά και αυτό οφείλεται πολλές φορές και λόγω της φύσης των προβλημάτων των ίδιων. Επειδή έχουν πολλούς περιορισμούς που πρέπει να λάβουμε υπόψιν ακόμα και μία μικρή παραλλαγή στην λύση δημιουργεί ένα ντόμινο εφέ που επηρεάζει το αποτέλεσμα. Ακόμα η λογική του ruin-and-recreate βασίζεται σε “μεγάλα” βήματα και μεγάλες αλλαγές ως προς τις πιθανές λύσεις που δημιουργεί. Αυτό σημαίνει ότι κατά την καταστροφή μίας πιθανής λύσης καταλήγουμε να καταστρέφουμε ένα πολύ μεγάλο ποσοστό της με σκοπό να πάμε σε μία άλλη με σκοπό να πέφτουμε εύκολα σε τοπικά ελάχιστα/μέγιστα.

Εφαρμογές του Ruin and Recreate από την επιστημονική κοινότητα 4.4.α

Παρακάτω βλέπουμε πως η επιστημονική κοινότητα έχει ενσωματώσει τον συγκεκριμένο αλγόριθμο για την επίλυση ποικίλων προβλημάτων:

A hybrid metaheuristic for the time-dependent vehicle routing problem with hard time windows [29]

Το παραπάνω paper πραγματεύεται το εξής. Πρόκειται για ένα χρονοεξαρτώμενο πρόβλημα δρομολόγησης οχημάτων με αυστηρά deadlines. Ακόμα κάποιες παραπάνω πληροφορίες είναι οι εξής:

- Παραγγελίες γίνονται από N πελάτες
- Κάθε πελάτης έχει ένα deadline παράδοσης.
- Ένα όχημα μπορεί να επισκεφθεί έναν πελάτη μόνο μία φορά.
- Κάθε όχι έχει συγκεκριμένο βάρος που μπορεί να μεταφέρει και πρέπει να γυρνάει στην ίδια βάση.

Στόχος του προβλήματος είναι μείωση των συνολικών οχημάτων και σε δεύτερο χρόνο η μείωση της συνολικής απόστασης από τα οχήματα.
Βάση των παραπάνω οι αρθρογράφοι του paper βελτίωσαν έναν ήδη υπάρχων αλγόριθμο ο οποίος βασίζεται στην κύρια λογική του ruin-and-recreate. Πιο συγκεκριμένα ο αλγόριθμος αφαιρούσε διαδοχικά όλους τους πελάτες που ήταν ανατεθειμένοι σε κάποιο όχημα μειώνοντας τις διαδρομές. Πρακτικά ανέθετε πελάτες σε διαφορετικό όχημα χωρίς να παραβιάζει τους προαναφερθέντες προορισμούς και χωρίς να παραβιάζει τα αυστηρά χρονικά περιθώρια του εκάστοτε πελάτη. Με άλλα λόγια κατέστρεφε μια ήδη υπάρχουσα λύση και βάση της προηγούμενης προσπαθούσε να δημιουργήσει μια καινούργια. Ακόμα επειδή η χρονική πολυπλοκότητα του αλγορίθμου ήταν μεγάλη, επέτρεψαν να παραβιάζονται τα

χρονικά περιθώρια μεμονωμένα στην προσπάθεια να ανακαλύψουν ποια ζευγάρια πελατών είναι τα πιο δύσκολα και προβληματικά στο να συνδυαστούν για να τα καλύψει ένα όχημα. Η ιδέα λοιπόν ήταν να επιτρέπονται όσο γίνεται μικρότερα λανθασμένα χρονικά περιθώρια και να αποκλείονται τα ζευγάρια των πελατών που είναι τα πιο προβληματικά. Αν δηλαδή ο πελάτης Α με τον πελάτη Β είναι προβληματικός ο συνδυασμός τους τότε το απορρίπτουμε σαν ζευγάρι και δεν επιτρέπεται ένα όχημα να περάσει και από τους δύο. Με αυτό έχουμε σαν αποτέλεσμα την μείωση των συνολικών οχημάτων που χρειάζονται για την κάλυψη των όλων των πελατών/διαδρομών.

A Destroy and Repair Algorithm for the Bike sharing Rebalancing Problem [30]

Στο παραπάνω άρθρο οι αρθρογράφοι ασχολούνται με το πρόβλημα της εξισορρόπησης κοινής χρήσης ποδηλάτων. Πρόκειται για ένα πρόβλημα που μοιάζει στη λογική με το προηγούμενο άρθρο λόγω του γεγονότος ότι έχουν πολλά κοινά στοιχεία όπως ότι πάλι έγκειται σε πρόβλημα δρομολόγησης οχημάτων τα οποία έχουν να παραλάβουν και να παραδώσουν εμπόρευμα καθώς και χρονικά περιθώρια για κάθε διαδρομή. Ο αλγόριθμος που προτάθηκε είναι ένας μεταευρετικός αλγόριθμος καταστροφής και επιδιόρθωσης. Πρόκειται δηλαδή για έναν αλγόριθμο ο οποίος χρησιμοποιεί κύρια χαρακτηριστικά του ruin-and-recreate. Πιο αναλυτικά ο αλγόριθμος αποτελείται κυρίως από έναν επαναλαμβανόμενο μηχανισμό καταστροφής και επιδιόρθωσης, εμπλουτισμένο με διαδικασίες τοπικής αναζήτησης. Κατασκευάζει πρώτα μια εφικτή αρχική λύση χρησιμοποιώντας έναν άπληστο αλγόριθμο και τον κάνει πιο συγκεκριμένο χρησιμοποιώντας ένα σύνολο τοπικών αναζητήσεων. Αυτό το διάλυμα στη συνέχεια καταστρέφεται επαναληπτικά και επισκευάζεται με σειρά, και βελτιώνεται ξανά από τις διαδικασίες τοπικής αναζήτησης, έως ότου να επιτευχθεί ένα κριτήριο διακοπής. Σημειώστε ότι ο αλγόριθμός λειτουργεί με διαδρομές που είναι εφικτές σε σχέση με τον περιορισμό χωρητικότητας, αλλά δέχεται λύσεις στις οποίες είναι ο αριθμός των διαδρομών μεγαλύτερο από m. Αυτός ο αριθμός ελαχιστοποιείται σιωπηρά από αρκετές από τις εγκεκριμένες διαδικασίες τοπικής αναζήτησης.

An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows [31]

Όπως και τα προαναφερθέντα επιστημονικά άρθρα έτσι και το παραπάνω πραγματεύεται το πρόβλημα παραλαβής και παράδοσης με τα χρονικά παράθυρα να είναι το πρόβλημα της εξυπηρέτησης ενός αριθμού αιτημάτων μεταφοράς χρησιμοποιώντας περιορισμένο αριθμό οχημάτων. Κάθε αίτημα περιλαμβάνει τη μεταφορά ενός αριθμού εμπορευμάτων από μια τοποθεσία παραλαβής σε τοποθεσία παράδοσης. Ο προτεινόμενος αλγόριθμος λοιπόν πρέπει να κατασκευάζει διαδρομές που τα οχήματα επισκέπτονται όλες τις τοποθεσίες, έτσι ώστε οι αντίστοιχες παραλαβές και οι παραδόσεις γίνονται στην ίδια διαδρομή, και έτσι ώστε να γίνεται παραλαβή πριν από την αντίστοιχη διανομή. Οι διαδρομές πρέπει επίσης να ικανοποιούν περιορισμούς χρόνου και χωρητικότητας. Ο αλγόριθμος που προτάθηκε πρόκειται για μία επέκταση του LNS (Large Neighborhood Search) ο οποίος μοιράζεται την λογική του ruin-and-recreate. Με την βοήθεια του LNS (και την επέκταση του) μπορούν να έχουν μεγαλύτερο εύρος τιμών συγκριτικά με την μέχρι τότε βιβλιογραφία που η μία πιθανή λύση με την άλλη δεν

είχαν μεγάλες διαφορές. Σε αντίθεση με την LNS που μοιάζει σε λογική με τον ruin-and-recreate πιθανή λύση που προκύπτει διαφέρει σε σχέση με τις υπόλοιπες. Έπειτα οι αρθρογράφοι εξετάζουν διάφορες removal (ruin κομμάτι) και insertion (recreate κομμάτι) μεθόδους, καθώς και κριτήρια διακοπής αφού πρόκειται για επαναλαμβανόμενο αλγόριθμο.

A general heuristic for vehicle routing problems [32]

Εδώ οι αρθρογράφοι παρουσίασαν ένα ενοποιημένο ευρετικό που είναι σε θέση να λύσει πέντε διαφορετικές παραλλαγές του προβλήματος δρομολόγησης οχημάτων. Το πρόβλημα με τα παράθυρα χρόνου (VRPTW), το πρόβλημα δρομολόγησης χωρητικότητας οχημάτων (CVRP), το πρόβλημα δρομολόγησης οχημάτων πολλαπλών αποθηκών (MDVRP), το πρόβλημα δρομολόγησης οχήματος που εξαρτάται από την τοποθεσία (SDVRP) και το πρόβλημα δρομολόγησης ανοιχτού οχήματος (OVRP). Όλες οι παραλλαγές προβλημάτων επιλύονται χρησιμοποιώντας την προσαρμοστική αναζήτηση μεγάλης γειτονιάς πλαίσιο (ALNS). Το ALNS είναι μια επέκταση της μεγάλης γειτονιάς πλαίσιο αναζήτησης με ένα προσαρμοστικό στρώμα. Αυτό το επίπεδο επιλέγει προσαρμοστικά μεταξύ ενός αριθμού ευρετικών εισαγωγής και αφαίρεσης που εντείνουν και διαφοροποιούν την αναζήτηση. Η προσέγγιση που παρουσιάστηκε έχει μια σειρά από πλεονεκτήματα: παρέχει λύσεις πολύ υψηλής ποιότητας, ο αλγόριθμος είναι ισχυρός και σε κάποιο βαθμό αυτό-προσαρμόζεται. Ο παραπάνω αλγόριθμος μπορεί να εφαρμοστεί σε μεγάλο αριθμό από προβλήματα βελτιστοποίησης που έχουν πολλούς περιορισμούς.

An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives [33]

Στο συγκεκριμένο άρθρο προτείνονται δύο νέοι αλγόριθμοι IG (Iterated Greedy σημειώστε ότι περιστασιακά οι IG αναφέρονται επίσης ως και Ruin-and-Recreate). Οι επαναλαμβανόμενοι αλγόριθμοι Greedy ξεκινούν από κάποια αρχική λύση και στη συνέχεια επαναλαμβάνονται μέσω ενός κύριου βρόχου στον οποίο λαμβάνεται πρώτα μια μερική υποψήφια λύση sp αφαιρώντας έναν αριθμό συστατικών της λύσης από μια πλήρη υποψήφια λύση s και στη συνέχεια από μία άλλη πλήρης λύση s' ανακατασκευάζεται ξεκινώντας με sp. Πριν συνεχίσει με τον επόμενο βρόχο, ένα κριτήριο αποδοχής αποφασίζει εάν το s' θα γίνει η νέα υπάρχουσα λύση. Αυτή η διαδικασία επαναλαμβάνεται μέχρι να ικανοποιηθεί κάποιο κριτήριο διακοπής, όπως ο μέγιστος αριθμός επαναλήψεων ή ένα χρονικό όριο υπολογισμού. Είναι επίσης εύκολο να γίνει πρόσθεση μιας προαιρετική φάσης τοπικής αναζήτησης για τη βελτίωση της ανακατασκευασμένης λύσης s′ πριν από την δοκιμή αποδοχής. Στο άρθρο επίσης εξετάζεται η απόδοση των IG ενάντια στους LS (Local Search) αλγόριθμους. Μία βασική διαφορά των δύο κατηγοριών είναι ότι ενώ η κεντρική ιδέα του IG είναι η επανάληψη σε εποικοδομητικούς αλγόριθμους, η κεντρική ιδέα του LS έγκειται στην επανάληψη σε τοπικές αναζητήσεις με συγκεκριμένο τρόπο. Ωστόσο, ειδικά όταν προσθέτουμε μια φάση τοπικής αναζήτησης σε αλγόριθμους IG, οι ομοιότητες μεταξύ IG και LS γίνονται ακόμη πιο έντονες.

Εφαρμογές του Ruin and Recreate με μονή ροή εργασιών 4.4.β

Για σύνθετα προβλήματα όπως η χρονοδρομολόγηση ροής εργασιών με χρήση της λογικής του ruin-and-recreate προτείνονται μεγάλες κινήσεις αντί για μικρότερες με την χρήση γνωστών μεθόδων όπως αυτών που προαναφέραμε. Ο λόγος είναι γιατί σε απλά δομημένα προβλήματα η βέλτιστη λύση μπορεί να βρεθεί χωρίς την ανάγκη για χρήση μεγάλων κινήσεων, επειδή οι αλγόριθμοι συνήθως παρέχουν σχεδόν βέλτιστες λύσεις με πολύ μικρές κινήσεις ήδη. Αντιμετωπίζοντας σύνθετα προβλήματα όμως, συναντάμε σοβαρές δυσκολίες στη χρήση αυτών των κλασικών αλγορίθμων. Αν αλλάξουμε μία δρομολόγηση μίας εργασίας από ένα υπολογιστικό πόρο σε κάποιον άλλον η ποιότητα και η διαφορά των δύο πιθανών λύσεων μπορεί να αλλάξει δραματικά. Ακόμα όσο αναφορά την χρονοδρομολόγηση ροής εργασιών υπάρχουν πολλοί περιορισμοί, όπως το ότι όλοι οι γονείς μίας εργασίας πρέπει να έχουν ολοκληρώσει πριν την έναρξη της εργασίας συμπεριλαμβανομένου και του χρόνου μετάδοσης πληροφοριών από τον γονέα στο παιδί. Με αποτέλεσμα συχνά να είναι δύσκολο να βρούμε μέχρι και απλές αποδεκτές λύσεις. Η προσέγγιση λοιπόν του ruin-and-recreate, και ίσως και το σημαντικό του πλεονέκτημα σε σχέση με άλλες μεθόδους, είναι ότι εάν έχουμε αποσυνθέσει ένα μεγάλο μέρος της προηγούμενης λύσης, έχουμε και μεγάλη ελευθερία να δημιουργήσουμε μία νέα. Μπορούμε εύλογα να ελπίζουμε ότι, σε αυτόν τον μεγάλο χώρο λύσεων, είναι δυνατόν να βρούμε ξανά μία νέα αποδεκτή λύση.
Ο κύριος στόχος του δικού μας προβλήματος είναι:
Ελαχιστοποίηση του συνολικού χρονοδιαγράμματος όλων το ροών εργασιών μαζί.
Ενώ σαν περιορισμούς έχουμε:

- Κάθε εργασίας γονέας πρέπει δρομολογηθεί πριν το παιδί του.
- Αλλαγή από έναν υπολογιστικό πόρο σε έναν άλλο προϋποθέτει ότι η εργασία παιδί θα περιμένει το χρονικό διάστημα να σταλθούν τα δεδομένα.

Έχοντας και σαν έμπνευση τις εφαρμογές του αλγορίθμου από την επιστημονική κοινότητα παράχθηκαν κάποιες παραλλαγές του αλγορίθμου οι οποίες είχαν πολύ καλή εφαρμογή στο πρόβλημα του προγραμματισμού της μονής ροής εργασιών. Οι μέθοδοι ruin-and-recreate που εφαρμόστηκαν:

Random Ruin

Στον random-ruin γίνεται unschedule μία εργασία από τον υπολογιστικό πόρο που έχει επιλεχθεί τα τρέξει. Το ποσοστό που επιλέχθηκε και έδειξε τα καλύτερα αποτελέσματα ήταν το 20% να γίνει unschedule μία εργασία. Αν και μικρό σαν ποσοστό οι επιπτώσεις και οι αλλαγές που γίνονται στην τελική λύση είναι τεράστιες. Αυτό συμβαίνει γιατί και η παραμικρή αλλαγή σε μία εργασία επηρεάζει άμεσα και έμμεσα όλο το χρονοδιάγραμμα. Όχι μόνο από την άποψη ότι ο συνολικός χρόνος της ροής θα είναι διαφορετικός αλλά και από το γεγονός ότι πάρα πολλές εργασίες ενδέχεται να αλλάξουν υπολογιστικούς πόρους στους οποίους θα ανατεθούν. Αυτό συμβαίνει γιατί αν για παράδειγμα η εργασία Β έχει σαν γονέα την εργασία Α και η Α ξεκινήσει εν τέλει την χρονική στιγμή 10 αντί για την χρονική στιγμή 5 τότε μπορεί να “συμφέρει” την Β για να τελειώσει όσο γίνεται πιο γρήγορα (Earliest Finish Time) να τρέξει σε διαφορετικό υπολογιστικό πόρο σε σχέση με αυτόν που ήταν προγραμματισμένη να τρέξει εξαρχής.

Random Recreate

Όσο αναφορά το recreate κομμάτι του αλγορίθμου υπάρχει ένας βρόχος ο οποίος περνάει διαδοχικά από κάθε εργασία και εξετάζει εάν βρίσκεται στις εργασίες στις οποίος έχουν γίνει ruin. Εάν ισχύει αυτή η υπόθεση τότε οι συγκεκριμένες εργασίες ξανά χρονοδρομολογούνται εκ νέου χωρίς την επιλογή να μπορούν να χρονοδρομολογηθούν στον υπολογιστικό πόρο που είχαν επιλεχθεί προηγουμένως. Τέλος ο τρόπος που χρονοδρομολογούνται είναι με την λογική να βρούνε τον υπολογιστικό πόρο που θα τους επιτρέψει να τελειώσουν όσο το δυνατόν πιο σύντομα (Earliest Finish Time).

Level ruin

Μία ιδέα για μερική καταστροφή της λύσης μας είναι να κάνουμε unschedule εργασίες βάση του level τους. Αυτό σημαίνει ότι έχοντας το level order μίας ροής εργασιών μπορούμε να ορίσουμε συγκεκριμένα επίπεδα τα οποία θέλουμε να κάνουμε unschedule από το σύστημα. Για να μπορέσουμε να το εφαρμόσουμε αυτό μπορούμε να βρούμε τον μέγιστο αριθμό των levels μίας ροής και έπειτα να καταστρέφουμε κάθε φορά το 10% των πρώτων levels μέχρι να φτάσουμε και στο τελευταίο. Αυτό μας επιτρέπει όταν καταστρέψουμε με βάση τα επίπεδα να έχουμε ένα τεράστιο domino effect το οποίο θα κάνει cascade από διαφορικό επίπεδο κάθε φορά το οποίο θα φτάνει μέχρι τα exit nodes των ροών. Αυτό θα γίνει γιατί όπως προαναφέραμε στους περιορισμούς μία εργασία παιδί δεν μπορεί να έχει μία ιδέα για το πότε μπορεί να ξεκινήσει εάν δεν έχουν όλοι οι γονείς λάβει ένα χρονικό περιθώριο για το πότε θα εκτελεστούν και που. Επομένως αυτό μας αφήνει και με πολύ μεγάλο χώρο και ελευθερία να εξετάσουμε πιθανές λύσεις.

Level recreate

Στο κομμάτι του level-recreate προσπαθούμε να κάνουμε schedule πρώτα τα levels που βρίσκονται πιο κάτω. Πιο συγκεκριμένα βάζουμε σε ένα σύνολο A όλες τις εργασίες και επιλέγουμε να κάνουμε schedule πρώτα αυτές που έχουν το μεγαλύτερο level και μεγαλύτερο upward-rank. Μόνο φυσικά αν επιτρέπεται βάση των περιορισμών. Αυτό έχει σαν αποτέλεσμα να επιλέγουμε εργασίες οι οποίες λόγω των heuristic αλγορίθμων δεν θα είχαν επιλεχθεί για schedule. Επομένως διευρύνουμε παραπάνω το εύρος των πιθανών μας λύσεων κρατώντας και μία εύλογη ποιότητα καθώς κοιτάμε να λάβουμε υπόψιν και το upward-rank της εργασίας.

Time ruin

Μία άλλη παραλλαγή καταστροφής που εφαρμόσαμε ήταν βάση χρονικών περιθωρίων. Κάτι παρόμοιο είχε προταθεί και στο αρχικό άρθρο του ruin-and-recreate. Σε αυτή την μέθοδο βρίσκεται τον χρόνο που θα χρειαστεί η υπάρχουσα λύση για να τρέξει και μέσα σε αυτό ορίζεται ένα χρονικό παράθυρο στο οποίο καταστρέφεται όσες εργασίες έχουν χρονοδρομολογηθεί να τρέξουν μέσα σε αυτό. Εδώ μπορούν να εφαρμοστούν αρκετές παραλλαγές. Μία ιδέα που μπορεί να υλοποιηθεί είναι να κάθε φορά να γίνεται επιλογή ενός τυχαίου χρονικού παραθύρου. Μία άλλη είναι να γίνει ορισμός ενός ποσοστού που θα καταλαμβάνει το χρονικό παράθυρο συγκριτικά με τον συνολικό χρόνο που χρειάζεται η πιθανή λύση για να τρέξει. Τέλος κάτι και αυτό που εντέλει χρησιμοποιήθηκε είναι ένας συνδυασμός των

δύο δηλαδή να γίνεται τυχαία επιλογή για την αρχή του χρονικού παραθύρου και σαν τέλος να είναι η αρχή συν ένα σταθερό ποσοστό από το συνολικό χρόνο που χρειάζεται η αρχική λύση για να τρέξει. Σημαντικό επίσης είναι να ειπωθεί το γεγονός ότι στην συγκεκριμένη ruin τεχνική όπως και με την random ruin δεν γίνονται unschedule οι εργασίες καθαυτό αλλά απορρίπτουμε τον υπολογιστικό πόρο στον οποίο έχουν ανατεθεί να τρέξουν. Αυτό επιτρέπει τον συνολικό χρόνο που θα χρειαστεί να τρέξει ο αλγόριθμος να είναι συντριπτικά μικρότερος και χωρίς να θυσιάζει πολύ καλύτερες πιθανές λύσεις. Ακόμα δεν έχουμε αποτύχει τον στόχο και την λογική του ruin-and-recreate. Δηλαδή του να καταστρέφουμε μερικά ένα μεγάλο κομμάτι της αρχικής λύσης με σκοπό να ξεφύγουμε από τα τοπικά ελάχιστα/μέγιστα. Αυτό γίνεται κυρίως λόγω του domino εφέ που προαναφέρθηκε. Δηλαδή με την παραμικρή αλλαγή σε κάποια εργασία γονέα ακόμα και η αλλαγή του από έναν υπολογιστικό πόρο σε κάποιον άλλον είναι πολύ πιθανό να επηρεάζει άμεσα τις εργασίες παιδιά όσο αναφορά σε ποιο υπολογιστικό πόρο θα τρέξουν και πότε θα τρέξουν.

Time recreate

Στο κομμάτι του recreate η μέθοδος που χρησιμοποιήθηκε είναι παραπλήσια του random-recreate με την διαφορά ότι γίνεται προσπάθεια χρήσης του βέλτιστου Time Type (EFT, EST) το οποίο διαφοροποιείται βάση του χρονικού παραθύρου στο οποίο βρίσκεται ο αλγόριθμος. Όταν δηλαδή περάσει το 50% του συνολικού χρόνου σε σχέση με την προηγούμενη λύση γίνεται προτίμηση χρήσης Earliest Finish Time αλλιώς Earliest Start Time. Αυτό δίνει ένα μεγαλύτερο εύρος πιθανών τιμών που μπορεί να προκύψουν. Εάν δεν προκύψει κάποιο καλύτερο αποτέλεσμα ο αλγόριθμος αρχίζει και χρησιμοποιεί σταδιακά όλο και λιγότερα ποσοστά για το Earliest Start Time μέχρι έως ότου οι εργασίες γίνονται όλες schedule με Earliest Finish Time.

Order ruin

Ίσως από τις καλύτερες παραλλαγές που εφαρμόστηκαν ήταν η order. Στην συγκεκριμένη παραλλαγή εξετάζεται η ιδέα του να δημιουργηθεί ένα μη έγκυρο χρονοδιάγραμμα το οποίο θα τρέχει παιδιά πριν τρέξουν οι γονείς. Αυτό απευθείας επιτρέπει το ενδεχόμενο να δημιουργηθεί μία καλύτερη τελική λύση χωρίς όμως να είναι έγκυρο. Η εγκυρότητα και η ορθότητα του χρονοδιαγράμματος διασφαλίζεται αργότερα στο κομμάτι της ανακατασκευής της λύσης.

Order recreate

Η παρούσα μέθοδος αποτελείται από έναν βασικό βρόχο ο οποίος διαδοχικά εξετάζει κάθε ροή εργασίας. Σε μία επανάληψη η εργασία χρόνο-δρομολογείται χωρίς να εξετάζεται εάν επιτρέπεται ή εάν παραβιάζει κάποια κριτήρια περιορισμών. Έπειτα για να γίνει όντως η διασφάλιση και να μην παραβιαστούν οι περιορισμοί ενημερώνει όλες τις εργασίες παιδιά που επηρεάζει και μόνο. Ο τρόπος που επιτυγχάνεται αυτό είναι με αναδρομική μέθοδο στην οποία κάθε πατέρας εργασία ενημερώνει και “σπρώχνει” τις εργασίες παιδιά πιο κάτω στο χρονοδιάγραμμα μέχρι να θεωρείται η θέση τους νόμιμη. Σημαντικό να σημειωθεί ότι δεν γίνεται χρόνο-δρομολόγηση των παιδιών ξανά απλά δέχονται μία μετατόπιση αναφορικά με τον

αρχικό τους χρόνο που θα εκτελεστούν στον ήδη επιλεγμένο υπολογιστικό πόρο που βρίσκονται. Τέλος τα παιδιά επηρεάζουν με την σειρά τους τα δικά τους παιδιά έως ότου φτάσουν σε εργασία εξόδου. Ο λόγος που υπάρχει περίπτωση να μην επηρεαστούν όλα τα παιδιά ή και κανένα είναι γιατί μπορεί η εκάστοτε εργασία να εξαρτιέται από άλλον γονέα. Αν για παράδειγμα η εργασία Γ έχει γονείς την Α και την Β και η Β πρακτικά επιτρέπει στην Γ να ξεκινήσει από την χρονική στιγμή 20 και μετά εάν η Α επιτρέπει στην Γ να ξεκινήσει από το 10 και μετά η Γ εξαρτιέται απόλυτα από την Β. Ο μόνος τρόπος που η Α θα μπορούσε να επηρεάσει το πότε θα ξεκινήσει ο Γ θα ήταν εάν είχε χρόνο-δρομολογηθεί έτσι ώστε επέτρεπε στην Γ να ξεκινήσει μετά την χρονική στιγμή 20.

Children ruin

Μία παρεμφερής σε λογική της order έγινε με την children-ruin. Το τελικό αποτέλεσμα φυσικά είναι ότι πάλι θα γίνει ανακατανομή στην σειρά στην οποία θα τρέξουν οι εργασίες, απλά όπως και με την order σαν κατάληξη βγαίνει ένα προσωρινά μη έγκυρο χρονοδιάγραμμα. Η κύρια διαφορά είναι ότι η επιλογή των εργασιών για τις οποίες θα μετατοπιστούν είτε σε άλλον υπολογιστικό πόρο είτε χρονικά μέσα σε αυτόν θα γίνει με το κριτήριο του πόσα παιδιά έχουν. Ο αλγόριθμος βρίσκει την εργασία με τα περισσότερα παιδιά και ξεκινάει ένας βρόχος ο οποίος σε κάθε επανάληψη επιλέγει τις εργασίες που έχουν παιδιά ίσων με N και θέτει το N = N - 1. Ο βρόχος σταματάει όταν το N φτάνει στο 0.

Children recreate

Επειδή δεν υπάρχει κάποιος περιορισμός πριν αναδόμηση της πραγματικής λύσης αυτόματα δίνει χώρο και ελευθερία από άποψη κινήσεων αλλά επιτρέπει και μεγάλο εύρος πιθανών λύσεων. Το γεγονός ότι γίνεται προσπάθεια βελτίωσης και νομιμοποίησης ενός μη έγκυρου χρονοδιαγράμματος δεν χαλάει ιδιαίτερα την ποιότητα της τελικής λύσης καθώς ακόμα ο αλγόριθμος όπως και στην περίπτωση του order-recreate εκμεταλλεύεται πλήρως την ύπαρξη των τρυπών. Με την διαφορά ότι δεν έχει σαν γνώμονα να δημιουργήσει όσο γίνεται περισσότερες. Εκεί που κερδίζει σχετικά με τον order είναι ότι μπορεί να παράγει ικανοποιητικές λύσεις σε πολύ πιο γρήγορο χρονικό διάστημα. Αυτό οφείλεται στο γεγονός ότι έχει συγκεκριμένο αριθμό επαναλήψεων ο οποίος εξαρτιέται πλήρως από την πυκνότητα της ροής εργασιών. Ακόμα σημαντικό είναι να αναφερθεί ότι πυκνές ροές εργασιών δηλαδή ροές οι οποίες έχουν πολλές ακμές είναι λογικό ότι θα κάνουν μερικές καταστροφές οι οποίες θα είναι μεγαλύτερες συγκριτικά με λιγότερα πυκνές ροές. Επειδή η λογική του ruin-and-recreate και επειδή η γενικότερη φύση του προβλήματος του προγραμματισμού μίας ροής εργασιών επωφελείται από μεγάλα βήματα η μέθοδος children έχει καλύτερη εφαρμογή σε πολύ πυκνές ροές εργασιών.

Στατιστικά ευρήματα σχετικά με τους αλγορίθμους 4.5

Στην παρόν υποκεφάλαιο θα παρουσιαστούν κάποια γραφήματα και σχεδιαγράμματα σχετικά με τους αλγορίθμους μονής ροής χρονοδρομολόγησης και την αποδοτικότητα τους κάτω από περιβάλλοντα με διαφορετική υπολογιστική ισχύ ενώ εξετάζεται το ερώτημα αν υπάρχει κάποιο είδος ροής εργασιών το οποίο ευνοείται από κάποιους αλγορίθμους.

Οι αλγόριθμοι εξετάστηκαν στην πρώτη κατηγορία δεδομένων των Custom generated. Αυτό έγινε σκόπιμα έχοντας σαν στόχο να βρεθεί κάποια συσχέτιση στην μορφολογία ενός γράφου και του πιθανού τρόπου χρονοδρομολόγησης του. Από τα ευρήματα αυτά θα συγκριθούν οι κυρίαρχοι αλγόριθμοι συγκριτικά με την απόδοση τους ενάντια στον αλγόριθμου του HEFT. Η συγκρίσεις θα έχουν να κάνουν όσο αφορά το συνολικό χρόνο εκτέλεσης μίας ροής, τον χρόνο που οι υπολογιστικοί πόροι έμειναν ανεκμετάλλευτοι αλλά ακόμα και αν υπάρχουν περιπτώσεις που γίνεται να αποδεσμευτεί ολοκληρωτικά κάποιος υπολογιστικός πόρος για να επαναχρησιμοποιηθεί κάπου αλλού.

Έπειτα θα γίνει επισκόπηση των ορισμάτων που επηρεάζουν περισσότερο την αποδοτικότητα του αλγορίθμου του HEFT και την αποδοτικότητα αυτών που προτάθηκαν σε σχέση με αυτών και τους λόγους που κατάφεραν να αποδώσουν περισσότερο συγκριτικά με τον πρώτο.

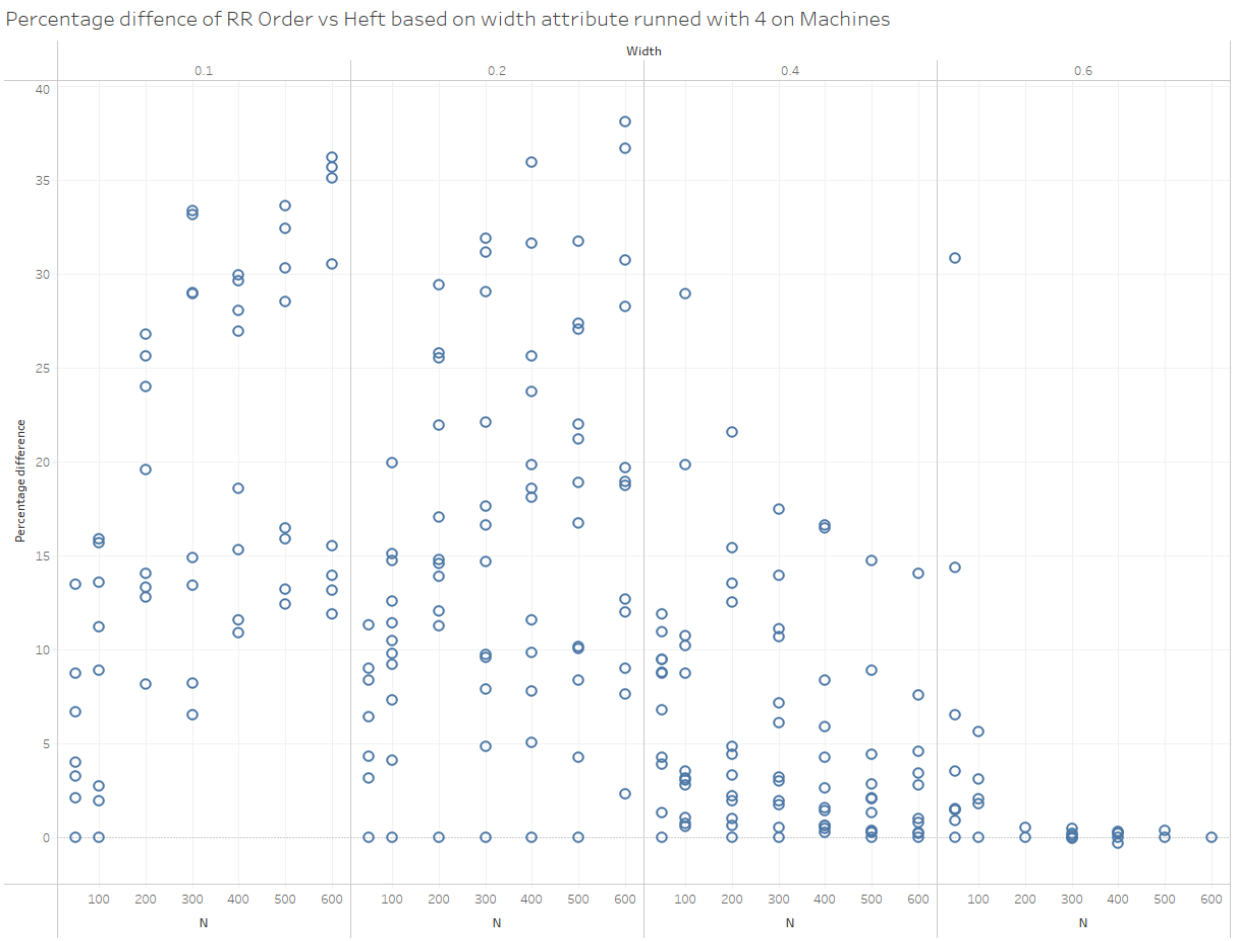
Τέλος θα παρουσιαστούν γραφικές παραστάσεις στις οποίες φαίνεται η ποσοστιαία βελτίωση που είχε ένα χρονοδιάγραμμα όταν εφαρμόστηκε ο αλγόριθμος Ruin and Recreate στην υπάρχουσα λύση.

Η επιρροή του πλάτος των ροών στην αποδοτικότητα των αλγορίθμων 4.5.α

Όπως προαναφέρθηκε έγινε χρήση των Custom generated δεδομένων. Ενώ στο υποκεφάλαιο 3.1 εξηγήθηκε ότι τα δεδομένα αυτά παράχθηκαν βάση συγκεκριμένων ορισμάτων. Ένα από τα ορίσματα αυτά έχει να κάνει με την παράμετρο fat. Ουσιαστικά αυτή η παράμετρος ελέγχει το πιθανό πλάτος που μπορεί να έχει ο γράφος που θα παραχθεί. Χρησιμοποιώντας έναν μικρό αριθμό για την συγκεκριμένη μεταβλητή θα είχε ως αποτέλεσμα έναν στενό σε πλάτος γράφο. Δηλαδή ένας γράφος ο οποίος δεν θα έχει πολλούς κόμβους ανά επίπεδο. Ενώ ένας γράφος που έχει λάβει έναν μεγάλο αριθμό για την μεταβλητή fat θεωρείτε πλατύς με πολλούς κόμβους ανά επίπεδο.

Ανάλογα με την μεταβολή της τιμής του ορίσματος παρατηρείται η αλλαγή της αποδοτικότητας των αλγορίθμων αλλά και της συμπεριφοράς τους. Το οποίο όπως θα συζητηθεί και αργότερα μπορεί να επηρεάζει ένα περιβάλλον το οποίο βρίσκεται στο υπολογιστικό νέφος. Ακόμα θα σχολιαστούν οι περιπτώσεις στις οποίες διαφοροποιείται η υπολογιστική ισχύ του συστήματος και το πως επηρεάζει αυτό τις τελικές λύσεις των αλγορίθμων αλλά και την αποδοτικότητα τους με συνδυασμό με το πόσο πλατύς είναι ένας γράφος.

Παρακάτω μπορεί κανείς να παρατηρήσει ότι υπάρχουν τέσσερα υπό-γραφήματα τα οποία έχουν ομαδοποιηθεί βάση της μεταβλητής Width (ή fat) των ροών εργασιών.



Εικόνα 4.7. Στιγμιότυπο από διάγραμμα που δημιουργήθηκε με το εργαλείο οπτικοποίησης Tableau [34].

Στο παραπάνω διάγραμμα αρχικά οι ροές ομαδοποιήθηκαν βάση του πλάτους τους. Ο άξονας x δείχνει τον αριθμό των κόμβων των γράφων. Ενώ ο άξονας y δείχνει το ποσοστό βελτίωσης του διαγράμματος όταν έγινε χρήση του αλγορίθμου Ruin Order πάνω σε μία πιθανή λύση που είχε παραχθεί με τον αλγόριθμο του HEFT. Κάθε κύκλος πάνω στα υπό-διαγράμματα πρόκειται για μία ροή εργασιών που έτρεξε και με τους δύο αλγόριθμους, ενώ η θέση του κύκλου πάνω στο διάγραμμα δείχνει πόσους κόμβους είχε (N) και το ύψος στο οποίο βρίσκεται δείχνει το πόσο μεγάλη ποσοστιαία βελτίωση είχε όταν για την συγκεκριμένη ροή εργασιών έτρεξε και ο Ruin Order.

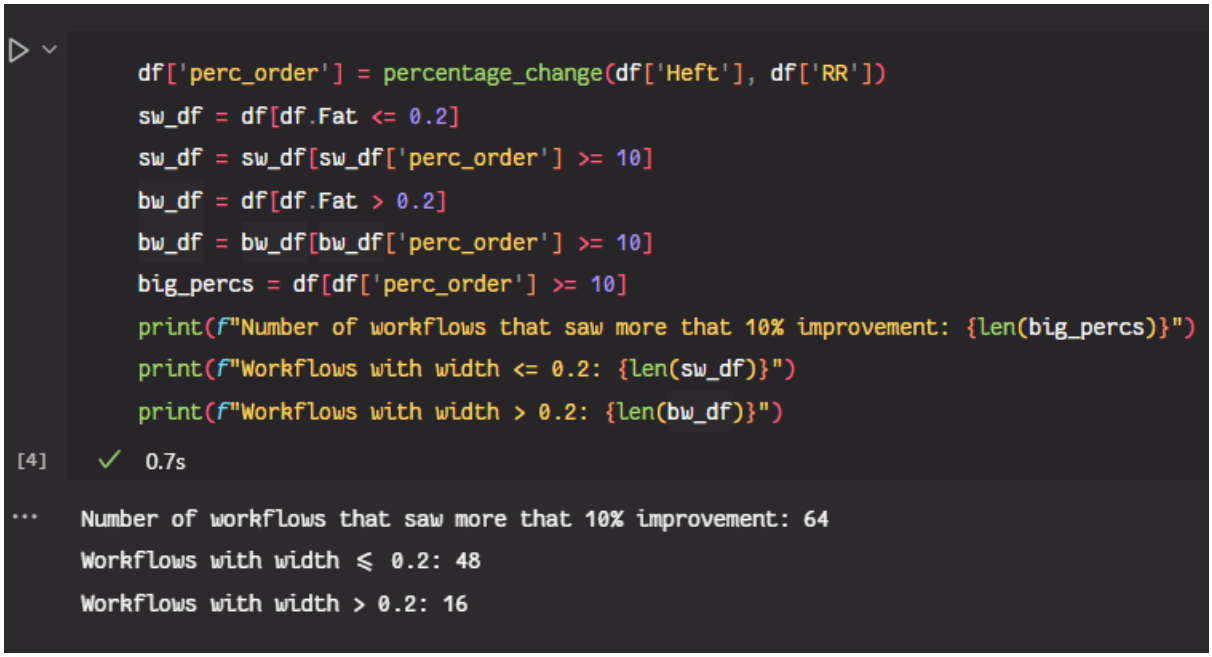
Ακόμα βάση του παραπάνω διαγράμματος μπορεί κάποιος να κάνει την παρατήρηση ότι οι στενότεροι γράφοι είχαν πολύ καλύτερο ποσοστό βελτίωσης του χρόνο-διαγράμματος τους όταν εφαρμόστηκε πάνω τους ο αλγόριθμος Ruin Order. Μάλιστα όταν οι γράφοι είχαν μεγάλο πλάτος η βελτιώσεις που μπορούσε να κάνει ο αλγόριθμος ήταν εξαιρετικά μικρές συγκριτικά τους γράφους με λιγότερο πλάτος. Ενώ εκ πρώτης όψης το γεγονός αυτό μπορεί να φανεί οξύμωρο. Η πιθανή εξήγηση έχει λογική συνοχή.

Σε μία πλατιά ροή εργασιών υπάρχει το θετικό στοιχείο ότι επιτρέπει πολύ μεγαλύτερο παραλληλισμό των εργασιών. Αυτό γιατί οι εργασίες ανά επίπεδο δεν έχουν αλληλεξαρτήσεις μεταξύ τους. Σε αυτές τις περιπτώσεις όπως διαπιστώνεται

και από την εικόνα 4.7 η εφαρμογή του αλγορίθμου του HEFT βρίσκει μία φοβερά αποδοτική λύση η οποία δύσκολα βελτιώνεται. Ο λόγος που αδυνατεί να βρει τόσο αποδοτική λύση όταν υπάρχουν λιγότερες εργασίες ανά επίπεδο έχει να κάνει με το γεγονός ότι πρόκειται για εκ φύσεως άπληστο αλγόριθμο. Όταν η πολιτική επιλογής ενός υπολογιστικού πόρου έχει να κάνει με την γρηγορότερη εκτέλεση της συγκεκριμένης εργασίας και μόνο για εκείνη την χρονική στιγμή, δεν μπορεί να υπολογίσει τι επιρροή θα έχει αυτή η επιλογή σε βάθος επιπέδων. Παρόλο που οι εργασίες έχουν προτεραιότητα βάση της μεταβλητής upward-rank δεν αρκεί. Αυτό γιατί αν γίνει η υπόθεση ότι έστω μία εργασία μπορεί να έχει επιλεχθεί βέλτιστα βάση του ορίσματος της προτεραιότητας της ο υπολογιστικός πόρος στον οποίο θα τεθεί να χρόνο-δρομολογηθεί μπορεί προσωρινά να προσδίδει το καλύτερο χρόνο αλλά μακροπρόθεσμα οι υπόλοιπες εργασίες ενδείκνυται να χρειαστεί να αλλάξουν υπολογιστικό πόρο για να μπορέσουν να έχουν την δικιά τους γρηγορότερη εκτέλεση βάση του ορίσματος EFT του HEFT. Αυτό έχει ως αποτέλεσμα να επιλέγεται ο καλύτερος πόρος για κάθε εργασία ξεχωριστά εκείνη την χρονική στιγμή αλλά με την πιθανότητα να θυσιάζεται μία καλύτερη πιθανή λύση που μπορεί προκύπτει από ανάθεση σε έναν άλλο υπολογιστικό πόρο.

Ακόμα ο αλγόριθμος φτάνει σε σημεία που δημιουργεί χρονικά κενά στο χρόνο-διάγραμμα τις οποίες δεν γίνεται να της αποφύγει λόγω του κόστους εκτέλεσης μίας εργασίας σε έναν πόρο ή ακόμα και λόγω του χρόνου επικοινωνίας που χρειάζεται από τον έναν υπολογιστικό πόρο στον άλλον. Υπάρχουν δηλαδή περιπτώσεις που ο αλγόριθμους του HEFT θα επιλέξει μία αρκετά αποδοτική σειρά για την εκτέλεση των εργασιών αλλά μπορεί μία εργασία να κληθεί να αλλάξει υπολογιστικό πόρο έχοντας ως συνέχεια να αφήνει χρονικά κενά οι οποίες δεν εκμεταλλεύονται από τις επόμενες εργασίες που είναι η σειρά τους να προγραμματιστούν. Αυτό βέβαια δεν ισχύει με τον αλγόριθμο του Ruin Order καθώς δεν περιορίζεται από τέτοιου είδους προβλήματα.

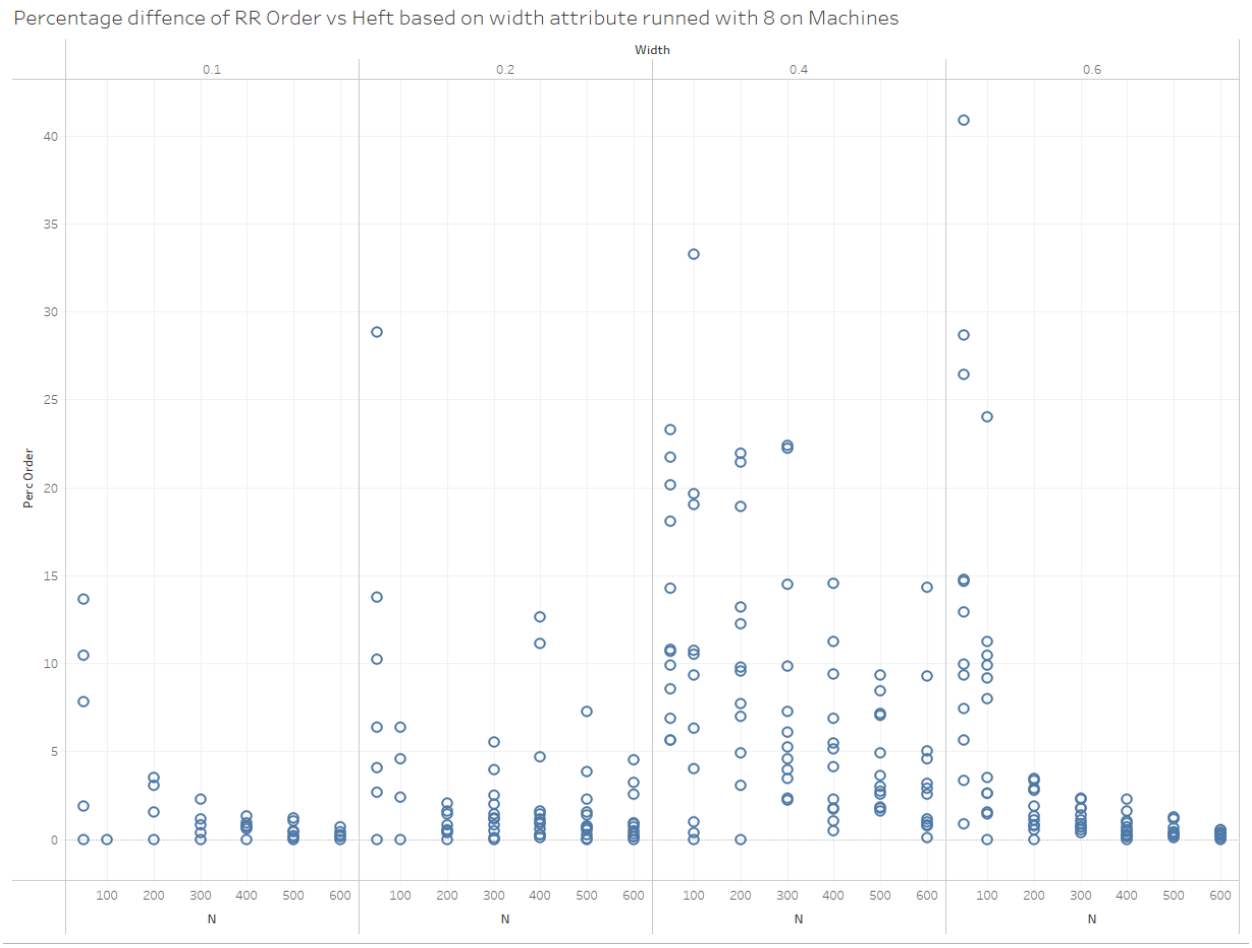
Όλα αυτά τα προβλήματα δεν διαπιστώνονται τόσο έντονα για τον αλγόριθμο του HEFT για έναν πλατύ γράφο γιατί οι εργασίες ανά επίπεδο δεν έχουν αλληλεξαρτήσεις. Με αποτέλεσμα το συντριπτικό μέρος των περιπτώσεων των ροών εργασιών που έχουν σαν όρισμα $fat \leq 0{,}2$ όταν δημιουργήθηκαν να βλέπουν ποσοστό βελτίωσης το οποίο ξεπερνάει το 10%. Ποιο συγκεκριμένα ακριβώς τα ¾ των περιπτώσεων που είδαν ποσοστό βελτίωσης πάνω από 10% είχαν κάτω από $fat \leq 0{,}2$ ενώ ¼ των περιπτώσεων είχαν $fat \geq 0.4$.



Εικόνα 4.8. Στιγμιότυπο από τον κώδικα της εργασίας.

Παρατηρώντας και την εικόνα 4.8 τα παραπάνω στατιστικά δεδομένα παράχθηκαν με την χρήση συναρτήσεων και εργαλείων της βιβλιοθήκης Pandas της Python. Αρχικά διαβάστηκε σε ένα DataFrame το αρχείο excel με τα δεδομένα, έπειτα υπολογίστηκε η ποσοστιαία διαφορά μεταξύ του χρόνου εκτέλεσης του αλγορίθμου του HEFT και του αλγορίθμου του Ruin Order για κάθε ροή εργασιών. Ενώ μετά ξεχωρίστηκαν τα δεδομένα βάση του πλάτους τους. Και τέλος μετρήθηκαν πόσο από αυτά είχαν ποσοστό βελτίωσης πάνω από 10%.

Όμως τα αποτελέσματα έχουν την ίδια μορφή όταν το υπολογιστικό περιβάλλον έχει περισσότερους υπολογιστικούς πόρους στην διάθεση του. Με την σημαντική διαφορά ότι λόγω των διαθέσιμων πόρων αργεί να φανεί το φαινόμενο. Στην παραπάνω περίπτωση οι ροές είχαν στην διάθεση τους μόλις 4 υπολογιστικούς πόρους με περιορισμένη υπολογιστική δύναμη ο καθένας. Στην περίπτωση περισσότερων υπολογιστικών πόρων και πιο συγκεκριμένα αν διπλασιαστούν επί δύο δηλαδή 8 υπολογιστικούς πόρους οι οποίοι ποικίλουν υπολογιστικής δύναμης ο καθένας, το διάγραμμα από την εικόνα 4.7 παίρνει την εξής μορφή:



Εικόνα 4.9. Στιγμιότυπο από διάγραμμα που δημιουργήθηκε με το εργαλείο οπτικοποίησης Tableau.

Από το παραπάνω διάγραμμα μπορεί κανείς να διαπιστώσει πως διπλασιάζοντας τους υπολογιστικούς πόρους του συστήματος έχει ως αποτέλεσμα να αλλάξει η πορεία των αποτελεσμάτων. Συγκεκριμένα αναλύοντας τα παραπάνω αποτελέσματα ξανά με την βοήθεια της βιβλιοθήκης Pandas βγήκαν τα εξής αποτελέσματα: Όλες οι ροές εργασιών που σημείωσαν άνω από 10% ποσοστό βελτίωσης ήταν μόλις 39. Ενώ 32 από αυτές τις περιπτώσεις είχαν πλάτος $fat \geq 0.4$ και οι υπόλοιπες 7 $fat \leq 0.2$.

Διπλασιάζοντας άλλη μια φορά την υπολογιστική δύναμη από 8 υπολογιστικούς πόρους στους 16 παρατηρείται το ίδιο φαινόμενο ακόμα πιο έντονα. Πιο συγκεκριμένα από τις 55 ροές εργασιών που σημείωσαν βελτίωση άνω των 10% οι 50 είχαν πλάτος τουλάχιστον $fat \geq 0.4$ ενώ οι υπόλοιπες 5 $fat \leq 0.2$.

Τα ποσοστά αυτά αυξάνονται ανάλογα με τον αριθμό των υπολογιστικών πόρων. Δηλαδή το συγκεκριμένο φαινόμενο μεταφράζεται σε απλά λόγια ως εξής: Όσο αυξάνονται οι υπολογιστικοί πόροι τόσο πιο αποδοτικός είναι ο αλγόριθμος του HEFT σε ροές εργασιών που δεν έχουν πολλές εργασίες ανά επίπεδο, ενώ τόσο λιγότερο αποδοτικός είναι όταν υπάρχουν πολλοί υπολογιστικοί πόροι αλλά οι ροές εργασιών έχουν πολλές εργασίες ανά επίπεδο.

Οι λόγοι που συμβαίνει το παραπάνω φαινόμενο είναι οι εξής:

- Ο συνδυασμός των πολλαπλών υπολογιστικών πόρων με συνδυασμό των πολλαπλών εργασιών σε ένα επίπεδο μεταφράζεται από τον αλγόριθμο του HEFT (και τους περισσότερους δηλαδή) ως πολλές εναλλαγές από έναν υπολογιστικό πόρο σε έναν άλλο. Άρα λοιπόν οι συχνές εναλλαγές του αλγορίθμου δημιουργούν χρονικά κενά στο σύστημα τα οποία αδυνατεί να καλύψει ο αλγόριθμος.

- Επειδή ο αλγόριθμος αναγκάζεται να επιλέξει κάθε φορά το καλύτερο υπολογιστικό πόρο αδιαφορεί για την πιθανότητα του να εξετάσει μία λιγότερη καλύτερη επιλογή η οποία μακροπρόθεσμα θα προσφέρει ίσως καλύτερα αποτελέσματα.

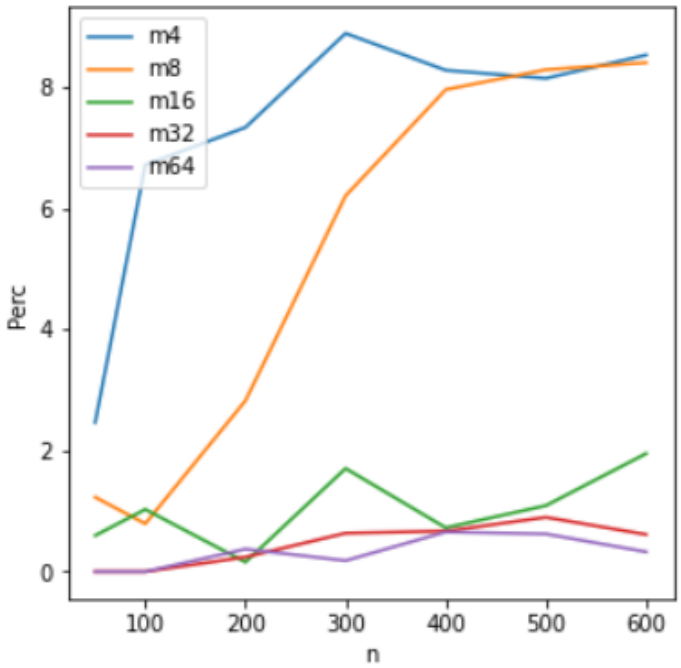
- Ακόμα το να μείνει στον ίδιο υπολογιστικό πόρο μπορεί να φανεί κερδοφόρο για τον αλγόριθμο του HEFT αλλά για τον αλγόριθμο του Ruin Order μπορεί να αποτελέσει πρόβλημα και να κερδίσει από μία πιθανή αλλαγή. Ο λόγος είναι γιατί ακόμα και σε μία εναλλαγή ενός υπολογιστικού πόρου ο Ruin είναι σε θέση να καλύψει το χρονικό κενό που θα μείνει πίσω ενώ παράλληλα λόγω του πως λειτουργεί ο αλγόριθμος μπορεί να επιλέξει μία τελείως διαφορετική προσέγγιση στην σειρά των εργασιών σε σχέση με τον HEFT.

Η επιρροή των υπόλοιπων ορισμάτων αναφορικά με την αποδοτικότητα των αλγορίθμων 4.5.β

Αναφορικά με τα υπόλοιπα ορίσματα τα οποία χρησιμοποιήθηκαν κατά την κατασκευή των γράφων δεν υπήρξε κάποιο άλλο το οποίο ξεχώρισε ιδιαίτερα από μόνο του. Οι ποσοστιαίες βελτιώσεις μειώθηκαν ακόμα περισσότερο όταν η υπολογιστική ισχύς αυξήθηκε. Παρόλα αυτά υπήρξαν σημαντικές σημειώσεις που έχουν ενδιαφέρον να αναφερθούν.

Αρχικά ένα όρισμα το οποίο δεν θα έπρεπε να ξαφνιάζει με την απόδοση του ήταν το regular. Πρακτικά το συγκεκριμένο όρισμα βοηθάει στο να οριστεί το πλάτος του γράφου σε συνδυασμό με την παράμετρο fat. Η διαφορά τους είναι ότι το fat ορίζει των αριθμό των κόμβων ανά επίπεδο βάση του συνολικού αριθμού των κόμβων του γράφου. Βάση του ορίσματος fat πρακτικά παράγεται ένας αριθμός x και με την βοήθεια του regular επιλέγεται ο αριθμός των κόμβων ανάμεσα σε αυτούς το εύρος αριθμών
[regular * x, 1.8 * x].

Επομένως δεν αποτελεί ξαφνικό το γεγονός ότι ο αλγόριθμος του HEFT δυσκολεύτηκε να είναι τόσο αποτελεσματικός πάνω σε ροές εργασιών οι οποίες είχαν μικρό πλάτος ενώ παράλληλα το σύστημα ήταν περιορισμένο σε υπολογιστικούς πόρους. Αυτό μπορεί να φανεί καλύτερα και σε μία γραφική αναπαράσταση με την βοήθεια της βιβλιοθήκης Matplotlib και της Pandas.



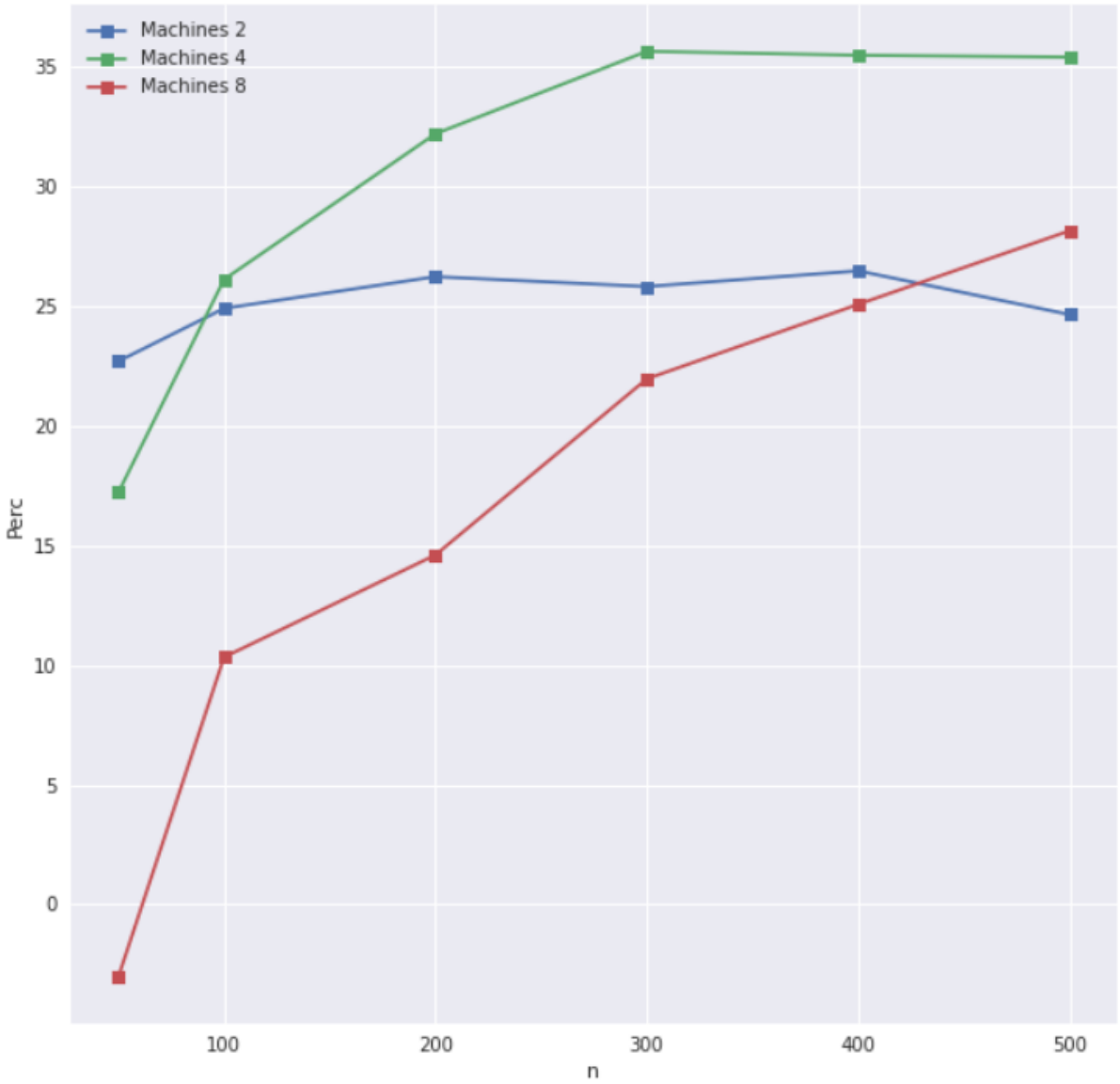
Εικόνα 4.10. Στιγμιότυπο από διάγραμμα που δημιουργήθηκε με την βιβλιοθήκη της Python Matplotlib. [35]

Το παραπάνω διάγραμμα περιγράφει τα εξής: Στον άξονα x βρίσκεται ο αριθμός των κόμβων κάθε ροής εργασιών. Στον άξονα y είναι η ποσοστιαία βελτίωση ανά κατηγορία των ροών εργασιών. Στην κατηγορία στην οποία ομαδοποιήθηκαν είναι ο αριθμός των κόμβων που ανήκει κάθε ροή. Τέλος το παραπάνω διάγραμμα περιλαμβάνει μόνο ροές εργασιών οι οποίες είχαν δεχτεί κατά την κατασκευή τους σαν όρισμα regular = 0.2 .

Από το διάγραμμα παρατηρείται παρόμοια συμπεριφορά με τα προηγούμενα αποτελέσματα αναφορικά με την παράμετρο fat. Όταν το σύστημα έχει περιορισμένη υπολογιστική ισχύ και κλείνεται να αναλάβει ροές εργασιών οι οποίες εκτείνονται σε πολλά επίπεδα η εφαρμογή του αλγορίθμου του HEFT δεν είναι πάντα απαραίτητα αρκετή. Σε συνδυασμό με την μέθοδο του Ruin Order μπορούν να γίνουν βελτιώσεις μέχρι και 50% ανάλογα την μορφολογία του γράφου. Ενώ μία βελτίωση 10% ανά μέσο όρο θεωρείται πολύ πιθανό σενάριο βάση των αποτελεσμάτων που βρέθηκαν παραπάνω.

Ένας συνδυασμός ορισμάτων ο οποίος είχε ενδιαφέρον στην τελική συμπεριφορά των αλγορίθμων είναι αυτός του jump και του density. Όπως προαναφέρθηκε και στο κεφάλαιο 3.1 το όρισμα jump ορίζει τους γονείς ενός κόμβου σε απόσταση επιπέδων βάση του αριθμού jump. Δηλαδή αν έχει δοθεί σαν jump ο αριθμός 2 οι γονείς ενός κόμβου μπορεί να απέχουν από αυτόν μέχρι και δύο επίπεδα. Το όρισμα density από την άλλη ορίζει πόσους γονείς δηλαδή ακμές μπορεί να έχει ένας κόμβος.

Στην συγκεκριμένη ενότητα θα παρουσιαστούν οι ποσοστιαίες βελτιώσεις που δέχτηκαν τα χρονοδιαγράμματα όταν εφαρμόστηκε ο αλγόριθμος Ruin and Recreate στην υπάρχουσα λύση ενός άλλου αλγορίθμου.



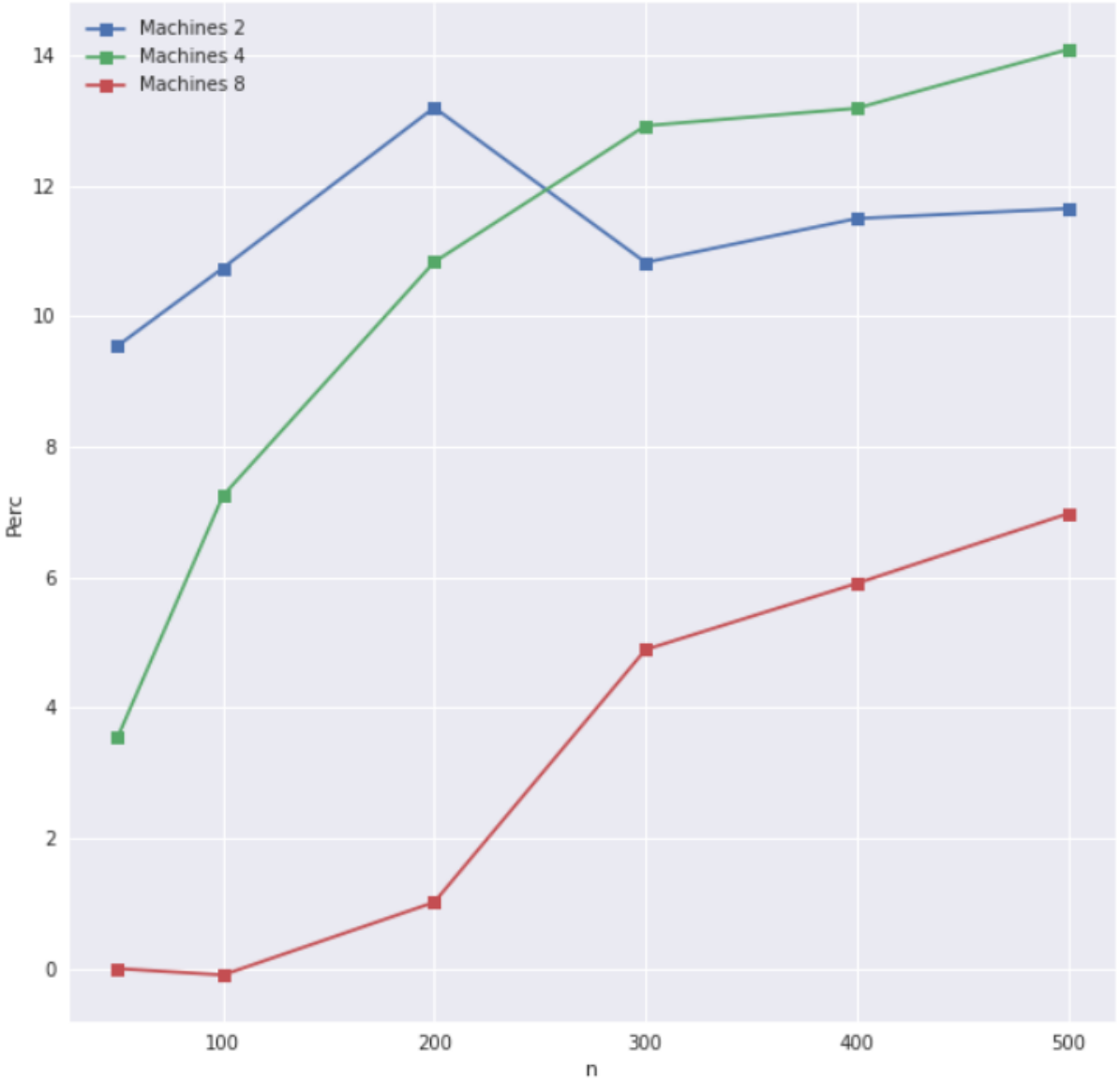
Εικόνα 4.11. Στιγμιότυπο που παρουσιάζει την ποσοστιαία βελτίωση που είχε το χρονοδιάγραμμα όταν εφαρμόστηκε ο αλγόριθμος του Ruin and Recreate στην λύση του αλγορίθμου CPOP.

Παρατηρώντας και από την εικόνα 4.11 η μέση ποσοστιαία βελτίωση του χρονοδιαγράμματος όταν εφαρμόζεται ο Ruin and Recreate σε αποτελέσματα του CPOP φτάνει μέχρι και 35%! Ενώ παρατηρείται επίσης ότι όσο αυξάνεται ο αριθμός των κόμβων τόσο ανεβαίνει και το ποσοστό βελτίωσης. Αυτό συμβαίνει για τους εξής λόγους:

- Στην περίπτωση του CPOP ο αλγόριθμος δημιουργεί τεράστια χρονικά κενά καθώς επιλέγει να βάλει ολόκληρο το critical path σε έναν πόρο και

μόνο. Αυτό έχει ως αποτέλεσμα να μην μπορεί να εκμεταλλευτεί μεγάλο ποσοστό των υπολοίπων υπολογιστικών κόμβων.

- Ο αλγόριθμος κάνει την εξής υπόθεση: Εάν προγραμματιστεί το critical path στον πιο δυνατό υπολογιστικό πόρο εκτός από το υπολογιστικό κόστος των εργασιών θα μειωθεί το επικοινωνιακό κόστος και κατ' επέκταση θα μειωθεί ο συνολικός χρόνος εκτέλεσης της ροής. Όμως τι συμβαίνει σε περιπτώσεις που ο ροή είναι μεγάλη σε πλάτος με αποτέλεσμα να παράγεται μικρό σχετικά critical path? Επίσης στις περιπτώσεις που ο γράφος δεν έχει μεγάλα επικοινωνιακά κόστη η αποδοτικότητα του αλγορίθμου πέφτει δραματικά.



Εικόνα 4.12. Στιγμιότυπο που παρουσιάζει την ποσοστιαία βελτίωση που είχε το χρονοδιάγραμμα όταν εφαρμόστηκε ο αλγόριθμος του Ruin and Recreate στην λύση του αλγορίθμου HEFT.

Παραπάνω βλέπουμε ότι η εφαρμογή του Ruin and Recreate βελτιώνει κατά μέσο όρο έως και 14% τον HEFT. Αυτό συμβαίνει γιατί ο HEFT είναι κατά βάση ένας greedy αλγόριθμος ο οποίος παρόλο που ανατρέχει σε όλη την ροή και βγάζει προτεραιότητες δεν αρκεί καθώς δεν μπορεί να κάνει βραχυπρόθεσμες θυσίες για μακροπρόθεσμα οφέλη. Επομένως όσο περιπλέκονται οι επιλογές που έχει να κάνει και όσο ανεβαίνουν οι πιθανές λύσεις του προβλήματος τόσο λιγότερο πιθανόν είναι να κάνει την σωστή επιλογή. Ενώ ο λόγος που για λίγους κόμβους και για πολλούς πόρους βλέπουμε μικρότερες βελτιώσεις έχει να κάνει με το γεγονός ότι δεν υπάρχει αρκετή πολυπλοκότητα για να υπάρξουν σημαντικές διαφορές και ο HEFT ειδικά μπορεί να βγάλει χρονοδιαγράμματα πιο κοντά στην optimal λύση.

Αλγόριθμοι πολλαπλών ροών εργασιών

Το πρόβλημα του προγραμματισμού ενός μόνο DAG σε ετερογενή συστήματα έχει μελετηθεί εκτενώς. Στην παρούσα εργασία έγινε μία εκτενής αναφορά όσο αναφορά την χρόνο-δρομολόγηση μίας ροής εργασιών. Όμως σε ένα ρεαλιστικό περιβάλλον, είτε γίνεται αναφορά σε πλέγματα είτε σε νέφη, είναι αρκετά σύνηθες η περίπτωση να χρειάζεται να γίνει παραλληλισμός πολλαπλών ροών εργασιών μαζί. Η ανάγκη αυτή μπορεί να είναι αποτέλεσμα ενός απαιτητικού χρήστη ο οποίος μπορεί να χρειάζεται μεγάλη υπολογιστική ισχύ, ενός οργανισμού ο οποίος έχει τεράστιο όγκο από εργασίες οι οποίες περιπλέκονται σε βαθμό που μπορεί να θεωρούνται και διαφορετικές ροές από μόνες τους. Ακόμα μία πιο πιθανή περίπτωση η οποία αποδεικνύει πόσο υπαρκτό είναι το πρόβλημα της χρόνο-δρομολόγησης πολλαπλών ροών εργασιών είναι αυτή ενός υπολογιστικού νέφους το οποίο έρχεται σε θέση να χρόνο-δρομολογήσει όλες τις ροές ταυτόχρονα. Λόγω των πολλαπλών ροών που έρχονται στο σύστημα ανεβαίνουν και οι απαιτήσεις από τους χρήστες. Πλέον μπορεί κάποιος χρήστης να έχει συγκεκριμένες απαιτήσεις από το σύστημα αλλά και το σύστημα να χρειάζεται να κάνει δικιά του διαχείριση για να μπορέσει να ανταπεξέλθει στο φόρτο εργασίας. Έτσι λοιπόν προκύπτουν νέες απαιτήσεις και νέοι στόχοι που ίσως χρειαστεί να εστιάσει κάποιος προγραμματιστής χρόνο-δρομολόγησης. Κάποιοι από τους πιθανούς στόχους είναι οι εξής:

- Ο βασικότερος στόχος εξακολουθεί να είναι παρόμοιος με την χρόνο-δρομολόγηση μίας ροής εργασιών, δηλαδή η μείωση του συνολικού χρόνου εκτέλεσης. Με την σημαντική διαφορά να αποτελεί ότι αντί να χρειάζεται να χρόνο-δρομολογηθεί μία μόνο ροή εδώ υπάρχουν πολλές και με ποικίλες μορφές. Κάτι το οποίο αυξάνει την πολυπλοκότητα και το σύνολο πιθανών λύσεων εκθετικά. Ο κύριος λόγος που παραμένει σαν βασικός στόχος είναι γιατί καλύπτει και τις ανάγκες του χρήστη και του συστήματος. Ένας βασικός στόχος ενός χρήστη είναι πάρα πολύ πιθανό να είναι να θέλει να τρέξουν οι ροές εργασιών που έχει όσο γίνεται πιο αποτελεσματικά και γρήγορα για να εξοικονομήσει χρόνο και χρήμα (το ίδιο ισχύει και στην περίπτωση ενός οργανισμού). Ενώ και ο στόχος του συστήματος είναι να πετύχει την γενικότερη αποσυμφόρηση των υπολογιστικών πόρων για να είναι επαναχρησιμοποιήσιμοι αλλού.

- Ένας άλλος πιθανός στόχος μπορεί να είναι η δίκαια χρόνο-δρομολόγηση αυτών των ροών. Αυτό γιατί μπορεί η ροές που έρχονται μπορεί να ανήκουν σε διαφορετικούς χρήστες. Με τον όρο δίκαια να σημαίνει ότι όλες οι ροές θα πρέπει ανάλογα βέβαια και του μέγεθος τους να έχουν ένα μέσο όρο εκτέλεσης μεταξύ τους.

- Ακόμα θα μπορούσε η επίτευξη ενός συγκεκριμένου επιπέδου ποιότητας μίας υπηρεσίας για τις δεδομένες ροές θα μπορούσε να αποτελεί έναν στόχο. Ένα τέτοιο επίπεδο ποιότητας μίας υπηρεσίας θα μπορούσε να βασίζεται σε προτεραιότητες (για παράδειγμα, μια απαίτηση θα μπορούσε να είναι η ολοκλήρωση της εκτέλεσης μιας συγκεκριμένης ροής το συντομότερο δυνατό) ή άλλους περιορισμούς.

- Στόχος επίσης θα μπορούσε να μην είναι μόνο η βελτιστοποίηση της συνολικής απόδοσης, αλλά και η επίτευξη δικαιοσύνης, η οποία ορίζεται με βάση την επιβράδυνση που θα αντιμετώπιζε κάθε ροής εργασιών ως αποτέλεσμα του ανταγωνισμού για πόρους με άλλες ροές εργασιών.

Μέχρι στιγμής η οπτική προσέγγισης για την λύση του προβλήματος ίσως να είχε σαν επίκεντρο τον χρήστη-πελάτη, τώρα αντιστρέφεται και αυτός που ωφελείται περισσότερο με την επίλυση του συγκεκριμένου προβλήματος ίσως είναι η εκάστοτε εταιρεία που προμηθεύει τις υπηρεσίες ενός υπολογιστικού νέφους. Αυτό γιατί είναι πολύ πιο σύνηθες πρόβλημα για εταιρίες ή μεγάλους οργανισμούς καθώς πολλαπλές ροές εργασιών που χρειάζονται τόσο μεγάλη υπολογιστική ισχύ ξεφεύγει από τα πιθανά θέλω ενός μέσου χρήστη.

Compositions 5.1

Στο συγκεκριμένο άρθρο οι συγγραφείς εισάγουν την έννοια της δικαιοσύνης, η οποία ορίζεται με βάση την επιβράδυνση που θα αντιμετώπιζε κάθε ροή εργασιών (ως αποτέλεσμα της κοινής χρήσης των πόρων με άλλες ροές εργασιών σε αντίθεση με την ύπαρξη των πόρων από μόνη της). Για να επιτευχθεί δικαιοσύνη, ο στόχος είναι να γίνει αυτή η επιβράδυνση ίση για όλες τις ροές εργασιών, χωρίς ωστόσο να επηρεαστεί η συνολική διάρκεια του προγράμματος. Το έγγραφο περιγράφει δύο πολιτικές για τον προγραμματισμό πολλαπλών ροών εργασιών, ειδικά με στόχο τη δικαιοσύνη. Αυτά αξιολογούνται μαζί με τέσσερις άλλες πολιτικές για τον προγραμματισμό πολλαπλών ροών εργασιών (χωρίς ιδιαίτερη εκτίμηση της δικαιοσύνης). Παρουσιάζονται και αξιολογούνται δύο πολιτικές που επικεντρώνονται ιδιαίτερα στην παροχή δικαιοσύνης μαζί με άλλες τέσσερις πολιτικές που μπορούν να χρησιμοποιηθούν για τον προγραμματισμό πολλαπλών ροών εργασιών. Μία ιδέα που βασίστηκε το άρθρο ήταν η ενοποίηση όλων των ροών εργασιών μαζί κάτω από μία ενιαία ροή η οποία μπορεί να προγραμματιστεί βάση ενός αλγορίθμου για μονή ροή εργασιών.

C1 5.1.α

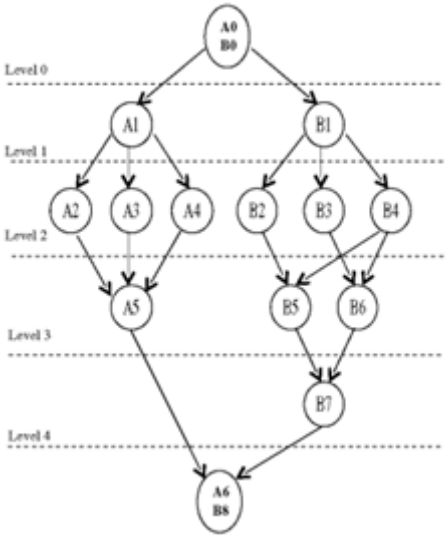
Η πρώτη ιδέα ήταν αυτή που δημιουργεί το σύνθετο γράφημα. Αφού ενοποιηθούν όλοι ο κόμβοι τρέχει τον αλγόριθμο του HEFT στην πλέον ενοποιημένη ροή εργασιών. Η ενοποίηση αυτή γίνεται καθιστώντας τους κόμβους εισόδου όλων των ροών εργασιών άμεσους διαδόχους ενός νέου κόμβου εισόδου και όλους τους κόμβους εξόδου των ροών εργασιών άμεσους προγόνους ενός νέου κόμβου εξόδου. Αυτοί οι δύο επιπλέον κόμβοι (δηλ. νέα είσοδος και νέος κόμβος εξόδου) δεν έχουν υπολογισμό και καμία επικοινωνία μεταξύ αυτών και άλλων κόμβων.

Η τεχνική C1 (Composition 1) ή αλλιώς Common Entry and Common Exit Node πρόκειται για μία προσέγγιση που συγχωνεύει όλες τις ροές σε μια μεγαλύτερη ροή εργασίας χρησιμοποιώντας έναν κοινό κόμβο εισόδου και έναν κοινό κόμβο εξόδου. Κάνοντας όλους τους κόμβους εισόδου των ροών άμεσους διάδοχους του νέου κοινού κόμβο εισόδου και παρομοίως όλοι κόμβοι εξόδου των ροών γίνονται πρόγονοι ενός νέου μεγάλου κόμβου εξόδου. Αυτοί οι κόμβοι δεν έχουν υπολογιστικό κόστος και ούτε επικοινωνιακό κόστος μεταξύ των υπολοίπων κόμβων.

Με αυτή την συγχώνευση μπορεί να επιλεχτεί η ιδανική εργασία σε κάθε επανάληψη του αλγορίθμου καθώς κάθε φορά υπάρχει ένα μεγάλο πλήθος από εργασίες για να επιλεχθούν. Έπειτα τρέχουμε τον αλγόριθμο HEFT πάνω στη συγχωνευμένη ροή εργασιών. Η παραπάνω τεχνική αν και απλοϊκή ως προς εκτέλεση της αλλά ως και προς την ιδέα της είναι φοβερά αποτελεσματική (εξαιτίας του HEFT) έχοντας σαν κύριο αποτέλεσμα να μειώνει με μεγάλη επιτυχία το συνολικό χρόνο που χρειάζονται να τρέξουν όλες οι ροές εργασιών μαζί.

C2 5.1.β

Σαν C2 (Composition 2) οι συγγραφείς του άρθρου ονομάζουν τον Level-Based Ordering. Όπως αναφέρθηκε και σε προηγούμενα κεφάλαια η γενικότερη ιδέα της μελέτης αυτής βασίζεται πάνω την ενοποίηση των ροών σε μία έχοντας με σκοπό την πιο εύκολη διαχείριση τους. Αυτή η προσέγγιση δημιουργεί ένα σύνθετο γράφο ακριβώς με τον ίδιο τρόπο όπως και ο C1 αλλά γίνεται ομαδοποίηση στις εργασίες βάση των επιπέδων (levels) τους.



Εικόνα 5.1. Το παραπάνω στιγμιότυπο απεικονίζει δύο ροές εργασιών οι οποίες ενοποιήθηκαν κάτω από ένα κοινό κόμβο εισόδου και έναν κοινό κόμβο εξόδου. Επίσης μπορεί να παρατηρηθεί ότι οι κόμβοι έχουν χωριστεί σε επίπεδα βάση του composition C2.

Ο προγραμματισμός των εργασιών γίνεται με βάση των επιπέδων που έχουν ανατεθεί σαν όρισμα στην κάθε μία. Αυτό δημιουργεί μία κατάσταση την οποία μπορεί να επωφεληθεί το σύστημα. Επειδή οι εργασίες που βρίσκονται στο ίδιο επίπεδο δεν έχουν καμία εξάρτηση μεταξύ τους πρακτικά μπορούν να συνδυαστούν και να προγραμματιστούν με αρκετά μεγάλη ελευθερία. Έτσι ο προγραμματισμός ανά επίπεδα εργασιών γίνεται εξαιρετικά πιο απλός. Τέλος οι εργασίες επιλέγουν κάποιον υπολογιστικό όρο βάση του ορίσματος EFT όπως και στον αλγόριθμο του HEFT.

C3 5.1.γ

Ο αλγόριθμος του C3 Alternating DAGs έχει σαν στόχο την πιο δίκαια κατανομή των πόρων στις εργασίες. Αυτό το επιτυγχάνει εναλλάσσοντας κάθε φορά εργασίες που χρόνο-δρομολογεί. Πιο συγκεκριμένα δημιουργεί ένα σύνθετο γράφημα με την ίδια διαδικασία όπως ο C1. Ο βασικός βρόχος του αλγορίθμου θα μπορούσε κανείς να πει ότι λειτουργεί σε κύκλους. Ποιο συγκεκριμένα σε κάθε πλήρη iteration του κύκλου ο αλγόριθμος θα πρέπει να έχει χρόνο-δρομολογήσει μία εργασία από κάθε ροή εργασιών που υπάρχει. Αυτό γιατί ο κυρίως στόχος του είναι η δίκαια κατανομή των πόρων. Επομένως δεν γίνεται μία ροή να έχει χρόνο-δρομολογήσει πάνω από μία εργασία παραπάνω συγκριτικά με κάποια άλλη ροή.

Φυσικά αυτό που αδυνατεί να προβλέψει ο αλγόριθμος και κάτι το οποίο θα επιχειρηματολογήσει κάποιος ως προς την βελτίωση του είναι ότι δεν είναι όλες οι εργασίες ίδιες. Επειδή κάποιες εργασίες διαφέρουν δραματικά όσον αφορά τους χρόνους εκτέλεσης θα μπορούσε να το λαμβάνει υπόψιν και να χρονοδρομολογεί Χ παραπάνω υπηρεσίες από τις υπόλοιπες ροές για να μπορέσει να κρατήσει την ισότητα μεταξύ των ροών.

Άρα χρησιμοποιώντας μια round-robin τεχνική. Ο αλγόριθμος επιλέγει μία εργασία και έπειτα διαλέγει τον πόρο στον οποίο θα χρόνο-δρομολογηθεί βάση του ορίσματος EFT.

Multiple Workflow Scheduling 5.2

Στο παρόν υποκεφάλαιο θα γίνει ανάλυση ενός άρθρου που διερευνάει τον προγραμματισμό πολλαπλών ροών εργασιών με προθεσμίες από άκρο σε άκρο σε ένα περιβάλλον που αποτελείται από ένα ετερογενές κατανεμημένο σύστημα πραγματικού χρόνου, χρησιμοποιώντας μια ευρετική λίστα χρονοπρογραμματισμού. Ενώ η φάση επιλογής μίας εργασίας βασίζεται σε συγκεκριμένες πολιτικές. Στην προσέγγισή αυτή, το κόστος επικοινωνίας λαμβάνεται υπόψη για τις περισσότερες πολιτικές ενώ στην φάση επιλογής υπολογιστικού πόρου γίνεται επιλογή του υπολογιστικού πόρου που μπορεί να παρέχει την επιλεγμένη έτοιμη εργασία με τον νωρίτερο εκτιμώμενο χρόνος έναρξης, είτε: (α) χωρίς την εκμετάλλευση πιθανών χρονικών τρυπών ή (β) με εκμετάλλευση του χρόνου αδράνειας, όπως στην περίπτωση των ISH και HEFT, αλλά με τη χρήση τεχνικών bin packing.
Σαν πρωταρχικό στόχο έχουν να εγγυηθούν ότι όλες οι θέσεις εργασίας που φτάνουν στο σύστημα θα τηρήσουν τις προθεσμίες τους. Η απόδοση του αλγορίθμου και οι πολιτικές προγραμματισμού αξιολογούνται με προσομοίωση, κάτω από διάφορους φόρτους εργασίας και επίπεδα ετερογένειας συστήματος. Οι τεχνικές συσκευασίας κάδου ενσωματώνονται στη διαδικασία επιλογής υπολογιστικού πόρου προκειμένου να εκμεταλλευτούν τα χρονικά κενά ενώ παρουσιάζονται και αναλύονται τα αποτελέσματα της προσομοίωσης.

Το σύστημα θεωρείται ότι είναι ένα ετερογενές κατανεμημένο σύστημα σε πραγματικό χρόνο, που αποτελείται από ένα σύνολο P από q υπολογιστικούς πόρους, ο καθένας εξυπηρετεί τη δική του ουρά (μνήμη). Κάθε υπολογιστικός πόρος pi χαρακτηρίζεται από έναν ρυθμό εκτέλεσης li. Δεδομένου ότι το σύστημα είναι ετερογενές, οι ρυθμοί εκτέλεσης του υπολογιστικού πόρου μπορεί να διαφέρουν. Υποτίθεται ότι οι υπολογιστικοί πόροι στο σύστημα είναι πλήρως συνδεδεμένοι, ότι η επικοινωνία είναι χωρίς διαμάχες και ότι ένας υπολογιστικός πόρος μπορεί να λαμβάνει και να στέλνει δεδομένα σε άλλους ταυτόχρονα, ακόμα και όταν εκτελεί μια εργασία. Έτσι, δεν δίνεται προσοχή στις στρατηγικές δρομολόγησης που χρησιμοποιούνται για την επικοινωνία. Οι ρυθμοί μεταφοράς δεδομένων μεταξύ των επεξεργαστών αποθηκεύονται σε ένα matrix V, όπου το στοιχείο mij υποδηλώνει τον ρυθμό μεταφοράς μεταξύ των υπολογιστικών πόρων pi και pj. Ο χρόνος εκκίνησης επικοινωνίας του επεξεργαστή pi λαμβάνεται υπόψη στην τιμή του ρυθμού μεταφοράς mij. Όπως και στην περίπτωση των ρυθμών εκτέλεσης του υπολογιστικού πόρου, οι ρυθμοί μεταφοράς δεδομένων μεταξύ των επεξεργαστών ενδέχεται επίσης να διαφέρουν.

Να αναφερθεί πως στην προκείμενη εργασία για να γίνει καλύτερη ανάλυση και μοντελοποίηση των δεδομένων που παράγει ο παραπάνω αλγόριθμος ο ρυθμός εκτέλεσης του υπολογιστικού πόρου παραμένει σταθερός. Ενώ οι ρυθμοί μεταφοράς

δεδομένων από έναν πόρο σε έναν άλλο έχει να κάνει με το περιβάλλον στο οποίο βρίσκεται ο κάθε υπολογιστικός πόρος.

Οι διάφορες πολιτικές που χρησιμοποιεί ο παραπάνω αλγόριθμος έχουν να κάνουν με την φάση επιλογής μίας εργασίας. Οι παραλλαγές είναι οι εξής:
α) EDF οι εργασίες που φτάνουν στο σύστημα ταξινομούνται βάση της πολιτικής EDF (Earliest Deadline First) η οποία λαμβάνει υπόψη μόνο τις προθεσμίες των θέσεων εργασίας για την ιεράρχηση εργασιών. Κάθε εργασία έχει ένα deadline το οποίο ορίζεται από την ροή εργασιών από την οποία προέρχεται η εκάστοτε εργασία. Αποτυχία εκπλήρωσης αυτής της προθεσμίας θεωρείται ότι όλη η ροή έχει αποτύχει στον στόχο της. β) HLF ο Highest Level First with Estimated Times (HLFET), ή πιο απλά, το πολιτική υψηλότερου επιπέδου πρώτου (HLF). Η πολιτική HLF (Highest Level First) που λαμβάνει υπόψη μόνο τα επίπεδα των εργασιών, αγνοώντας τυχόν περιορισμούς από χρονικά περιθώρια που μπορεί να έχει θέση κάποιος χρήστης. Δηλαδή ανάμεσα σε δύο εργασίες δύο διαφορετικών επιπέδων θα γίνει προσπάθεια χρόνο-δρομολόγησης της εργασίας της οποίας το επίπεδο είναι υψηλότερο. γ) LSTF η πολιτική (Least Space-Time First) λαμβάνει υπόψιν τόσο προτεραιότητα όσο και τους χρονικούς περιορισμούς μεταξύ των εργασιών και των καθηκόντων τους. Καθώς ακόμα, και το κόστος επικοινωνίας μεταξύ των εργασιών σε από μία ροή εργασιών λαμβάνεται υπόψη.

MWCP 5.3

Όπως προαναφέρθηκε και στο κεφάλαιο 2, έχοντας σαν έμπνευση μεθόδους και τεχνικές χρονοδρομολόγησης από την βιβλιογραφία προτάθηκαν νέες καινοτόμες προτάσεις όπως η ιδέα του αλγορίθμου MWCP (Multiple Workflow Complementary Packing algorithm). Ο συγκεκριμένος αλγόριθμος ανατρέχει σε μία ροή και βρίσκει στοιχεία σχετικά με την μορφολογία της. Προσπαθεί έπειτα να κατηγοριοποιήσει της ροές σχετικά με το CCR (Communication to computation ratio) τους. Έτσι δημιουργούνται δύο κατηγορίες ροών μία με μικρό CCR και μία με μεγάλο. Η ιδέα είναι ότι δύο ροές από τις δύο κατηγορίες θα είναι συμπληρωματική η μία στην άλλη. Δημιουργούνται πρακτικά ζευγάρια ροών εργασιών τα οποία μπορούν να συμπληρώσουν το ένα το άλλο. Αρχικά χρόνο-δρομολογείται η ροή η οποία έχει μεγαλύτερα κόστη επικοινωνίας με σκοπό να αφήσει μεγάλα χρονικά κενά πίσω στους υπολογιστικούς πόρους ενώ έπειτα χρόνο-δρομολογείται η αντίστοιχη ροή η οποία λόγου του μικρού επικοινωνιακού κόστους είναι σε θέση να συμπληρώσει τα κενά που αφήνει πίσω η πρώτη. Αυτό επιτρέπει μία δίκαια χρόνο-δρομολόγηση των ροών ενώ καταφέρνει παράλληλα να καλύπτει θέσης αδράνειας των πόρων οι οποίες υπό άλλες συνθήκες έμεναν ανεκμετάλλευτες. Για την συμπλήρωση των συγκεκριμένων χρονικών τρυπών γίνεται χρήση τεχνικών bin packing. Πιο συγκεκριμένα οι τεχνικές οι οποίες χρησιμοποιήθηκαν είναι οι First Fit (FR), Quickest Fit (FF), Best Fit (BF), Worst Fit (WF). Με την Fastest Fit να μην αποτελεί ακριβώς τεχνική bin packing αλλά να εκμεταλλεύεται τα χρονικά κενά έχοντας ως στόχο να πετύχει μικρότερο μήκος χρόνο-διαγράμματος. Γίνεται επίσης συνδυασμός προγραμματισμού λιστών μαζί με χρήση ενός ορίσματος το οποίο ορίζει ποιος υπολογιστικός πόρος θα επιλεχθεί.

Ο αλγόριθμος αρχικά υπολογίζει το CCR κάθε ροής εργασιών. Έπειτα ταξινομεί τις ροές εργασιών βάση του ορίσματος του CCR. Μετά χρόνο-δρομολογεί ανά ροές. Δηλαδή χρόνο-δρομολογεί μία ολόκληρη ροή με το χρονικό όρισμα που έχει δοθεί και συμπληρώνει τα χρονικά κενά βάση του bin packing ορίσματος που έχει δοθεί επίσης, ενώ προσπαθεί σε κάθε επανάληψη και σε κάθε χρονοδρομολόγηση μίας εργασίας πρώτα να γίνει ένταξη αυτής σε ένα χρονικό κενό και μετά σε κάποιο άλλο σημείο ενός υπολογιστικού πόρου.

Διαφορετικά χρονικά ορίσματα 5.3.α

Από την βιβλιογραφία παρατηρήθηκε ότι έγινε χρήση διαφορετικών χρονικών ορισμάτων κατά την επιλογή ενός υπολογιστικού πόρου. Παραδείγματος χάρη στον αλγόριθμο του HEFT γινόταν επιλογή του υπολογιστικού πόρου βάση του χρονικού ορίσματος EFT (Earliest Start Time). Ενώ στην περίπτωση των EDF, HLF, LSTF γίνεται επιλογή ενός υπολογιστικού πόρου βάση του EST (Earliest Start Time). Προφανώς σε κάθε επιλογή ενός πόρου λαμβάνεται υπόψη και ο χρόνος μεταφοράς δεδομένων από έναν υπολογιστικό πόρο σε έναν άλλο, η χρονική στιγμή που τελειώνουν την εκτέλεση τους οι εργασίες γονείς καθώς και η χρονική στιγμή που υπάρχει εντός του πόρου.

Στον προκείμενο αλγόριθμο εξετάστηκαν χρονικά ορίσματα όπως:

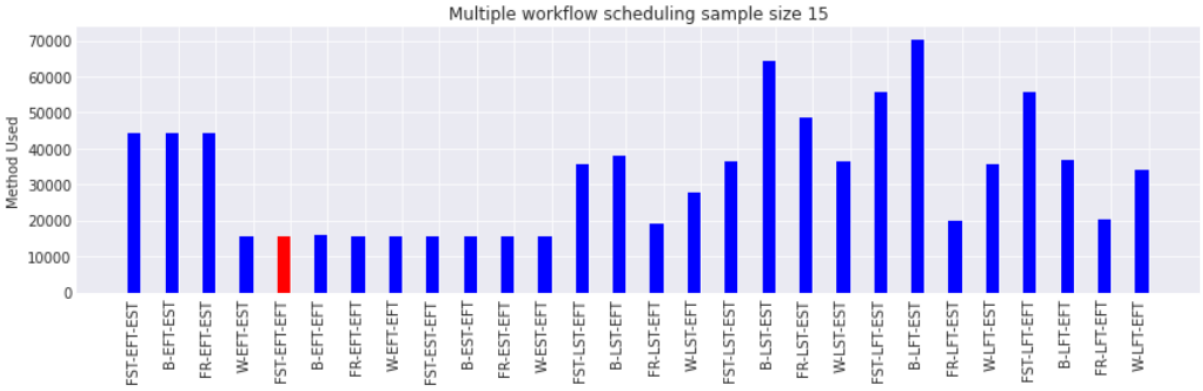
- EST (Earliest Start Time): Επιλέγεται ο πόρος ο οποίος θα καταφέρει να ξεκινήσει την εργασία όσο το δυνατό νωρίτερα.

- EFT (Earliest Finish Time): Επιλέγεται ο πόρος ο οποίος θα καταφέρει να εκτελέσει την εργασία όσο το δυνατό νωρίτερα.

- LST (Latest Start Time): Επιλέγεται ο πόρος ο οποίος θα καταφέρει να ξεκινήσει την εργασία όσο το δυνατό αργότερα.

- LFT (Latest Finish Time): Επιλέγεται ο πόρος ο οποίος θα καταφέρει να εκτελέσει την εργασία όσο το δυνατό αργότερα.

Βάση των παραπάνω ορισμάτων έγιναν διάφοροι συνδυασμοί. Εξετάστηκε η ιδέα του να χρόνο-δρομολογηθεί με ένα διαφορετικό όρισμα κάθε συμπληρωματικό ζευγάρι. Για παράδειγμα η πρώτη από της δύο ροές ενός ζευγαριού να χρόνο-δρομολογηθεί με EST ενώ η δεύτερη με EFT. Επειδή οι παραλλαγές που και οι ροές έχουν χρονοδρομολογηθεί με ‘L’ δηλαδή Latest ήταν εξαιρετικά αργές, απορρίφθηκαν και δεν χρησιμοποιήθηκαν στις αναλύσεις.



Εικόνα 5.2. Στιγμιότυπο αποτελεσμάτων διαφόρων παραλλαγών του αλγορίθμου MWCP.

Στο παραπάνω διάγραμμα παρατηρείται ο μέσος όρος απόδοσης των διαφορετικών παραλλαγών όσο αναφορά τα χρονικά ορίσματα και τα καθώς και των τεχνικών bin packing. Από το διάγραμμα διαπιστώνονται τα εξής:

- Οι παραλλαγές με χρονικά ορίσματα που επέλεγαν τον αργότερο σε χρόνο υπολογιστικό πόρο είτε ήταν LST είτε LFT δεν είχαν βέλτιστα αποτελέσματα ειδικά όταν συνδυάζονταν με την μέθοδο bin packing Best Fit.

- Ενώ όλος παραδόξως οι τεχνικές οι οποίες έκαναν χρήση του LFT με συνδυασμό της τεχνικής bin packing First Fit είχαν εξαιρετική απόδοση παρόλο που δεν κατάφεραν να συναγωνιστούν με τις καλύτερες.

- Οι παραλλαγές οι οποίες είχαν σαν δεύτερο συνθετικό το χρονικό όρισμα EST δεν είχαν πολύ καλή απόδοση κατά μέσο όρο.

- Οι παραλλαγές οι οποίες κατά μέσο όρο έδειξαν τα πιο θετικά αποτελέσματα κατά μέσο όρο ασχέτως με την τεχνική bin packing που χρησιμοποιήθηκε ήταν αυτές που είχαν σαν χρονικά ορίσματα το EST-EFT.

- Πολύ καλή απόδοση ανεξαρτήτως των τεχνικών bin packing που εφαρμόστηκαν ενώ είχαν και μία κατηγορία που είχε την καλύτερη απόδοση συγκριτικά με τις υπόλοιπες ήταν ο συνδυασμός χρονικών ορισμάτων EFT-EFT.

Ο λόγος που ο συνδυασμός χρονικών ορισμάτων EST-EFT είχε εξαιρετική απόδοση ενώ ο EFT-EST όχι έχει να κάνει με διάφορους λόγους. Αρχικά ο στόχος ήταν η συνολική μείωση του ολικού χρόνο-διαγράμματος, το επομένως όταν ο αλγόριθμος κοιτάει για το πότε θα ξεκινήσει μία εργασία δεν μπορεί να προβλέψει και το πότε θα τελειώσει. Έτσι επιλέγει υπολογιστικούς πόρους οι οποίοι μπορεί και με την τελευταία ροή που θα χρόνο-προγραμματιστεί να παράγουν μεγαλύτερα χρονοδιαγράμματα. Ο άλλος λόγος ο οποίος είναι και ο κυριότερος έχει να κάνει με την μορφολογία της κάθε ροής εργασιών. Οι ροές έχουν χωριστεί σε ζεύγη, η δεύτερη ροή από το ζεύγος συνήθως πρόκειται για ροή η οποία έχει εργασίες με

μεγάλους χρόνους εκτέλεσης με αποτέλεσμα να όταν χρόνο-δρομολογούνται μόνο έχοντας σαν γνώμονα να ξεκινήσουν όσο γίνεται πιο γρήγορα δεν ελέγχεται ο χρόνος εκτέλεσης που μπορεί να χρειαστεί η συγκεκριμένη εργασία.

Τέλος ο λόγος για την επιτυχία του ζεύγους EST-EFT είναι γιατί οι βαριές εργασίες υπολογισμού προγραμματίζονται σύμφωνα με το κριτήριο χρόνου έναρξης της πρώτης. Και στις δύο εναλλακτικές λύσεις, πρέπει να σαρώσετε ολόκληρη τη λίστα των πόρων για να επιλέξετε αυτόν που θα ελαχιστοποιεί περισσότερο το EST ή το EFT κριτήριο, αντίστοιχα. Η διαίσθηση πίσω από την εναλλακτική EST είναι να διατηρείται μια ισορροπία μεταξύ της διάρκειας του χρόνου αδράνειας και του συνολικού χρόνου εκτέλεσης των πόρων, ενώ το EFT εφαρμόζεται προκειμένου να ελαχιστοποιηθεί το αντίκτυπο των μεγάλων υπολογιστικών εργασιών πάνω στην συνολική διάρκεια του τελικού χρόνο-διαγράμματος.

Παράδειγμα εφαρμογής και η εφαρμογή διαφόρων τεχνικών bin packing 5.3.β

Όπως είναι γνωστό το πρόβλημα του bin packing αφορά τη συσκευασία ενός συνόλου αντικειμένων σε ένα σύνολο κάδων, χρησιμοποιώντας όσο το δυνατόν λιγότερους κάδους. Μερικοί από τις πιο δημοφιλείς πολιτικές συσκευασίας κάδου είναι: First Fit (FF), Quickest Fit (QF), Best Fit (BF) και Worst Fit (WF). Με το QF να αποτελεί το πιο απλό μεταξύ αυτών των τεχνικών. Τοποθετεί ένα αντικείμενο στον πρώτο κάδο που βρει ότι χωράει. Το BF από την άλλη πλευρά, εκχωρεί ένα αντικείμενο στο κάδο όπου αφήνει τον ελάχιστο δυνατό αχρησιμοποίητο χώρο. Αντίθετα, το WF τοποθετεί ένα αντικείμενο στον κάδο όπου αφήνει τον μέγιστο δυνατό αχρησιμοποίητο χώρο. Σε αυτή την εργασία, η εκμετάλλευση πιθανών χρονικών τρυπών βασίζεται είτε σε FF, QF, BF ή WF. Εξ όσων είναι γνωστό, η εκμετάλλευση των χρονικών τρυπών ενός χρονοδιαγράμματος με τη ρητή χρήση τεχνικών συσκευασίας κάδων δεν έχει συζητηθεί εκτενώς από την βιβλιογραφία.

Για την καλύτερη κατανόηση του αλγορίθμου σε σχέση με την εφαρμογή των τεχνικών bin packing παράχθηκε το εξής παράδειγμα λειτουργίας του MWCP.

Έστω σε ένα υποθετικό σενάριο υπάρχουν δύο μηχανές ($v1$ και $v2$) με ικανότητα επεξεργασίας 1 και 2, αντίστοιχα ($v2$ επεξεργάζεται εργασίες δύο φορές πιο γρήγορα σε σύγκριση με το $v1$). Επίσης, ο ρυθμός μεταφοράς μεταξύ τους είναι 1 μονάδα δεδομένων ανά μονάδα χρόνου. Στα γραφήματα 5.3 έως 5.6, υπογραμμίζουν τη σημασία των μεθοδολογιών συσκευασίας κάδου στο πρόβλημα προγραμματισμού πολλαπλών ροής εργασιών καθώς και τα δυνατά και τα αδύνατα σημεία κάθε μεθόδου. Επιλέχθηκαν να παρουσιαστούν τέσσερα παραδείγματα το καθένα με διαφορετική συσκευασία κάδου που νικάει κάθε φορά. Θεωρείται ότι υπάρχει μία ροή εργασιών (wx) η οποία έχει μία εργασία ανά επίπεδο και που αποτελείται από τέσσερις εργασίες ($t1$ έως $t4$) ότι φτάνει στο σύστημα τη χρονική στιγμή 0. Σημαντική σημείωση ότι εκείνη τη στιγμή και το $v1$ και το $v2$ έχουν ήδη προγραμματίσει την εκτέλεση των ροών εργασίας $w1$ έως $wx-1$ (γκρίζες περιοχές). Ως αποτέλεσμα, αρκετές χρονοθυρίδες αδράνειας (χρονικά κενά προγραμματισμού) έχουν ήδη δημιουργηθεί πριν από την εκτέλεση του wx. Για την ακρίβεια, τα κενά είναι οι εξής:

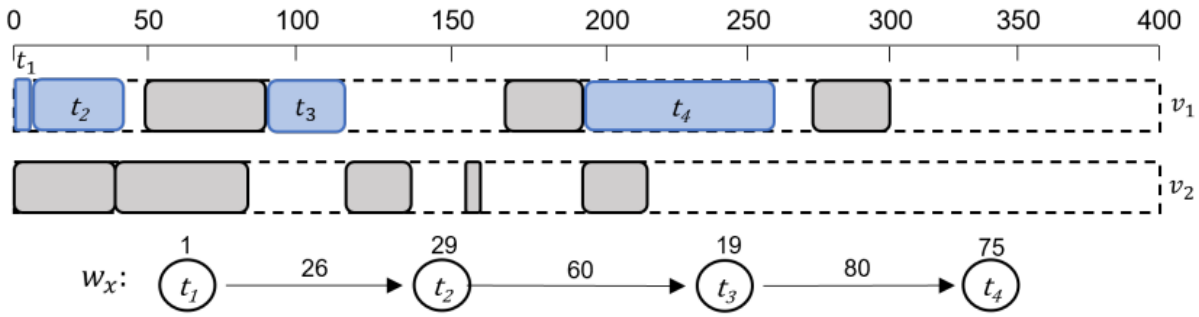
Κενά που βρίσκονται στον υπολογιστικό πόρο $v1$

- $h1$ = [0, 37,5]
- $h2$ = [87,5, 156,5]
- $h3$ = [176,5, 270]

Κενά που βρίσκονται στον υπολογιστικό πόρο $v2$

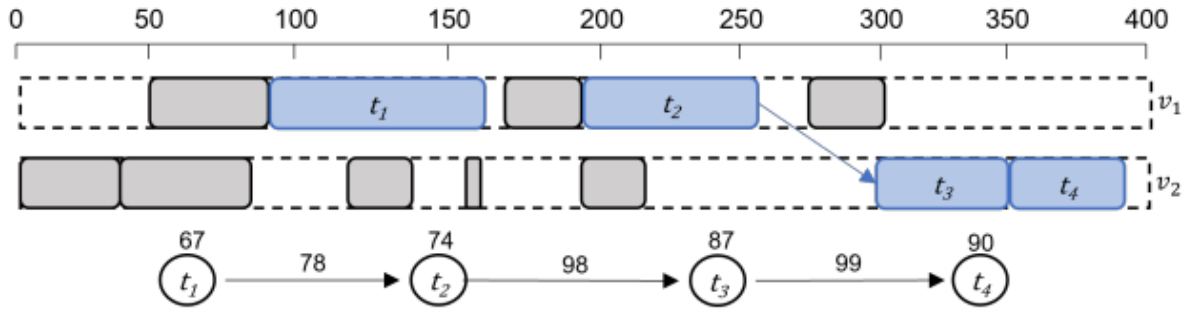
- $h4$ = [80,5, 119,5]
- $h5$ = [131.5, 150]
- $h6$ = [152, 174.5]

Τα κατειλημμένα διαστήματα και τα χρονικά διαστήματα αδράνειας παραμένουν τα ίδια σε όλα τα παραδείγματα, ωστόσο, η απαίτηση υπολογισμού και επικοινωνίας του wx ποικίλλει. Παρουσιάζεται το πρόγραμμα μόνο για τη μέθοδο συσκευασίας του κερδισμένου κάδου και παραλείφτηκαν να σχεδιαστούν τα χρονοδιαγράμματα για τις μεθόδους επιλαχόντων.



Σχήμα 5.5. Σχήμα χρόνο-δρομολόγησης που προέκυψε με την εφαρμογή του αλγορίθμου MWCP που ευνοεί την χρήση του First Fit.

Στο σχήμα 5.3, η μέθοδος First Fit υπερτερεί των BF, WF και QF με συνολικό μήκος ροής να φτάνει 251,5. Το συνολικό μήκος του BF ισούται με 256,5. Το μήκος εκτέλεσης της ροής WF ισούται με 342,0 και το μήκος εκτέλεσης της ροής QF ισούται με 254,5. Η First Fit εκμεταλλεύεται το πρώτο κενό προγραμματισμού $h1$ για να προγραμματίσει μαζί τις μισές από τις εργασίες του wx στον καλύτερο διαθέσιμο χρόνο έναρξης. Ενώ το συνολικό μήκος και των δύο ροών παραμένει το ίδιο η ροή εργασιών wx μείωσε το δικό της συνολικό μήκος εκτέλεσης. Αυτό επιτεύχθηκε μόνο με την First Fit. Ενώ ο λόγος που η Best Fit έφτασε κοντά στην βέλτιστη λύση αλλά δεν τα κατάφερε έχει να κάνει με το γεγονός ότι επέλεξε να προγραμματιστεί η t2 στο κενό $h4$ = [80,5, 119,5] του $v2$ καθώς εάν προγραμματιστεί η εργασία t2 στο συγκεκριμένο κενό θα μειώσει το χρονικό χάσμα που υπήρχε λόγο του κενού.

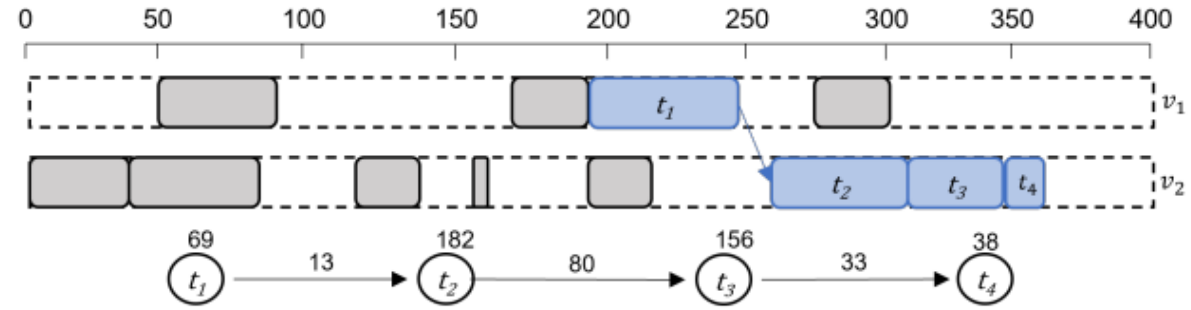


Σχήμα 5.5. Σχήμα χρόνο-δρομολόγησης που προέκυψε με την εφαρμογή του αλγορίθμου MWCP που ευνοεί την χρήση του Best Fit.

Στο σχήμα 5.4, η μέθοδος Best Fit κερδίζει με συνολικό χρόνο όλης της χρόνο-δρομολόγησης 388,0. Το εύρος τιμών FF και QF ισούται με 394,5 ενώ το WF είχε ως αποτέλεσμα που ισούται με το 438,0. Τόσο το πρόγραμμα FR όσο και η τεχνική QF χρόνο-δρομολόγησαν την wx ως εξής:

- $t1$ = [80,5 − 114,0] σε $v2$
- $t2$ = [183,0 − 257,0] σε $v1$
- $t3$ = [306.0 − 349.5]
- $t4$ = [349.5 − 394.5] σε $v2$

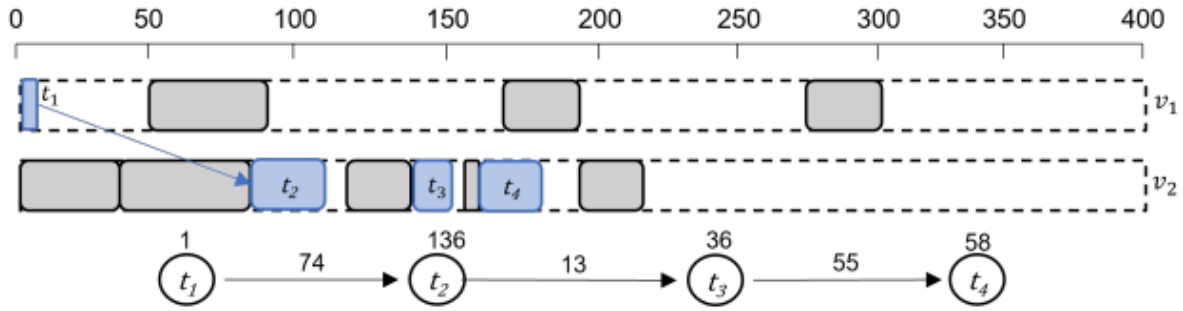
Αντί να επιλεχθεί το χρονικό κενό $h4$ για να φιλοξενήσετε η $t1$, ο Best Fit προγραμματίζει το $t1$ στην $h2$. Ως εκ τούτου, αποφεύγει το μεγάλο κόστος επικοινωνίας μεταξύ $t1$ και $t2$. Σε αντίθεση με τις τεχνικές FF και QF. Η FF συγκεκριμένα επέλεξε το πρώτο χρονικό κενό που χώραγε η $t1$ που στην προκειμένη περίπτωση ήταν η $h2$. Ενώ η FF επέλεξε το χρονικό κενό το οποίο θα εκτελούσε την εργασία $t1$ πιο γρήγορα από τα υπόλοιπα και λόγω της άπληστης φύσης της τεχνικής δεν μπόρεσε να δει ότι θυσιάζοντας ένα καλύτερο αποτέλεσμα προσωρινά θα μειώσει το συνολικό κόστος.



Σχήμα 5.5. Σχήμα χρόνο-δρομολόγησης που προέκυψε με την εφαρμογή του αλγορίθμου MWCP που ευνοεί την χρήση του Worst Fit.

Στο τρίτο παράδειγμα στο Σχήμα 5.5, η τεχνική WF παρουσιάζει περίπου 11% καλύτερη απόδοση σε σύγκριση με τις ανταγωνιστές μεθόδους με συνολικό χρόνο εκτέλεσης 352,5. Ενώ οι υπόλοιπες FF, BF και QF ολοκληρώνουν με συνολικό

χρόνο εκτέλεσης σε 397.0. Το FF και το QF τοποθετούν την $t1$ στο $h4$ στο 80,5 έως το 115,0 ενώ το BF τοποθετεί την ίδια εργασία στο $h2$ στο 87,5 έως το 156,5. Οι εργασίες $t2$, $t3$ και $t4$ τοποθετούνται στο $v1$ ως εξής: $t2$ = [176,5−267,5], $t3$ = [300−378,0] και $t4$ = [378,0−397,0]. Σε αντίθεση με την τεχνική WF, η $t1$ εκτελείται στο χρονικό κενό που έχει ως αποτέλεσμα τον υψηλότερο υπολειπόμενο χώρο, δηλαδή $h3$. Παρά το γεγονός ότι η $t1$ έχει αναβληθεί σημαντικά, αυτό επιτρέπει στις εργασίες $t2$, $t3$ και $t4$ να εκτελούνται στο πιο γρήγορο μηχάνημα χωρίς να υπάρχουν επικοινωνιακές καθυστερήσεις. Εάν για κάποιο λόγο είχαν επιλεχθεί προηγούμενα χρονικά κενά για την χρόνο-δρομολόγηση της $t1$ (όπως και στις άλλες περιπτώσεις δηλαδή) ο αλγόριθμος δεν θα άλλαζε σύντομα υπολογιστικό πόρο καθώς δεν θα τον σύμφερε, καθώς το χρονικό όρισμα με το οποίο επιλέγουν υπολογιστικό πόρο είναι το EFT.



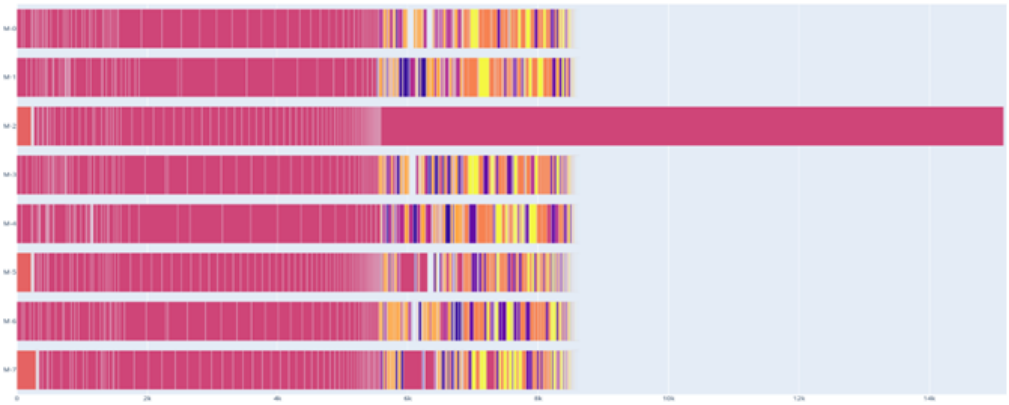
Σχήμα 5.6. Σχήμα χρόνο-δρομολόγησης που προέκυψε με την εφαρμογή του αλγορίθμου MWCP που ευνοεί την χρήση του Quickest Fit.

Στο τέταρτο και τελευταίο παράδειγμα στο Σχήμα 5.6, παρουσιάζεται η QF ως αδιαμφισβήτητο νικήτρια μέθοδος με συνολικό χρόνο εκτέλεσης μόλις 166,5. Ενώ το ο χρόνος του FF ανέρχεται στο 205,5 του BF ισούται με 276,5 και του WF ισούται με 329. Όταν οι απαιτήσεις υπολογισμού είναι σχετικά μικρές, το QF μπορεί να ξεπερνάει τις υπόλοιπες 3 τεχνικές bin packing.

CTF 5.4

Ο αλγόριθμος CTF (Critical Tasks First) εμπνεύστηκε κάποιες ιδέες από τον αλγόριθμο του CPOP αλλά διαφοροποιείται τελείως όσο αφορά την εκτέλεση και την γενικότερη λειτουργεία του. Στην προσπάθεια εύρεσης μία μεθόδου η οποία να βελτιώνει τον συνολικό χρόνο εκτέλεσης όλων των ροών εργασιών μαζί, παρατηρήθηκε ένα φαινόμενο το οποίο είχε τόσο μεγάλη επίπτωση πάνω στο τελικό χρονοδιάγραμμα που μακράν επισκίαζε κάθε μέθοδο και στρατηγική χρονοδρομολόγησης που είχε επιλεχθεί για να προγραμματιστούν οι ροές εργασιών μέχρι στιγμής.

Αυτό το φαινόμενο εμφανίζεται όταν υπάρχουν εργασίες οι οποίες έχουν τόσο μεγάλο υπολογιστικό κόστος οι οποίες επισκιάζουν όλες τις υπόλοιπες ροές εργασιών.



Εικόνα 5.7. Στιγμιότυπο χρονοδιαγράμματος το οποίο προγραμματίστηκε με την μέθοδο C1.

Από το παραπάνω στιγμιότυπο είναι εμφανές ότι είτε εφαρμόζονταν οι τεχνικές bin packing είτε κάποια άλλη χρονικό όρισμα με σκοπό να γίνει καλύτερη εκμετάλλευση των πόρων, δεν υπάρχουν μεγάλες ελπίδες για κάποια βελτίωση στο τελικό αποτέλεσμα, χωρίς αυτό να σημαίνει ότι δεν γίνεται. Το παραπάνω διάγραμμα όμως αξίζει να αντιμετωπιστεί με διαφορετικό τρόπο.

Η ιδέα του CTF είναι ότι προσπαθεί να αναγνωρίσει μέσα σε όλες τις ροές εργασιών εργασίες οι οποίες χρειάζονται τέτοιους δυσανάλογους χρόνους εκτέλεσης συγκριτικά με το υπόλοιπο μέσο όρο των εργασιών.

Η μέθοδος αναγνώρισης των critical tasks μπορεί να οριστεί ως εξής: Οποιαδήποτε εργασία έχει μέσο όρο κόστος εκτέλεσης σε όλους τους υπολογιστικούς πόρους και λαμβάνει σε ποσοστιαία κλίμακα πάνω από 5% του συνολικού χρόνου εκτέλεσης όλων των εργασιών θεωρείται ως critical. Η προτεραιότητα των εργασιών αυτών αλλάζει δραματικά σε σχέση με τις υπόλοιπες. Πρώτο μέλημα του αλγορίθμου είναι να χρονοδρομολογηθούν το συντομότερο δυνατό τα critical tasks καθώς και οι εργασίες γονείς τους φυσικά καθώς δεν γίνεται να ξεκινήσει η χρονοδρομολόγηση τους πριν χρόνο-δρομολογηθούν αυτοί. Αφού χρονοδρομολογηθούν οι εργασίες αυτές το υπόλοιπο πρόγραμμα μπορεί να χρονοδρομολογηθεί με τον ίδιο τρόπο όπως ο αλγόριθμος του HEFT. Αυτό επιτρέπει βέλτιστη κατανομή των εργασιών βάζοντας σε προτεραιότητα μόνο τις εργασίες οι οποίες χρειάζονται να τρέξουν το συντομότερο δυνατό.

Η προτεραιότητα στον CTF λειτουργεί ανάλογα με το ποιο μονοπάτι μέχρι μία critical εργασία η οποία καταλαμβάνει το μεγαλύτερο ποσοστό του συνολικού χρόνου εκτέλεσης. Για παράδειγμα έστω ότι μία εργασία η οποία καταλαμβάνει ποσοστιαία ένα 5% από το συνολικό χρόνο εκτέλεσης των εργασιών. Εάν υπάρχει μία ακόμα η οποία καταλαμβάνει ένα 10% στο τέλος αυτή που καταλαμβάνει το περισσότερο καθώς και όλες πριν από αυτήν θα είναι και αυτές οι οποίες θα έχουν μεγαλύτερη προτεραιότητα καθώς γιατί θα είναι και αυτές θα επηρεάσουν παραπάνω το τελικό αποτέλεσμα όπως και στο γράφημα παραπάνω.

Στις περιπτώσεις που χρειάστηκε να γίνει χρήση του CTF παρατηρήθηκε ακόμα ότι η κάλυψη χρονικών τρυπών δεν επηρέασε ιδιαίτερα το τελικό αποτέλεσμα.

Αυτό συμβαίνει γιατί όπως παρατηρείται και από την εικόνα 5.7 δεν επισκιάζονται όλες οι υπόλοιπες ροές και εργασίες από μία εργασία η οποία μακράν ξεπερνάει τις υπόλοιπες από πλευράς χρόνους εκτέλεσης. Όμως όταν ο στόχος έχει να κάνει με την να αξιοποίηση ενός υπολογιστικού πόρου πλήρως τότε η χρήση των χρονικών τρυπών ακόμα και στις περιπτώσεις που χρειάζεται ο CTF είναι χρήσιμες.

Όταν έχουν έρθει σε ένα σύστημα για χρονοδρομολόγηση ροές εργασιών που ανάμεσα τους υπάρχουν πολύ μεγάλες διαφορές στους χρόνους εκτέλεσης των εργασιών, και ο κύριος στόχος του συστήματος είναι η μείωση του συνολικού χρονοδιαγράμματος τότε η χρήση του CTF είναι μία αξιόπιστη λύση.

Η υλοποίηση του αλγορίθμου μπορεί να σπάσει σε μερικά διακριτά βήματα. Αρχικά αναλύονται όλες οι εργασίες από κάθε ροή εργασιών. Βάση της ποσοστιαίας μερίδας που καταλαμβάνει η εργασία σχετικά με το συνολικό χρόνο εκτέλεσης όλων των εργασιών, ορίζεται αυτή και όλα τα μονοπάτια που οδηγούν μέχρι αυτήν ως critical tasks και λαμβάνουν τον ανάλογο αριθμό προτεραιότητας. Το ποσοστό από το οποίο από εκεί και πάνω θεωρείται μία εργασία ως critical ενδείκνυται να είναι σχετικά μικρό καθώς παρατηρήθηκε πως ένα ποσοστό ανάμεσα στο 4-10% είναι ιδανικό για την εύρεση των εργασιών που έχουν την μεγαλύτερη επίπτωση στο χρονοδιάγραμμα.

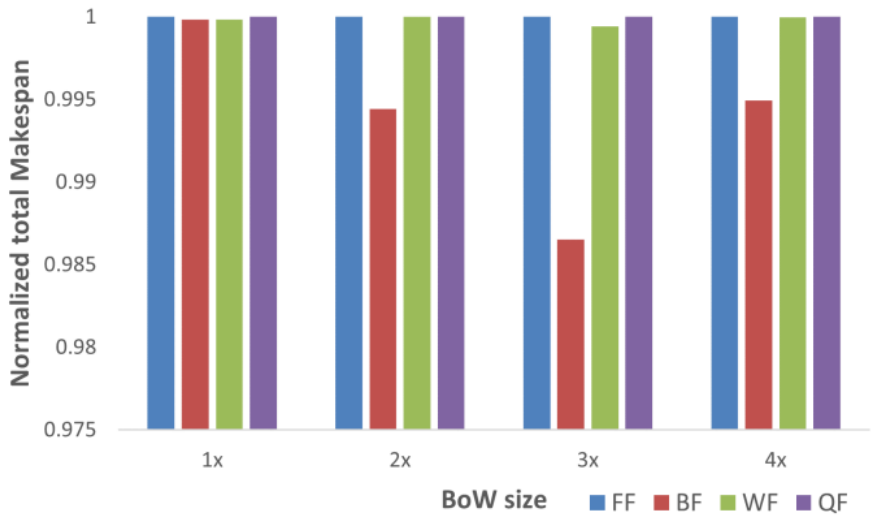
Πειραματική αξιολόγηση 5.5

Στο παρόν υποκεφάλαιο συγκρίθηκαν ο MWCP με κάποιες ομάδες αλγόριθμων οι οποίες μείωναν το συνολικό χρόνο εκτέλεσης των ροών. Οι αλγόριθμοι που εξετάστηκαν ήταν ο MWCP μαζί με όλες τις παραλλαγές τους αναφορικά με τις bin packing τεχνικές και σαν χρονικό όρισμα είχε το ζεύγος EST-EFT. Δηλαδή η πρώτη ροή εργασιών του ζεύγους χρόνο-δρομολογείται με EST και η δεύτερη με EFT.

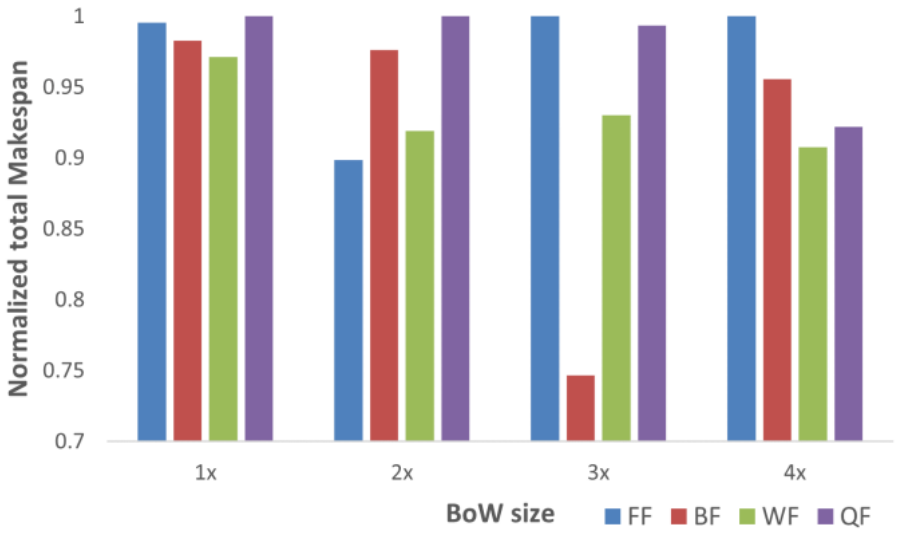
Για την πειραματική προσομοίωση έγινε χρήση 7 διαφορετικών τύπων ροών εργασιών από τα ρεαλιστικά δεδομένα ως μέρος που διανέμονται του συστήματος διαχείρισης ροής εργασιών Pegasus [3] . Ροές εργασιών βιοπληροφορικής περιλαμβάνουν Genome, Epigenomics, SoyKB και SRA Search. Οι ροές εργασίας Αγροοικοσυστήματος και Αστρονομίας περιλαμβάνουν Cycles και Montage, ενώ η ροή εργασίας Seismic Cross Correlation (SCC) αντιπροσωπεύει το πεδίο της Σεισμολογίας. Τα συζητηθέντα πρόκειται για ροές εργασίας που χρησιμοποιούνται εκτενώς για πειραματισμούς στη βιβλιογραφία καθώς καλύπτουν όλα τα βασικά πρότυπα εκτέλεσης όπως η διοχέτευση, η διαδικασία, η συγκέντρωση δεδομένων, η διανομή δεδομένων και η αναδιανομή δεδομένων. Όλες οι ροές εργασιών παρέχονται σε μορφή JSON που περιγράφει την εκτέλεσή τους στο Chameleon Cloud [7] και, μεταξύ άλλων, τη δομή τους, το συνολικό τον αριθμό των εργασιών και την ποσότητα μεταφοράς δεδομένων μεταξύ διασυνδεδεμένων εργασιών. Τέλος, εξετάστηκε μια παρτίδα ροών εργασίας διαφορετικού μεγέθους, π.χ., 7, 14, 18 και 56

ροές εργασίας ανά παρτίδα. Θεωρείται ένα ενιαίο κέντρο δεδομένων που προσφέρει ετερογενή πόρους διαφορετικών υπολογιστικών δυνατοτήτων. Για κάθε πείραμα, μεταβάλλεται ο αριθμό των υπολογιστικών πόρων από 2 έως 8 ενώ ο καθένας έρχεται με διαφορετική ταχύτητα CPU. Υποθέτετε ότι η επικοινωνία μεταξύ των πόρων γίνεται με συγκεκριμένη ταχύτητα χρησιμοποιώντας bandwidth 10, 20, 50 και 100 Mbps ανά πείραμα. Η απόδοση των αλγορίθμων αξιολογούνται ως προς το γενικό χρόνο εκτέλεσης των ροών (M). M είναι ο χρόνος που μεσολάβησε μεταξύ της έναρξης και του τερματισμού του προγράμματος. Αξιολόγησης της απόδοσης:

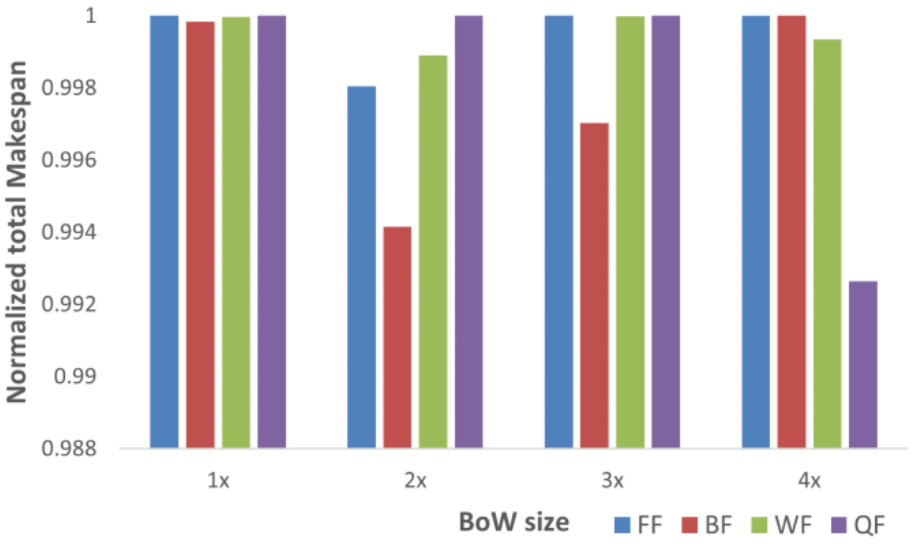
Αρχικά, εκτελείται ένα σύνολο πειραμάτων για διαφορετικές μεθοδολογίες bin packing. Στα Γραφήματα 5.8 και 5.9 παρουσιάζεται το συνολικό makepsan (M) για 7 ($1x$) έως 56 ($8x$) ροές εργασίας και για μικρό εύρος ζώνης (BW = $10Mbps$). Παρόμοια, στα Γραφήματα. 5.10 και 5.11 παρατηρούμε το παρόν M όταν band width ισούται με BW = $100Mbps$.



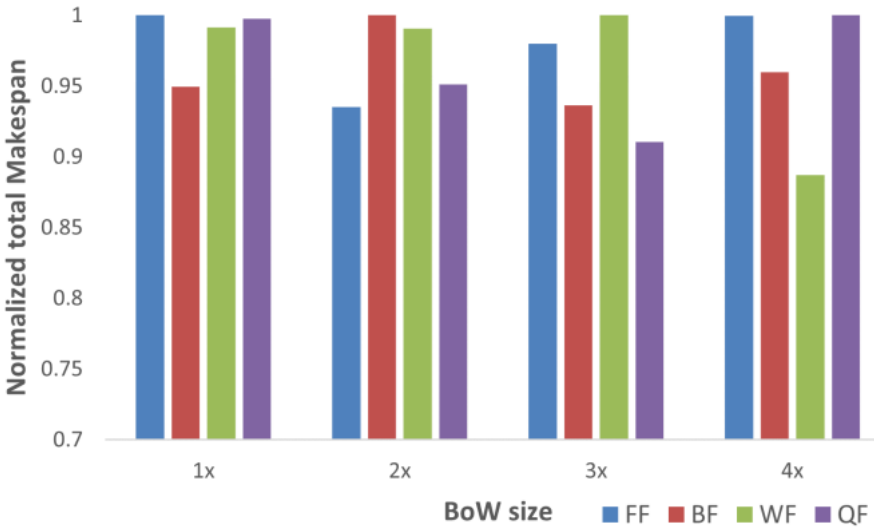
Γράφημα 5.8. Γράφημα το οποίο απεικονίζει τις διαφορές των αλγορίθμων που MWCP που έτρεξαν με 2 υπολογιστικούς πόρους και band width ίσο με 10 MBs.



Γράφημα 5.9. Γράφημα το οποίο απεικονίζει τις διαφορές των αλγορίθμων που MWCP που έτρεξαν με 8 υπολογιστικούς πόρους και band width ίσο με 10 MBs.



Γράφημα 5.10. Γράφημα το οποίο απεικονίζει τις διαφορές των αλγορίθμων που MWCP που έτρεξαν με 2 υπολογιστικούς πόρους και band width ίσο με 100 MBs.

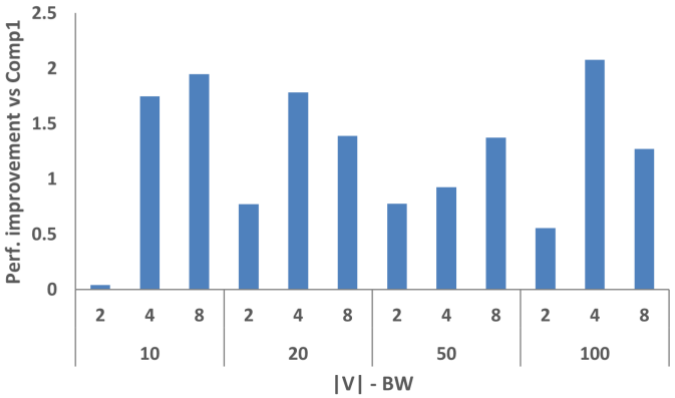


Γράφημα 5.11. Γράφημα το οποίο απεικονίζει τις διαφορές των αλγορίθμων που MWCP που έτρεξαν με 8 υπολογιστικούς πόρους και band width ίσο με 100 MBs.

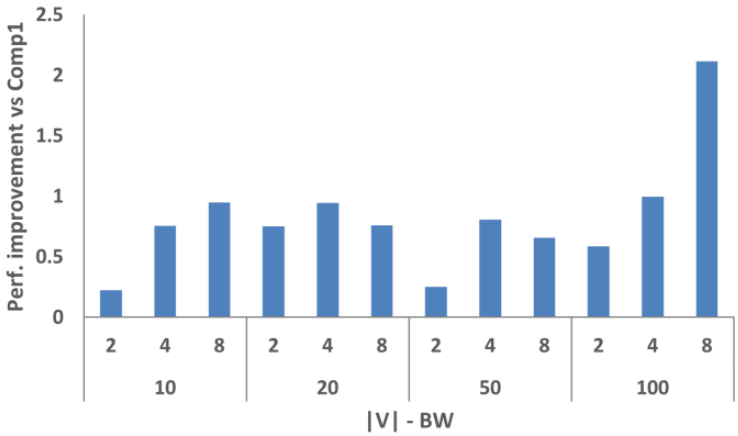
Πρώτον, γίνεται σαφές ότι για ένα μικρό αριθμό μηχανών οι διαφορές απόδοσης είναι μάλλον μικρές, κάτι που είναι λογικό αφού σπάνια συμβαίνει παραλληλισμός μεταξύ των εργασιών. Η απόδοση του WF υποβαθμίζεται καθώς ο αριθμός των ροών εργασίας αυξάνεται. Αυτό μπορεί να εξηγηθεί ως η υπολειπόμενη αχρησιμοποίητη χωρητικότητα υπολογισμού που απομένει μετά Η κατανομή εργασιών θα γίνει ανεκμετάλλευτη από εργασίες με μεγάλη ζήτηση υπολογισμού. Το FF και το QF επιτυγχάνουν συνολικά το καλύτερο χρόνο εκτέλεσης. Θυμηθείτε ότι το FF επιλέγει την πρώτη αχρησιμοποίητη χωρητικότητα ενώ το QF επιλέγει αυτή που έχει ως αποτέλεσμα την ελάχιστη EFT . Σε πολλές

περιπτώσεις, ειδικά για έναν μικρό αριθμό ροών εργασίας, το FF και το QF επιλέγουν την ίδια αχρησιμοποίητη χωρητικότητα για να τοποθετήσουν μια εργασία που εξηγεί την παρόμοια απόδοσή τους. Μια ενδιαφέρουσα παρατήρηση είναι η κακή απόδοση της συσκευασίας BF αλγόριθμος. Αυτό προκύπτει από το γεγονός ότι το QF δεν είναι σε θέση να κάνει βραχυπρόθεσμες θυσίες για μακροπρόθεσμα οφέλη. Παρατηρώντας τα παραπάνω αναδρομικά μπορεί κάποιος να πει ότι για χαμηλό εύρος ζώνης με συνδυασμό ενός μεγάλου αριθμού μηχανών ο καλύτερος υποψήφιος αλγόριθμος συσκευασίας είναι ο FF ενώ για μεγάλο εύρος ζώνης και μέτριο αριθμό ροών εργασίας ο QF είναι η καλύτερη επιλογή.

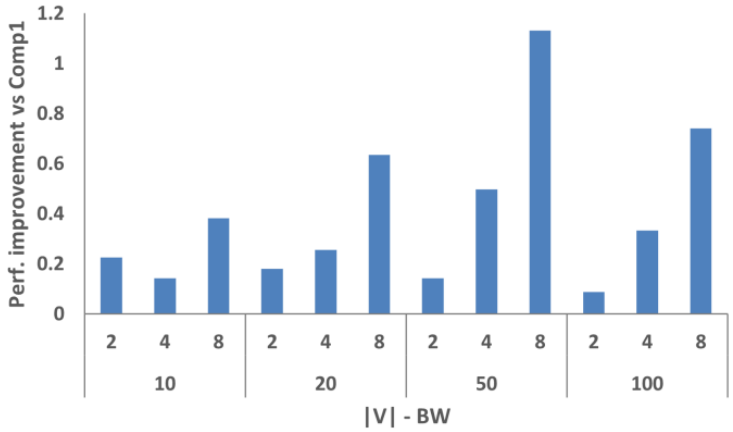
Έπειτα έγινε σύγκριση της προτεινόμενης μεθοδολογίας με δύο γνωστούς αλγορίθμους που αναλύθηκαν παραπάνω: Μια προσέγγιση σύνθεσης που προτείνεται στο [18] και μια δυναμική προσέγγιση που προτείνεται στο [14] (12 διαφορετικοί αλγόριθμοι). Εφαρμόστηκαν και οι 4 παραλλαγές από το [18] (εμφανίζεται ως $Comp1$) και τους 11 διαφορετικούς συνδυασμούς του [14] (εμφανίζονται ως $Comp2$) και αποθηκεύουν την καλύτερη απόδοση που πέτυχε. Για να μπορέσουν να συγκριθούν κατάλληλα, ορίστηκε η προθεσμία για κάθε ροή εργασίας ίση με +∞. Στα Γραφήματα 12-15 και στα Γραφήματα. 16-19 σχεδιάστηκε η ποσοστιαία βελτίωση του MWCP έναντι των $Comp1$ και $Comp2$, αντίστοιχα. Σε κάθε σχήμα καθορίζουμε τον συνολικό αριθμό των μηχανές σε 2, 4 και 8. ενώ το εύρος ζώνης κυμαίνεται από 10 έως 100 MBps.



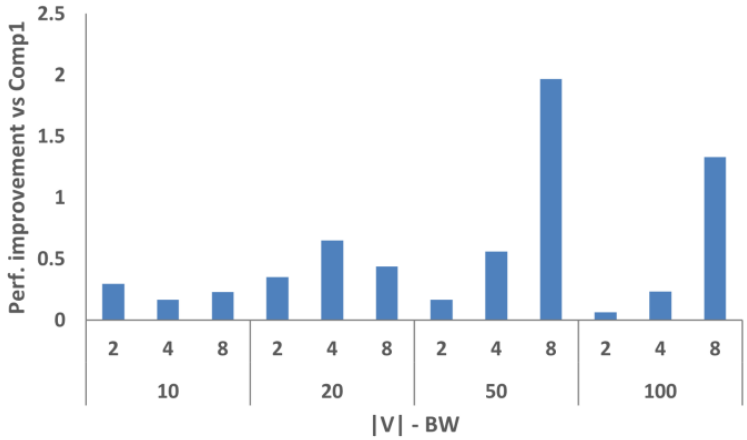
Γράφημα 5.12. Γράφημα το οποίο απεικονίζει τις ποσοστιαίες βελτιώσεις στον με τον συνολικό χρόνο εκτέλεσης του αλγορίθμου MWCP συγκριτικά με τους Comp1 για 7 ροές εργασιών.



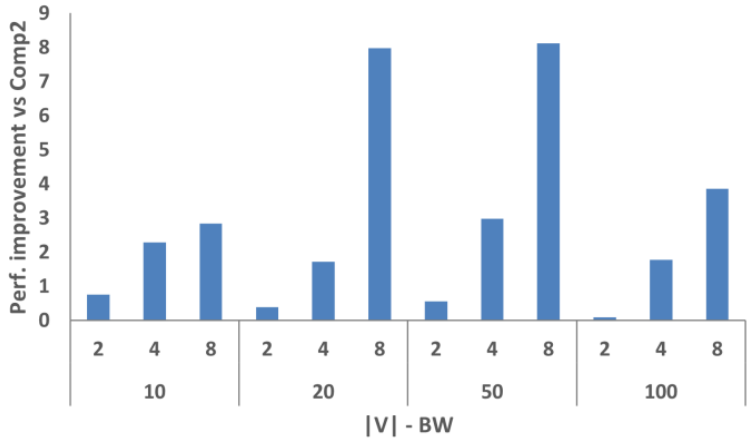
Γράφημα 5.13. Γράφημα το οποίο απεικονίζει τις ποσοστιαίες βελτιώσεις στον με τον συνολικό χρόνο εκτέλεσης του αλγορίθμου MWCP συγκριτικά με τους Comp1 για 14 ροές εργασιών.



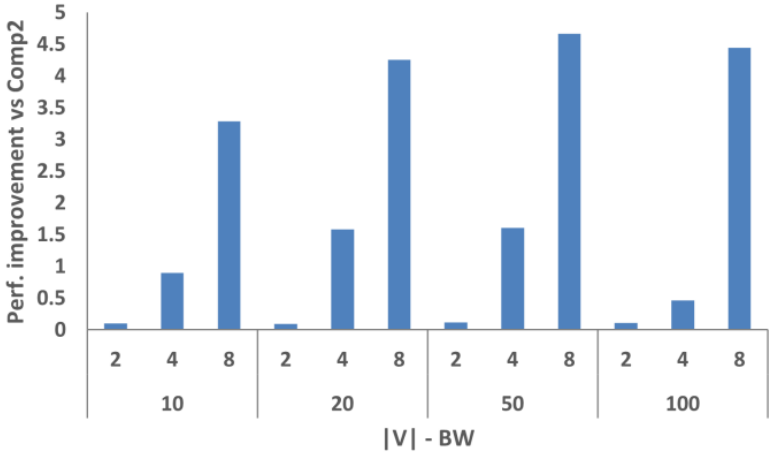
Γράφημα 5.14. Γράφημα το οποίο απεικονίζει τις ποσοστιαίες βελτιώσεις στον με τον συνολικό χρόνο εκτέλεσης του αλγορίθμου MWCP συγκριτικά με τους Comp1 για 28 ροές εργασιών.



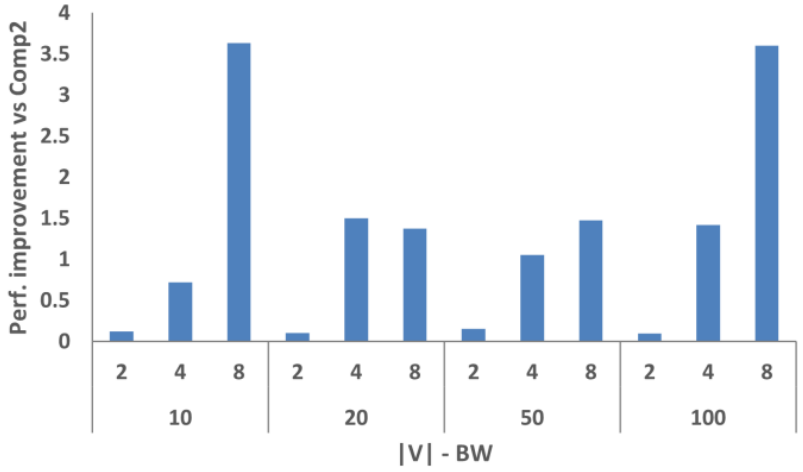
Γράφημα 5.15. Γράφημα το οποίο απεικονίζει τις ποσοστιαίες βελτιώσεις στον με τον συνολικό χρόνο εκτέλεσης του αλγορίθμου MWCP συγκριτικά με τους Comp1 για 56 ροές εργασιών.



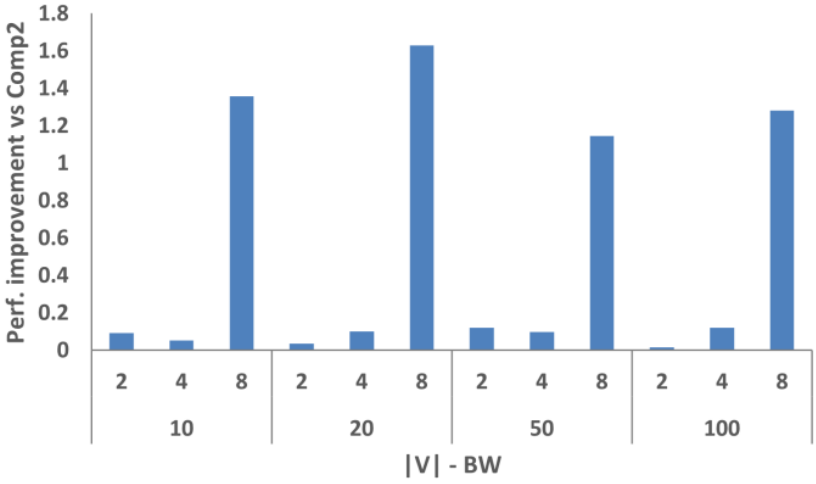
Γράφημα 5.16. Γράφημα το οποίο απεικονίζει τις ποσοστιαίες βελτιώσεις στον με τον συνολικό χρόνο εκτέλεσης του αλγορίθμου MWCP συγκριτικά με τους Comp2 για 7 ροές εργασιών.



Γράφημα 5.17. Γράφημα το οποίο απεικονίζει τις ποσοστιαίες βελτιώσεις στον με τον συνολικό χρόνο εκτέλεσης του αλγορίθμου MWCP συγκριτικά με τους Comp2 για 14 ροές εργασιών.



Γράφημα 5.18. Γράφημα το οποίο απεικονίζει τις ποσοστιαίες βελτιώσεις στον με τον συνολικό χρόνο εκτέλεσης του αλγορίθμου MWCP συγκριτικά με τους Comp2 για 28 ροές εργασιών.



Γράφημα 5.19. Γράφημα το οποίο απεικονίζει τις ποσοστιαίες βελτιώσεις στον με τον συνολικό χρόνο εκτέλεσης του αλγορίθμου MWCP συγκριτικά με τους Comp2 για 58 ροές εργασιών.

Πρώτον, μπορεί να γίνει η παρατήρηση ότι το MWCP έχει καλύτερη απόδοση και από τους $Comp1$ και τους $Comp2$ όσον αφορά τη συνολική απόδοση. Τα παραπάνω οφείλονται στην κατάλληλη τοποθέτηση των επιθυμητών εργασιών στους διαθέσιμους πόρους λόγο χρονικών τρυπών είτε σωστής προτεραιότητας των ροών όπως συνάπτονται από τον MWCP. Αυτό ισχύει για όλα τα πειραματικά σενάρια. Η απόδοση του MWCP αυξάνεται με τον αυξανόμενο αριθμό μηχανών, κάτι που είναι αρκετά ενθαρρυντικό. Για V = 8 MWCP επιτυγχάνει τη μέγιστη βελτίωση απόδοσης καθώς υπάρχει περισσότερος χώρος για αποφάσεις συσκευασίας. Ωστόσο, όπως ο αριθμός για τις ροές εργασίας αυξάνετε η απόδοση του MWCP είναι ελαφρώς υποβαθμισμένη. Αυτό είναι λογικό, καθώς υπάρχουν περισσότερες εργασίες, ανά απόφαση, στην κεντρική ουρά δίνοντας στους $Comp2$ ακόμη χώρο για βελτιστοποίηση. Ενώ η απόδοση του MWCP έναντι του $Comp1$ δεν επηρεάζεται από τον συνολικό αριθμό ροών εργασίας.

Εργαλεία προγραμματισμού και τεχνικές οπτικοποίησης

Στο παρόν κεφάλαιο θα γίνει μια επισκόπηση των εργαλείων που χρησιμοποιήθηκαν για την ανάπτυξη και των αλγορίθμων αλλά και μία μικρή σύντομη επισκόπηση ολόκληρου του περιβάλλοντος που στήθηκε για την εξασφάλιση μίας σωστής αποσφαλμάτωσης, λειτουργίας και ανάλυση των αλγορίθμων. Ενώ παράλληλα θα γίνει αναφορά σε τεχνικές οπτικοποίησης που μελετήθηκαν και ποιες από αυτές φάνηκαν πιο αποτελεσματικές.

Ανάπτυξη κώδικα για περισυλλογή δεδομένων 6.1

Όπως είναι γνωστό και από προηγούμενα κεφάλαια στην παρούσα εργασία έγινε χρήση 2 τύπων δεδομένων. Στην μία κατηγορία έγινε χρήση ενός εργαλείου το οποίο προσαρμόστηκε για να παράγει άκυκλους κατευθυνόμενους γράφους. Οι αλλαγές έγιναν στον κώδικα του εργαλείου έγιναν σε γλώσσα C [36], ενώ στο τέλος γράφτηκε κώδικας σε γλώσσα Python για να τρέξει επαναληπτικά μέσο του terminal το πρόγραμμα για να δημιουργήσει όλες τις περιπτώσεις των γράφων που χρειάστηκε να εξεταστούν. Ο βασικός βρόχος του συγκεκριμένου εργαλείου ήταν αυτής της μορφής:

```
for v in product(*values):
    param = dict(zip(keys, v))
    filename = "./data/generated_dags/{}_{}_{}_{}_{}.dot".format(param['n'], param['fat'],
                                                                 param['density'],
                                                                 param['regularity'],
                                                                 param['jump'])
    if os.path.isfile(filename):
        continue
    param = dict(zip(keys, v))
    os.system("./daggen/daggen -n {} --fat {} --density {} --regular {} --jump {} --minalpha {} --maxalpha {} --dot -o {}".format(
        param['n'], param['fat'], param['density'], param['regularity'], param['jump'], minalpha, maxalpha, filename))
```

Εικόνα 6.1. Στιγμιότυπο από κώδικα Python αναφορικά με το εργαλείο δημιουργίας γράφων διαφόρων ορισμάτων.

Πολύ περιληπτικά ο βρόχος τρέχει κάθε συνδυαστική περίπτωση που μπορεί να παραχθεί βάση των ορισμάτων που παίρνει το πρόγραμμα και αν δεν υπάρχει ήδη το ζητούμενο αρχείο τρέχει το κύριο πρόγραμμα που έχει αναπτυχθεί σε κώδικα C.

Το δεύτερο κομμάτι έχει να κάνει με την περισυλλογή των αρχείων .dot και των .json τα οποία δημιουργήθηκαν με την χρήση την βιβλιοθήκης της Python WfCommons. Κάθε τύπος αρχείου είχε διαφορετική μεταχείριση ενώ δημιουργήθηκαν τρόποι ανάλυσης αρχείων τύπου .dot, .json, csv, excel.

Αποσφαλμάτωση χρονοδιαγραμμάτων 6.2

Κάτι στο οποίο δόθηκε ιδιαίτερη σημασία ήταν η σωστή αποσφαλμάτωση των αποτελεσμάτων που παράχθηκαν από τους αλγορίθμους που είτε προτάθηκαν από την βιβλιογραφία είτε αναπτύχθηκαν στο πλαίσιο της μελέτης της παρούσας εργασίας. Τα εργαλεία που δημιουργήθηκαν για αυτόν τον σκοπό μπορούν να χωριστούν σε επίπεδα ξεκινώντας από το χαμηλότερο επίπεδο. Τα επίπεδα μαζί με τα εργαλεία που δημιουργήθηκαν σε αυτά ήταν τα εξής:

Αποσφαλμάτωση σε επίπεδο Εργασίας 6.2.α

Κάθε εργασία κατά την διάρκεια της δημιουργία της εντός του συστήματος εξετάζεται εάν έχει γονείς και παιδιά. Εάν και η εργασία δεν έχει ούτε γονείς ούτε παιδιά τότε σημαίνει ότι η εργασία δεν είναι συνδεδεμένη στον γράφο με αποτέλεσμα ολόκληρη η ροή εργασιών να θεωρείται λανθασμένη.

Ακόμα κάθε φορά που γίνεται ένταξη μίας εργασίας σε έναν υπολογιστικό πόρο τρέχει μία μέθοδος η οποία εξετάζει τα εξής:

1. Η εργασία δεν πρέπει να έχει λάβει μέχρι εκείνη την χρονική στιγμή την κατάσταση Scheduled.
2. Η εργασία εξετάζει ότι έχει ενημερωθεί από όλες τις εργασίες γονείς ότι έχουν τελειώσει και έχουν λάβει την κατάσταση Scheduled για να μπορέσει να προγραμματιστεί η ίδια.
3. Τέλος η εργασία ενημερώνει με την σειρά της όλες τις εργασίες παιδιά ότι έχει λάβει τέλος και ότι έχει μπει σε κατάσταση Scheduled.

Αποσφαλμάτωση σε επίπεδο Υπολογιστικού Πόρου 6.2.β

Όταν ένας υπολογιστικός πόρος όταν λαμβάνει μία εργασία στο σύστημα του είναι υποχρεωμένος να κάνει τα εξής βήματα:

1. Εξετάζει ότι δεν έχει λάβει να προγραμματίσει την ίδια εργασία δύο φορές.
2. Γίνεται έλεγχος ότι η εισερχόμενη εργασία δεν μπαίνει σε θέση στην οποία υπάρχει σφάλμα επικάλυψης.
3. Ανατρέχει εντός της εσωτερικής ουράς και κοιτάει για τυχόν χρονικά κενά και τα δημιουργεί εάν δεν υπάρχουν ήδη. Το τελευταίο αυτό στάδιο είναι αρκετά χρήσιμο καθώς υπάρχουν περιπτώσεις που εισέρχονται εργασίες εντός τρυπών στις οποίες το σύστημα θα πρέπει να αφαιρέσει το παλιό χρονικό κενό από το σύστημα αλλά να δημιουργήσει κατάλληλα τυχόν καινούργια που μπορεί να προκύψουν.

Αποσφαλμάτωση σε επίπεδο ροής εργασιών 6.2.γ

Για να μπορέσει να θεωρηθεί ότι μία ροή έχει προγραμματιστεί σωστά θα πρέπει να περάσει από μία σειρά ελέγχων των οποίων εάν σε έστω έναν από αυτούς αποτύχει η ροή θεωρείται ότι έχει προγραμματιστεί λανθασμένα και θα πρέπει να απορριφθεί και ο αλγόριθμος ή τρόπος που βοήθησε στο αποτέλεσμα αυτό. Οι έλεγχοι είναι οι εξής:

1. Με την δημιουργία μίας ροής εργασιών γίνεται έλεγχος για την κύκλους ανάμεσα στις εργασίες. Αναδρομικά ελέγχονται η εργασίες ενώ παράλληλα υπολογίζεται το upward rank τους, downward rank και το critical path.
2. Μόλις θεωρηθεί ότι μία ροή εργασιών έχει προγραμματιστεί πλήρως ελέγχεται εάν όλες οι εργασίες εντός αυτής έχουν κατάσταση Scheduled. Ακόμα ελέγχεται εάν υπάρχει εργασία (πέρα των ψεύτικων εργασιών που εισάχθηκαν από τους αλγορίθμους) για κόστος η τελική εκτέλεση ίσα με 0.
3. Τέλος ελέγχονται εάν οι εργασίες που προγραμματίστηκαν δεν παραβιάζουν κάποια σειρά προτεραιότητας.

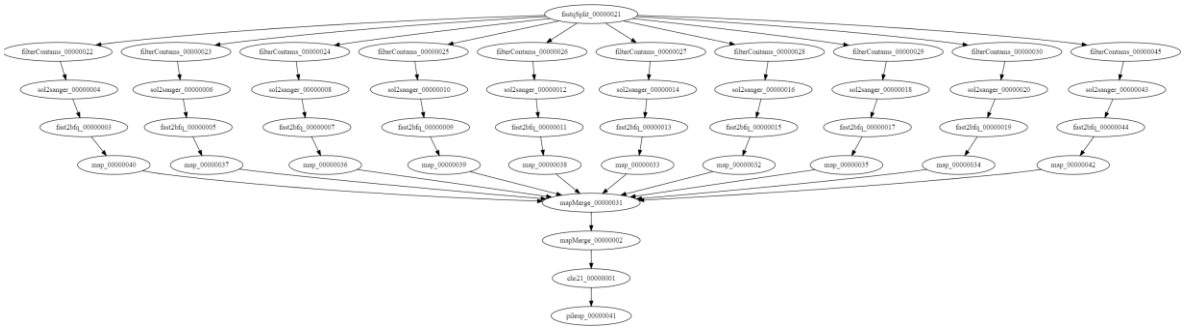
Αποσφαλμάτωση σε επίπεδο χρονοδιαγράμματος 6.2.δ

Τέλος υπάρχει ένας τελευταίος έλεγχος ο οποίος συμβαίνει σε επίπεδο χρονοδιαγράμματος. Στο τέλος του προγραμματισμού ελέγχονται όλοι οι υπολογιστικοί πόροι ξανά για τυχόν επικαλύψεις εντός της ουράς που έχουν. Κάτι το οποίο μπορεί να συμβεί και σε επίπεδο ροής εργασιών εάν υπάρχει ανάγκη για έλεγχο κάθε ροής ξεχωριστά.

Οπτικοποίηση και αποθήκευση αποτελεσμάτων 6.3

Κατά την διάρκεια της μελέτης δημιουργήθηκαν πολλοί τρόποι αποθήκευσης εργασιών σε τύπους αρχείων όπως csv, txt, json, dot, excel. Στο τέλος δύο τύποι αρχείων υπερίσχυσαν το json, το dot και το excel χωρίς αυτό να αποκλείει την δυνατότητα υποστήριξης των υπολοίπων. Οι προαναφερθέντες τύποι αρχείων ήταν ιδιαίτερα χρήσιμοι όταν χρειάστηκε να γίνει οπτικοποιηση των δεδομένων και των στατιστικών που μελετήθηκαν καθόλη την διάρκεια της εργασίας.

Μία πρώτη αρχικοποίηση έγινε με την βοήθεια του WfCommons που και των Custom generated δεδομένων που παρουσιάστηκαν σε προηγούμενα κεφάλαια καθώς τα αρχεία .dot μπορούν με την βοήθεια του εργαλείου Graphviz [37] να γίνει μία πρώτη απλή οπτικοποίηση των αρχείων για να μπορέσει να γίνει μία περισκόπηση αναφορικά με την μορφολογία των ροών.



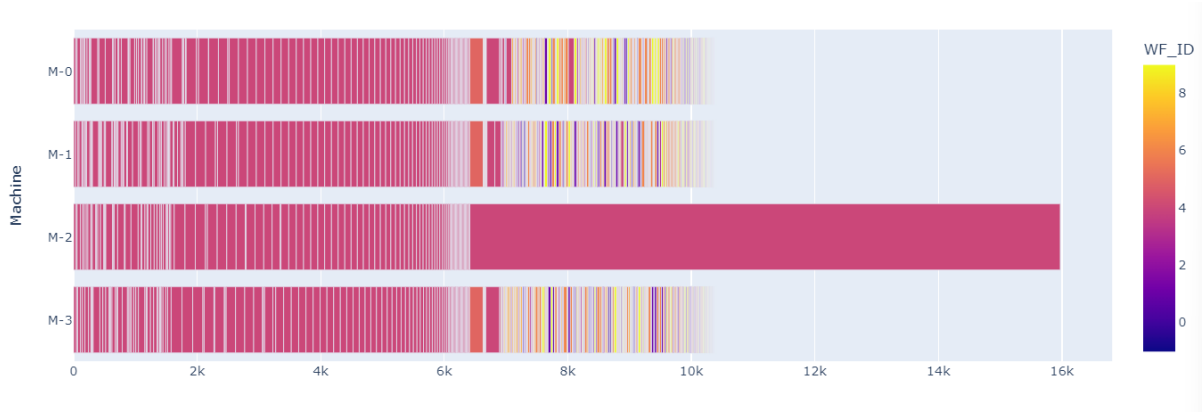
Γράφημα 6.2. Ροή εργασιών τύπου blast που οπτικοποιήθηκε από το εργαλείο Graphviz.

Επειδή όμως ο παραπάνω τρόπος οπτικοποίησης ήταν περιορισμένος για τις απαιτήσεις τις παρούσας εργασίας αναπτύχθηκε ένα εργαλείο στην προγραμματιστική γλώσσα JavaScript [38] με την βοήθεια της βιβλιοθήκης GoJS [39]. Αυτό έγινε για την καλύτερη μοντελοποίηση των ροών εργασιών και για να μπορεί να γίνει μία πιο εμπεριστατωμένη ανάλυση των ροών αναφορικά με την μορφολογία και την δομή τους.

Γράφημα 6.3. Ροή εργασιών τύπου epigenomics που παράχθηκε με την βοήθεια της βιβλιοθήκης GoJS.

Το Γράφημα 6.2 παράχθηκε με την χρήση της βιβλιοθήκης GoJS και επιτρέπει να ελέγχουμε αναλυτικά το που βρίσκεται κάθε εργασία πάνω στην ροή κάθε αλληλεξάρτηση μεταξύ των εργασιών ενώ ο κύριος λόγος χρήσης της παραπάνω οπτικοποίησης είναι για να γίνει πιο εύκολα αντιληπτή η μορφολογία της ροής καθώς και αλληλεξαρτήσεις μεταξύ των εργασιών.

Πολλές τεχνικές οπτικοποίησης επίσης ενσωματώθηκαν στο κύριο πρόγραμμα για να βοηθήσει στην καλύτερη κατανόηση της ροής του προγράμματος. Μία συγκεκριμένη τεχνική που γράφτηκε σε κώδικα Python και θεωρήθηκε ιδιαίτερα χρήσιμη είχε να κάνει με την οπτικοποίηση ολόκληρου του χρονοδιαγράμματος.



Γράφημα 6.4. Χρονοδιάγραμμα πολλαπλών ροών εργασιών προγραμματίστηκε από τον αλγόριθμο C1 και οπτικοποιήθηκε με την βοήθεια της βιβλιοθήκης plotly.

Το παραπάνω διάγραμμα οπτικοποιήθηκε με την βοήθεια της βιβλιοθήκης plotly και αποτέλεσε κομβικό σημείο ακόμα και για την ανάπτυξη κάποιων αλγορίθμων της παρούσας εργασίας. Το συγκεκριμένο εργαλείο επιτρέπει στον χρήστη να δει αναλυτικά για κάθε εργασία τα εξής:

- Σε ποια ροή εργασιών ανήκει
- Σε ποιον υπολογιστικό πόρο προγραμματίστηκε
- Το ξεκίνημα εκτέλεσης της
- Το πέρας εκτέλεσης της
- Τον μοναδικό αναγνωριστικό αριθμό (id)
- Τον αργότερο γονέα της εργασίας.

Εργασίες που ανήκουν στην ίδια ροή εργασιών έχουν ίδιο χρώμα. Αυτό συνέβη για να φανεί πιο εύκολα ο παραλληλισμός των ροών εργασιών μεταξύ των υπολογιστικών πόρων. Τέλος ο χρήστης μπορεί να ζουμάρει σε συγκεκριμένο σημείο πάνω στο χρονοδιάγραμμα για να μπορέσει να δει πιο αναλυτικά την εργασία που επιθυμεί.

Πολλές οπτικοποιήσεις επίσης έλαβαν χώρο μέσο του προγράμματος Microsoft Excel, ενώ ακόμα έγινε και εκτενής χρήση για ποικίλες οπτικοποιήσεις στατιστικών αποτελεσμάτων και συγκρίσεων η βιβλιοθήκη matplotlib.

Συμπεράσματα

Στην παρούσα εργασία εξετάστηκαν το πρόβλημα του προγραμματισμού μονής ροής εργασιών καθώς και το πρόβλημα πολλαπλών ροών εργασιών σε ένα ετερογενές υπολογιστικό περιβάλλον ενώ γίνεται και προσπάθεια προσομοίωσης και του υπολογιστικού νέφους.

Ενώ για την κατάλληλη ανάλυση των αλγορίθμων έγινε εκτενής έρευνα για να βρεθούν τα κατάλληλα δεδομένα τα οποία χωρίστηκαν σε δύο κατηγορίες: Ρεαλιστικά, Custom generated. Ακόμα κάθε κατηγορία δεδομένων είχε διαφορετική εφαρμογή εντός της εργασίας, βοηθώντας να προσομοιωθούν διαφορετικά σενάρια.

Επιπλέον αναφορικά με την μονή ροή εργασιών αναλύθηκαν διάφοροι αλγόριθμοι και ελέγχθηκαν ως προς την απόδοση τους αναφορικά με την μείωση του συνολικού χρόνου εκτέλεσης που χρειάζεται για να προγραμματίσουν μία μονή ροή εργασιών. Αρχικά ο αλγόριθμος του HEFT ήταν αυτός που ξεχώρισε λόγω της σχετικής χαμηλής πολυπλοκότητας και του αρκετά ικανοποιητικού αποτελέσματος που ήταν σε θέση να παράγει. Ενώ ο αλγόριθμος του CPOP παρατηρήθηκε ότι είχε εφαρμογές σε συγκεκριμένες περιπτώσεις στις οποίες όπως: α) Η ροή είχε μεγάλα επικοινωνιακά κόστη β) Η ροή αποτελούταν από πολλά επίπεδα και ήταν «ψηλή». Από εκεί ήταν αρκετά εμφανές ότι η χρήση του Ruin and Recreate βελτίωνε τα πιθανά χρονοδιαγράμματα των προαναφερθέντων έως και 35%. Επίσης αναφορικά με την χρονοδρομολόγηση πολλαπλών ροών εργασιών εξετάστηκαν αλγόριθμοι από την βιβλιογραφία οι οποίοι είχαν να εξέταζαν το πρόβλημα έχοντας ως γνώμονα ποικίλους στόχους. Οι στόχοι αυτοί διαφοροποιόντουσαν ανάλογα με τον αλγόριθμο παραδείγματος χάρη οι αλγόριθμοι που προτάθηκαν από το [10] είχαν ως κύριο στόχο να υπάρχει ισότιμη κατανομή μεταξύ των ροών εργασιών, ενώ στο [9] οι αρθρογράφοι προτάσσουν δύο διαφορετικούς τρόπους σύνθεσης εκ των οποίων παράγονται 4 διαφορετικές παραλλαγές χρονοδρομολόγησης των εργασιών. Ενώ ο στόχος της λύσης του προβλήματος διαφέρει ανάλογα τον αλγόριθμο που εφαρμόζουν.

Έχοντας ως έμπνευση κάποιους από τους αλγορίθμους που προτάθηκαν νέες λύσεις όπως ο MWCP ο οποίος συνδυάζει το πως θα χρονοδρομολογήσει της ροές το οποίο επιτυγχάνει συνολικά έναν πιο γρήγορο τρόπο εκτέλεσης όλων των ροών μαζί ενώ παράλληλα καταφέρνει να καλύψει τα χρονικά κενά τον υπολογιστικών πόρων με αποτέλεσμα να ανεβαίνει ο μέσος όρος του utilization των υπολογιστικών πόρων. Επίσης σε περιπτώσεις στις οποίες οι ροές είναι μικρές σε αριθμό και υπάρχουν εργασίες οι οποίες καταλαμβάνουν μεγάλα ποσοστά από τον συνολικό χρόνο εκτέλεσης, τότε η χρήση του CTF ήταν μία ικανοποιητική λύση καθώς λαμβάνει υπόψιν το συγκεκριμένο edge case. Ακόμα παρατηρήθηκε πότε μία bin packing μέθοδος ήταν αποτελεσματική, ενώ υπήρξαν περιπτώσεις στις οποίες η χρήση τους άλλαζε δραματικά το πιθανό χρονοδιάγραμμα.

Τέλος γίνεται μία γρήγορη επισκόπηση των εργαλείων οπτικοποίησης και αποσφαλμάτωσης που ανακτήθηκαν καθόλη την διάρκεια της έρευνας. Έχοντας σαν επίκεντρο κάποιες συγκεκριμένες μεθόδους και εργαλεία που χρησιμοποιήθηκαν.

Βάση των αλγορίθμων που προτάθηκαν και μελετήθηκαν η ανάπτυξη ενός πραγματικού περιβάλλοντος για την ανάπτυξή αυτών ίσως αποτελεί πρωταρχικός ρόλο για μετέπειτα έρευνα που μπορεί να διεξαχθεί. Αυτό μπορεί να επιτευχθεί με προσομοιωμένα υπολογιστικά περιβάλλοντα όπως Virtual Machines η Dockerized Imaged [40]. Αυτό γιατί θα η δυνατότητες προσομοίωσης διαφορετικών σεναρίων θα είναι πρακτικά απύθμενες. Σε συνδυασμό ότι η εναλλαγή κατανεμημένου περιβάλλοντος θα μπορούσε να απλοποιηθεί με ευκολία. Ακόμα αναφορικά με τον τρόπο επίλυσης του προβλήματος, βλέποντας την αποδοτικότητα του μεταευρετικού αλγορίθμου Ruin and Recreate η ανάπτυξη ενός γενετικού αλγορίθμου για πολλαπλές ροές εργασιών θα μπορούσε να βελτιώσει αποτελέσματα άπληστων αλγορίθμων με μεγάλη αποτελεσματικότητα. Ο λόγος που γίνεται πρόταση για έναν γενετικό αλγόριθμο έχει να κάνει με το γεγονός ότι μπορεί να προγραμματιστεί με τρόπο που θα αποφεύγει τα τοπικά ελάχιστα και μέγιστα στα οποία πέφτει έναν άπληστος αλγόριθμος φτάνοντας έτσι πιο κοντά στην optimal λύση.

Βιβλιογραφία

[1] Nicky Opitz, Tobias F. Langkau, Nils H. Schmidt, Lutz M. Kolbe, «Technology Acceptance of Cloud Computing: Empirical Evidence from German IT Departments,» σε *45th Hawaii International Conference on System Sciences*, Maui, HI, USA, 2012.

[2] Dimpi Rani, Rajiv Kumar Ranjan , «A Comparative Study of SaaS, PaaS and IaaS in Cloud,» τόμ. 4, αρ. 6, 2014.

[3] C.-J. H. Huang, «An adaptive resource management scheme in cloud computing,» *Engineering Applications of Artificial Intelligence,* 2013.

[4] Lovejit Singh, Sarbjeet Singh, «A Survey of Workflow Scheduling Algorithms and,» *International Journal of Computer Applications (0975 – 8887),* τόμ. 74, αρ. 15, pp. 21-28, 2013.

[5] M. Chand, «https://www.c-sharpcorner.com,» C#Corner, 15 July 2021. [Ηλεκτρονικό]. Available: https://www.c-sharpcorner.com/article/top-10-cloud-service-providers/.

[6] Kavishi, «https://docs.microsoft.com/en-us/power-platform/admin/pay-as-you-go-overview,» 07 07 2022. [Ηλεκτρονικό]. Available: https://docs.microsoft.com/en-us/power-platform/admin/pay-as-you-go-overview.

[7] Haluk Topcuoglu, Salim Hariri, Min-You Wu, «Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing,» *IEEE Transactions on parallel and distributed systems,* τόμ. 13, αρ. 3, pp. 260-274, 2002.

[8] Gerhard Schrimpf, Johannes Schneider, HermannStamm-Wilbrandt, GunterDueck, «Record Breaking Optimization Results Using the Ruin and Recreate Principle,» *Journal of Computational Physics,* τόμ. 159, αρ. 2, pp. 139-171, 2000.

[9] Henan Zhao, Rizos Sakellariou, «Scheduling Multiple DAGs onto Heterogeneous Systems,» *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium,* τόμ. 14, 2006.

[10] Georgios L.Stavrinides, Helen D.Karatza, «Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques,» *Simulation Modelling Practice and Theory,* τόμ. 19, αρ. 1, pp. 540-552, 2011.

[11] Nikos Pappas, Panagiotis Oikonomou, Nikos Tziritas, Kostas Kolomvatsos, «Bin Packing Heuristics for the Multiple Workflow Scheduling,» σε *25th Pan-Hellenic Conference on Informatics (PCI 2021)*, Volos, Greece, 2021.

[12] S. Carpenter, «https://medium.com,» 10 February 2022. [Ηλεκτρονικό]. Available: https://medium.com/codex/why-is-python-so-slow-f434075a372.

[13] R. Mayani, «https://pegasus.isi.edu/,» [Ηλεκτρονικό].

[14] WfCommons, «https://wfcommons.org/,» [Ηλεκτρονικό].

[15] Pegasus, «https://pegasus.isi.edu,» [Ηλεκτρονικό]. Available: https://pegasus.isi.edu/about/ .

[16] XML, «https://www.w3.org/XML/,» [Ηλεκτρονικό].

[17] Python, «https://www.python.org/,» A interpreted programming language. [Ηλεκτρονικό].

[18] 1000Genome, «https://www.internationalgenome.org/,» [Ηλεκτρονικό].

[19] BLAST, «https://en.wikipedia.org/wiki/BLAST,» [Ηλεκτρονικό].

[20] NASA, IPAC, «https://confluence.pegasus.isi.edu/display/pegasus,» [Ηλεκτρονικό].

[21] USC, «https://confluence.pegasus.isi.edu/display/pegasus,» [Ηλεκτρονικό].

[22] NGS, «https://pegasus.isi.edu/portfolio/soykb/,» [Ηλεκτρονικό].

[23] Seismology, «https://github.com/pegasus-isi/seismology-workflow,» [Ηλεκτρονικό].

[24] N. Pappas, «https://github.com,» 18 September 2021. [Ηλεκτρονικό]. Available: https://github.com/wfcommons/wfcommons/issues/23.

[25] Yingchun Yuan, Xiaoping Li, Qian Wang, Xia Zhu, «Deadline division-based heuristic for cost optimization in workflow scheduling,» *Information Sciences,* τόμ. 179, αρ. 15, pp. 2562-2575, 2009.

[26] Luiz Fernando Bittencourt, Edmundo Roberto Mauro Madeira , «HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds,» *Journal of Internet Services and Applications,* p. 207–227, 2011.

[27] Luiz F. Bittencourt, Rizos Sakellariou, Edmundo R. M. Madeira, «DAG Scheduling Using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm,» σε *18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, Pisa, Italy, 2010.

[28] Henan Zhao, Rizos Sakellariou , «An Experimental Investigation into the Rank Function of the Heterogeneous Earliest Finish Time Scheduling Algorithm,» σε *Euro-Par 2003 Parallel Processing. Lecture Notes in Computer Science*, 2004.

[29] N. Rincon-Garcia, B.J. Waterson, T.J. Cherrett, «A hybrid metaheuristic for the time-dependent vehicle routing problem with hard time windows,» *International Journal of Industrial Engineering Computations,* τόμ. 8, αρ. 1, pp. 141-160, 2017.

[30] Mauro Dell'Amico, Manuel Iori, Stefano Novellani, Thomas Stützle, «A Destroy and Repair Algorithm for the Bike sharing Rebalancing Problem,» *Computers & Operations Research,* τόμ. 71, pp. 149-162, 2016.

[31] Stefan Ropke, David Pisinger, «An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows,» *Transportation Science,* τόμ. 40, αρ. 4, pp. 455-472, 2006.

[32] David Pisinger, Stefan Ropke, «A general heuristic for vehicle routing problems,» *Computers & Operations Research,* τόμ. 34, αρ. 8, pp. 2403-2435, 2007.

[33] Rubén Ruiz, Thomas Stützleb, «An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives,» *European Journal of Operational Research,* τόμ. 187, αρ. 3, pp. 1143-1159, 2008.

[34] Tableau, «https://www.tableau.com/,» [Ηλεκτρονικό]. Available: https://www.tableau.com/.

[35] Matplotlib, «https://matplotlib.org/,» [Ηλεκτρονικό]. Available: https://matplotlib.org/.

[36] Wikipedia, «https://en.wikipedia.org,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/C_(programming_language).

[37] Graphviz, «https://graphviz.org/,» [Ηλεκτρονικό]. Available: https://graphviz.org/.

[38] JavaScript, «https://www.javascript.com/,» [Ηλεκτρονικό]. Available: https://www.javascript.com/.

[39] GoJS, «https://gojs.net/,» [Ηλεκτρονικό]. Available: https://gojs.net/latest/index.html.

[40] D. Co., «https://en.wikipedia.org,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Docker_(software).