ΑΠΟΔΟΤΙΚΗ ΚΑΤΑΝΟΜΗ ΡΟΩΝ ΕΡΓΑΣΙΑΣ ΥΠΗΡΕΣΙΩΝ ΔΙΑΔΙΚΤΥΟΥ

Η
ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

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

από τον

Σταμκόπουλο Κωνσταντίνο

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΟ ΛΟΓΙΣΜΙΚΟ

Ιούλιος 2006

ΑΠΟΔΟΤΙΚΗ ΚΑΤΑΝΟΜΗ ΡΟΩΝ ΕΡΓΑΣΙΑΣ ΥΠΗΡΕΣΙΩΝ ΔΙΑ[illegible]ΕΛΕΓΧΟΥ

Η

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

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

από τον

Στεφικόπουλο Κωνσταντίνο

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΟ ΛΟΓΙΣΜΙΚΟ

Ιούλιος 2006

ΑΦΙΕΡΩΣΗ

Αφιερώνω την εργασία αυτή στους γονείς μου που με βοήθησαν να ολοκληρώσω τις σπουδές μου.

ΕΥΧΑΡΙΣΤΙΕΣ

Ευχαριστώ θερμά τους καθηγητές μου Ευαγγελία Πιτουρά και Παναγιώτη Βασιλειάδη για την βοήθειά τους και την συνεργασία που είχαμε στην συγγραφή αυτής της εργασίας. Οι συμβουλές τους και οι γνώσεις που μου μετέδωσαν είναι πολύτιμες για τη συνέχεια.

ΠΕΡΙΕΧΟΜΕΝΑ

ΑΦΙΕΡΩΣΗ II
ΕΥΧΑΡΙΣΤΙΕΣ III
ΠΕΡΙΕΧΟΜΕΝΑ IV
ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ VI
ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ VII
ΠΕΡΙΛΗΨΗ IX
EXTENDED ABSTRACT IN ENGLISH X
ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ 11
ΚΕΦΑΛΑΙΟ 2. ΣΧΕΤΙΚΕΣ ΕΡΓΑΣΙΕΣ 16
2.1. Υπηρεσίες Διαδικτύου 16
2.2. Ανακάλυψη Υπηρεσιών Διαδικτύου με Κατανεμημένο Τρόπο 17
2.3. Διαχείριση Σύνθετων Υπηρεσιών με Βάση την Ποιότητα της Υπηρεσίας 24
2.4. Διαχείριση Δυναμικών Σύνθετων Υπηρεσιών σε Δυναμικά Δίκτυα 27
2.5. Εκτέλεση Σύνθετων Υπηρεσιών με Κατανεμημένο Τρόπο 28
2.6. Βελτιστοποίηση Ερωτήσεων για Σύνθετες Υπηρεσίες Διαδικτύου 30
2.7. Τοποθέτηση Αντικειμένων σε Απομακρυσμένη Μνήμη 31
2.8. Ανακεφαλαίωση 33
ΚΕΦΑΛΑΙΟ 3. ΓΕΝΙΚΟΣ ΟΡΙΣΜΟΣ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ 37
3.1. Ορισμός του Workflow 37
3.2. Παράδειγμα 39
3.3. Διαστασιοποίηση του Προβλήματος 42
3.3.1. Παράμετροι για το Workflow 45
3.3.2. Παράμετροι για το Δίκτυο Εξυπηρετητών 45
3.3.3. Χαρακτηριστικά Αξιολόγησης Μιας Αντιστοίχισης 46
3.4. Μοντελοποίηση του Προβλήματος 46
3.5. Ιδιότητες Μιας Καλής Λύσης του Προβλήματος 47
3.6. Συμβολισμοί και Τύποι 49
ΚΕΦΑΛΑΙΟ 4. ΑΛΓΟΡΙΘΜΟΙ 53
4.1. Εξαντλητικός Αλγόριθμος 53
4.2. Αλγόριθμος Γραμμή–Γραμμή 55
4.2.1. Παραλλαγές αλγορίθμου Γραμμή–Γραμμή 62
4.3. Αλγόριθμοι Γραμμή–Δίαυλος 62
4.4. Αλγόριθμοι Τυχαίος Γράφος–Δίαυλος 77
ΚΕΦΑΛΑΙΟ 5. ΠΕΙΡΑΜΑΤΙΚΕΣ ΜΕΤΡΗΣΕΙΣ 79
5.1. Τιμές των Παραμέτρων στα Πειράματα 79
5.2. Πειράματα για τον Αλγόριθμο Γραμμή–Γραμμή 83
5.3. Πειράματα για τους Αλγορίθμους Γραμμή–Δίαυλος 90

5.4. Πειράματα για τους Αλγορίθμους Τυχαίος Γράφος–Δίαυλος 99
5.5. Ανακεφαλαίωση 105
ΚΕΦΑΛΑΙΟ 6. ΕΠΙΛΟΓΟΣ 107
6.1. Ανακεφαλαίωση 107
6.2. Μελλοντικές Επεκτάσεις 108
ΑΝΑΦΟΡΕΣ 109
ΣΥΝΤΟΜΟ ΒΙΟΓΡΑΦΙΚΟ 111

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

ΠΙΝΑΚΑΣ.....ΣΕΛ.

ΠΙΝΑΚΑΣ 3.1 ΛΕΙΤΟΥΡΓΙΚΟΤΗΤΑ ΤΩΝ ΛΕΙΤΟΥΡΓΙΩΝ ΤΟΥ WORKFLOW ΤΟΥ ΣΧΗΜΑΤΟΣ 3.2 41

ΠΙΝΑΚΑΣ 3.2 ΠΕΡΙΓΡΑΦΗ ΤΩΝ ΜΗΝΥΜΑΤΩΝ ΤΟΥ WORKFLOW ΤΟΥ ΣΧΗΜΑΤΟΣ 3.2 ... 42

ΠΙΝΑΚΑΣ 3.3 ΠΕΡΙΠΤΩΣΕΙΣ ΠΟΥ ΜΕΛΕΤΑΜΕ ΓΙΑ ΤΟΝ ΌΓΚΟ ΕΡΓΑΣΙΑΣ ΤΩΝ ΛΕΙΤΟΥΡΓΙΩΝ ΤΟΥ WORKFLOW. 43

ΠΙΝΑΚΑΣ 3.4 ΠΕΡΙΠΤΩΣΕΙΣ ΠΟΥ ΜΕΛΕΤΑΜΕ ΓΙΑ ΤΟ ΜΕΓΕΘΟΣ ΤΩΝ ΜΗΝΥΜΑΤΩΝ ΤΟΥ WORKFLOW..... 43

ΠΙΝΑΚΑΣ 3.5 ΠΙΘΑΝΕΣ ΠΕΡΙΠΤΩΣΕΙΣ ΓΙΑ ΤΗΝ ΥΠΟΛΟΓΙΣΤΙΚΗ ΙΣΧΥ ΤΩΝ ΕΞΥΠΗΡΕΤΗΤΩΝ..... 44

ΠΙΝΑΚΑΣ 3.6 ΣΥΜΒΟΛΙΣΜΟΙ – ΤΕΛΕΣΤΕΣ ΤΩΝ ΠΑΡΑΜΕΤΡΩΝ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ..... 50

ΠΙΝΑΚΑΣ 5.1 ΤΑ ΜΕΓΕΘΗ ΜΗΝΥΜΑΤΩΝ ΤΩΝ ΠΕΙΡΑΜΑΤΩΝ ΤΗΣ ΚΑΤΗΓΟΡΙΑΣ Α. 81

ΠΙΝΑΚΑΣ 5.2 ΤΑ ΜΕΓΕΘΗ ΜΗΝΥΜΑΤΩΝ ΤΩΝ ΠΕΙΡΑΜΑΤΩΝ ΤΗΣ ΚΑΤΗΓΟΡΙΑΣ Β..... 82

ΠΙΝΑΚΑΣ 5.3 ΤΑ ΜΕΓΕΘΗ ΤΩΝ ΜΗΝΥΜΑΤΩΝ ΤΟΥ WORKFLOW ΤΟΥ ΣΧΗΜΑΤΟΣ 3.4.. 96

ΠΙΝΑΚΑΣ 5.4 ΟΙ ΚΥΚΛΟΙ ΤΩΝ ΛΕΙΤΟΥΡΓΙΩΝ ΤΟΥ WORKFLOW ΤΟΥ ΣΧΗΜΑΤΟΣ 3.4.... 96

ΠΙΝΑΚΑΣ 5.5 ΤΑ ΜΕΓΕΘΗ ΤΩΝ ΜΗΝΥΜΑΤΩΝ ΤΟΥ WORKFLOW ΤΟΥ ΣΧΗΜΑΤΟΣ 3.3. 100

ΠΙΝΑΚΑΣ 5.6 ΟΙ ΚΥΚΛΟΙ ΤΩΝ ΛΕΙΤΟΥΡΓΙΩΝ ΤΟΥ WORKFLOW ΤΟΥ ΣΧΗΜΑΤΟΣ 3.3.. 100

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

ΣΧΗΜΑ ..... ΣΕΛ.

ΣΧΗΜΑ 3.1 ΠΑΡΑΔΕΙΓΜΑ ΕΝΟΣ WORKFLOW. Ο ΤΥΠΟΣ ΤΗΣ ΔΙΑΚΛΑΔΩΣΗ ΜΠΟΡΕΙ ΝΑ ΕΙΝΑΙ ΈΝΑΣ ΑΠΟ ΤΟΥΣ ΤΡΕΙΣ ΠΟΥ ΑΠΕΙΚΟΝΙΖΟΝΤΑΙ. 38

ΣΧΗΜΑ 3.2 ΠΑΡΑΔΕΙΓΜΑ WORKFLOW ΜΕ ΔΙΑΚΛΑΔΩΣΕΙΣ. 40

ΣΧΗΜΑ 3.3 ΠΑΡΑΔΕΙΓΜΑ WORKFLOW ΧΩΡΙΣ ΔΙΑΚΛΑΔΩΣΕΙΣ. 40

ΣΧΗΜΑ 3.4 ΤΟ ΣΥΣΤΗΜΑ ΟΜΑΔΟΠΟΙΗΣΗΣ ΔΕΧΕΤΑΙ ΩΣ ΕΙΣΟΔΟ ΈΝΑ WORKFLOW, ΈΝΑ ΔΙΚΤΥΟ ΕΞΥΠΗΡΕΤΗΤΩΝ ΚΑΙ ΈΝΑ ΣΥΝΟΛΟ ΠΕΡΙΟΡΙΣΜΩΝ, ΧΡΗΣΙΜΟΠΟΙΕΙ ΈΝΑΝ ΑΠΟ ΤΟΥΣ ΑΛΓΟΡΙΘΜΟΥΣ ΚΑΙ ΤΗ ΣΥΝΑΡΤΗΣΗ ΚΟΣΤΟΥΣ ΚΑΙ ΔΙΝΕΙ ΈΞΟΔΟ ΜΙΑ ΑΝΤΙΣΤΟΙΧΙΣΗ ΤΩΝ ΛΕΙΤΟΥΡΓΙΩΝ ΤΟΥ WORKFLOW ΣΤΟΥΣ ΕΞΥΠΗΡΕΤΗΤΕΣ. 47

ΣΧΗΜΑ 4.1 Η ΤΟΠΟΛΟΓΙΑ ΤΟΥ WORKFLOW ΚΑΙ ΤΟΥ ΔΙΚΤΥΟΥ ΕΞΥΠΗΡΕΤΗΤΩΝ ΟΠΟΥ ΧΡΗΣΙΜΟΠΟΙΕΙΤΑΙ Ο ΑΛΓΟΡΙΘΜΟΣ ΓΡΑΜΜΗ-ΓΡΑΜΜΗ. 56

ΣΧΗΜΑ 4.2 ΠΑΡΑΔΕΙΓΜΑ ΚΡΙΣΙΜΗΣ ΓΕΦΥΡΑΣ. 57

ΣΧΗΜΑ 4.3 Η ΤΟΠΟΛΟΓΙΑ ΤΟΥ WORKFLOW ΚΑΙ ΤΟΥ ΔΙΚΤΥΟΥ ΕΞΥΠΗΡΕΤΗΤΩΝ ΟΠΟΥ ΧΡΗΣΙΜΟΠΟΙΟΥΝΤΑΙ ΟΙ ΑΛΓΟΡΙΘΜΟΙ ΓΡΑΜΜΗ-ΔΙΑΥΛΟΣ. 63

ΣΧΗΜΑ 5.1 ΠΕΙΡΑΜΑΤΑ ΚΑΤΗΓΟΡΙΑΣ Α ΓΙΑ ΤΟΝ ΑΛΓΟΡΙΘΜΟΥ ΓΡΑΜΜΗ–ΓΡΑΜΜΗ... 84

ΣΧΗΜΑ 5.2 ΠΕΙΡΑΜΑΤΑ ΚΑΤΗΓΟΡΙΑΣ Β ΓΙΑ ΤΟΝ ΑΛΓΟΡΙΘΜΟ ΓΡΑΜΜΗ–ΓΡΑΜΜΗ. 86

ΣΧΗΜΑ 5.3 ΠΕΙΡΑΜΑΤΑ ΚΑΤΗΓΟΡΙΑΣ Γ ΓΙΑ ΤΟΝ ΑΛΓΟΡΙΘΜΟ ΓΡΑΜΜΗ–ΓΡΑΜΜΗ. 87

ΣΧΗΜΑ 5.4 ΕΠΙΔΟΣΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ ΓΡΑΜΜΗ–ΓΡΑΜΜΗ ΚΑΙ ΤΩΝ ΠΑΡΑΛΛΑΓΩΝ ΤΟΥ ΣΕ ΜΕΓΑΛΥΤΕΡΑ ΣΥΣΤΗΜΑΤΑ. 88

ΣΧΗΜΑ 5.5 ΣΥΓΚΡΙΣΗ ΤΩΝ ΠΑΡΑΛΛΑΓΩΝ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ ΓΡΑΜΜΗ–ΓΡΑΜΜΗ ΓΙΑ ΣΥΝΔΕΣΕΙΣ ΜΕ ΜΕΓΑΛΕΣ ΤΑΧΥΤΗΤΕΣ. 89

ΣΧΗΜΑ 5.6 ΣΥΓΚΡΙΣΗ ΤΩΝ ΠΑΡΑΛΛΑΓΩΝ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ ΓΡΑΜΜΗ–ΓΡΑΜΜΗ ΓΙΑ ΣΥΝΔΕΣΕΙΣ ΜΕ ΜΙΚΡΕΣ ΤΑΧΥΤΗΤΕΣ. 90

ΣΧΗΜΑ 5.7 ΠΕΙΡΑΜΑΤΑ ΚΑΤΗΓΟΡΙΑΣ Α ΓΙΑ ΤΟΥΣ ΑΛΓΟΡΙΘΜΟΥΣ ΓΡΑΜΜΗ–ΔΙΑΥΛΟΣ. 91

ΣΧΗΜΑ 5.8 ΠΕΙΡΑΜΑΤΑ ΚΑΤΗΓΟΡΙΑΣ Β ΓΙΑ ΤΟΥΣ ΑΛΓΟΡΙΘΜΟΥΣ ΓΡΑΜΜΗ–ΔΙΑΥΛΟΣ. 92

ΣΧΗΜΑ 5.9 ΠΕΙΡΑΜΑΤΑ ΚΑΤΗΓΟΡΙΑΣ Γ ΓΙΑ ΤΟΥΣ ΑΛΓΟΡΙΘΜΟΥΣ ΓΡΑΜΜΗ–ΔΙΑΥΛΟΣ. 93

ΣΧΗΜΑ 5.10 ΕΠΙΔΟΣΗ ΤΟΥ ΑΛΓΟΡΙΘΜΩΝ ΓΡΑΜΜΗ–ΔΙΑΥΛΟΣ ΣΕ ΜΕΓΑΛΥΤΕΡΑ ΣΥΣΤΗΜΑΤΑ ΜΕ ΔΙΑΥΛΟ ΤΑΧΥΤΗΤΑΣ 100 MBPS. 94

ΣΧΗΜΑ 5.11 ΕΠΙΔΟΣΗ ΤΟΥ ΑΛΓΟΡΙΘΜΩΝ ΓΡΑΜΜΗ–ΔΙΑΥΛΟΣ ΣΕ ΜΕΓΑΛΥΤΕΡΑ ΣΥΣΤΗΜΑΤΑ ΜΕ ΔΙΑΥΛΟ ΤΑΧΥΤΗΤΑΣ 10 MBPS. 95

ΣΧΗΜΑ 5.12 ΕΠΙΔΟΣΗ ΤΟΥ ΑΛΓΟΡΙΘΜΩΝ ΓΡΑΜΜΗ–ΔΙΑΥΛΟΣ ΣΤΟ WORKFLOW ΤΟΥ ΣΧΗΜΑΤΟΣ 3.3. 97

ΣΧΗΜΑ 5.13 ΣΥΓΚΡΙΣΗ ΑΛΓΟΡΙΘΜΩΝ ΓΡΑΜΜΗ–ΔΙΑΥΛΟΣ ΓΙΑ WORKFLOW ΜΕ 19 ΛΕΙΤΟΥΡΓΙΕΣ, ΚΑΤΗΓΟΡΙΑΣ Γ. 98

ΣΧΗΜΑ 5.14 ΕΠΙΔΟΣΗ ΤΟΥ ΑΛΓΟΡΙΘΜΩΝ ΤΥΧΑΙΟΣ ΓΡΑΦΟΣ–ΔΙΑΥΛΟΣ ΣΤΟ WORKFLOW ΤΟΥ ΣΧΗΜΑΤΟΣ 3.2. 101

ΣΧΗΜΑ 5.15 ΕΠΙΔΟΣΗ ΤΟΥ ΑΛΓΟΡΙΘΜΩΝ ΤΥΧΑΙΟΣ ΓΡΑΦΟΣ–ΔΙΑΥΛΟΣ ΣΕ BUSHY WORKFLOWS. 102

ΣΧΗΜΑ 5.16 ΕΠΙΔΟΣΗ ΤΟΥ ΑΛΓΟΡΙΘΜΩΝ ΤΥΧΑΙΟΣ ΓΡΑΦΟΣ–ΔΙΑΥΛΟΣ ΣΕ LENGTHY WORKFLOWS. 103

ΣΧΗΜΑ 5.17 ΣΥΓΚΡΙΣΗ ΑΛΓΟΡΙΘΜΩΝ ΤΥΧΑΙΟΣ ΓΡΑΦΟΣ–ΔΙΑΥΛΟΣ ΓΙΑ WORKFLOWS ΜΕ 19 ΛΕΙΤΟΥΡΓΙΕΣ ΚΑΙ 10 ΕΞΥΠΗΡΕΤΗΤΕΣ, ΚΑΤΗΓΟΡΙΑΣ Γ. 104

ΠΕΡΙΛΗΨΗ

Κωνσταντίνος Σταμκόπουλος του Ευσταθίου και της Σταματίας. MSc, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Ιούλιος, 2006. Αποδοτική Κατανομή Ροών Εργασίας Υπηρεσιών Διαδικτύου. Επιβλέποντες: Ευαγγελία Πιτουρά και Παναγιώτης Βασιλειάδης.

Οι υπηρεσίες διαδικτύου προσφέρουν ένα συστηματικό τρόπο για προγραμματισμό εφαρμογών διαδικτύου. Το πρόβλημα που μας απασχολεί είναι, δοθέντος μιας ροής εργασίας υπηρεσιών διαδικτύου και ενός δικτύου εξυπηρετητών, πώς πρέπει να κατανεμηθούν οι λειτουργίες της ροής εργασίας στους εξυπηρετητές ώστε να ελαχιστοποιείται μία συνάρτηση κόστους και να ικανοποιούνται οι περιορισμοί του χρήστη. Χρησιμοποιούμε δύο συναρτήσεις κόστους που αφορούν το χρόνο εκτέλεσης της ροής εργασίας και την κατανομή του φορτίου στους εξυπηρετητές. Επικεντρωνόμαστε στην περίπτωση που ο χρήστης θέλει να ελαχιστοποιήσει και τις δύο συναρτήσεις κόστους, δηλαδή, να ελαχιστοποιήσει το χρόνο εκτέλεσης της ροής εργασίας και να κατανείμει όσο πιο δίκαια γίνεται το φορτίο στους εξυπηρετητές, να απασχολούνται δηλαδή, όλοι οι εξυπηρετητές τον ίδιο χρόνο. Μελετάμε ξεχωριστά διαφορετικές τοπολογίες της ροής εργασίας και του δικτύου εξυπηρετητών και προτείνουμε αλγορίθμους για την εύρεση λύσης σε κάθε περίπτωση. Η εύρεση της βέλτιστης λύσης στο πρόβλημα, δεν είναι πάντα εφικτή γιατί έχει εκθετική πολυπλοκότητα. Η συνεισφορά της εργασίας είναι η διαστασιοποίηση του σύνθετου προβλήματος που αναφέραμε, ο ορισμός ενός απλού μοντέλου για την περιγραφή του, οι αλγόριθμοι που προτείνουμε για την επίλυση του, καθώς και τα πειράματα που δείχνουν την επίδοση των αλγορίθμων για διαφορετικές αναθέσεις στις παραμέτρους του συστήματος.

EXTENDED ABSTRACT IN ENGLISH

Konstantinos, Stamkopoulos. MSc, Computer Science Department, University of Ioannina, Greece. July, 2006. Efficient Allocation of Web Service Workflows. Thesis Supervisors: Evaggelia Pitoura and Panos Vassiliadis.

The technology of web services offers a systematic way for web applications programming. The problem we work on is, given a workflow of web services' operations and a network of servers, how we must allocate the operations of workflow to the servers, so that to minimize a cost function and do not violate the user's constraints. We use two cost functions that concern the execution time of workflow and the load on servers. We focus on the case where the user wants to minimize both the cost functions, that is, to minimize the execution time of workflow and to balance the load on servers. The load balancing means that all servers spend the same time for processing the workflow. We study different topologies for the workflow and for the network of servers and we propose algorithms which give solutions in each case. Is difficult to find the optimal solution because the complexity is exponential. The contribution of this work is the enumeration of different dimensions of the problem, the definition of a simple model which describes the problem, the algorithms we propose for the solution of the problem and the experiments which indicate the algorithms' performance in different cases of parameters' configuration.

ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ

Τα τελευταία χρόνια υπάρχει μεγάλη εξέλιξη και έρευνα στην τεχνολογία των *υπηρεσιών διαδικτύου* (web services). Υπηρεσία διαδικτύου είναι μία συλλογή από λειτουργίες, οι οποίες μπορούν να κληθούν μέσω του διαδικτύου. Με την τεχνολογία των υπηρεσιών διαδικτύου είναι δυνατόν να πραγματοποιούνται από απλές μέχρι πολυσύνθετες διαδικασίες, όταν συνεργάζονται πολλές υπηρεσίες διαδικτύου. Σε μια τέτοια συνεργασία, οι λειτουργίες διάφορων υπηρεσιών διαδικτύου ανταλλάσσουν μεταξύ τους μηνύματα ώστε να εκτελεστεί μία σύνθετη λειτουργία. Συνολικά οι λειτουργίες και τα μηνύματα που συμμετέχουν στη διαδικασία αυτή αποτελούν μία *ροή εργασίας* ή αλλιώς ένα *workflow*. Δηλαδή, ένα workflow υπηρεσιών διαδικτύου μπορεί να αναπαρασταθεί ως ένα γράφο με κόμβους τις λειτουργίες υπηρεσιών διαδικτύου και ακμές τα μηνύματα-δεδομένα που ανταλλάσσουν οι υπηρεσίες διαδικτύου. Ένα τέτοιο μήνυμα αποτελεί το αποτέλεσμα που δίνει η κλήση της λειτουργίας που το στέλνει και παράλληλα την είσοδο που χρειάζεται για να κληθεί η λειτουργία που το δέχεται.

Ένα workflow, όπου συμμετέχουν πολλές υπηρεσίες διαδικτύου, μπορεί να φιλοξενείται σε ένα δικτυακό τόπο, η δουλειά του οποίου είναι να εξυπηρετεί τέτοια workflows. Θεωρώντας ότι ο δικτυακός τόπος χρησιμοποιεί περισσότερους του ενός εξυπηρετητές-υπολογιστές, οι οποίοι επικοινωνούν μεταξύ τους, το ερώτημα είναι πώς πρέπει να κατανεμηθούν οι λειτουργίες του workflow στους εξυπηρετητές. Οι παράμετροι που σχετίζονται με αυτό το πρόβλημα είναι πολλές και αφορούν τους εξυπηρετητές, το workflow αλλά και τα κριτήρια με τα οποία γίνεται η αξιολόγηση της κατανομής. Ο μεγάλος αριθμός των παραμέτρων σε συνδυασμό με τον μεγάλο αριθμό λύσεων που μπορεί να έχει το πρόβλημα, δυσκολεύει τον ορισμό και τη λύση του προβλήματος.

Η δουλειά που έχει γίνει σε σχετικές εργασίες αφορά κυρίως την κατανομή της ανακάλυψης, της διαχείρισης και της εκτέλεσης σύνθετων υπηρεσιών διαδικτύου. Οι εργασίες που ασχολούνται με την ανακάλυψη υπηρεσιών διαδικτύου προτείνουν μεθόδους που ομαδοποιούν τις υπηρεσίες διαδικτύου. Όμως, δεν ασχολούνται με το πώς πρέπει να κατανεμηθούν οι υπηρεσίες ή, σε περίπτωση που ασχολούνται, δεν εξετάζουν καθόλου την κατανομή του φορτίου. Σε αντίθεση με τις εργασίες αυτές, οι οποίες χρησιμοποιούν κυρίως τη σημασιολογία των υπηρεσιών για την ομαδοποίησή τους, εμείς χρησιμοποιούμε απλές τεχνικές που μειώνουν το χρόνο εκτέλεσης της σύνθετης υπηρεσίας και κατανέμουν δίκαια το φόρτο εργασίας στους εξυπηρετητές. Οι εργασίες που ασχολούνται με τη διαχείριση σύνθετων υπηρεσιών προτείνουν μοντέλα για τον υπολογισμό της ποιότητας υπηρεσίας σύνθετων υπηρεσιών διαδικτύου και είναι χρήσιμα για τον ορισμό του δικού μας μοντέλου. Μερικές από αυτές τις εργασίες ασχολούνται με το θέμα της κατανομής των υπηρεσιών διαδικτύου σε εξυπηρετητές, σε διαφορετικό όμως πλαίσιο από το δικό μας. Για παράδειγμα, είτε θεωρούν κοινότητες υπηρεσιών διαδικτύου με την ίδια λειτουργικότητα είτε προσπαθούν να βελτιώσουν τη λύση χρησιμοποιώντας οσουσδήποτε εξυπηρετητές. Όσον αφορά την κατανομή της εκτέλεσης σύνθετων υπηρεσιών διαδικτύου, προτείνεται μία μέθοδος που προσπαθεί να μειώσει το χρόνο εκτέλεσης μιας σύνθετης υπηρεσίας διαδικτύου, χωρίς όμως να μπορεί να αλλάξει η θέση μιας υπηρεσίας από τον εξυπηρετητή όπου βρίσκεται δημοσιευμένη. Εμείς θέλουμε να δώσουμε μεγαλύτερη ευελιξία στη θέση των υπηρεσιών και οι λειτουργίες των υπηρεσιών να μπορούν να τοποθετηθούν σε οποιονδήποτε εξυπηρετητή.

Το πρόβλημα λοιπόν που μας απασχολεί είναι η αποδοτική κατανομή λειτουργιών υπηρεσιών διαδικτύου ενός workflow σε ένα δίκτυο εξυπηρετητών. Θεωρούμε ως αποδοτική κατανομή, αυτή που αναθέτει τις λειτουργίες στους εξυπηρετητές με τέτοιο τρόπο ώστε να είναι δίκαιη η κατανομή του φορτίου και η εκτέλεση του workflow να είναι όσο το δυνατόν γρηγορότερη. Οι ποσότητες αυτές προσδιορίζονται με συναρτήσεις κόστους, τις οποίες προσπαθούν να ελαχιστοποιήσουν οι αλγόριθμοι που κατασκευάζουμε.

Η προσέγγιση του προβλήματος γίνεται αρχικά με τη διαστασιοποίηση και τη μοντελοποίηση του προβλήματος. Το μοντέλο που χρησιμοποιείται είναι απλό και θεωρεί ένα workflow και ένα δίκτυο εξυπηρετητών. Το πρόβλημα μπορεί να γίνει πολύ πιο δύσκολο αν θεωρήσουμε πολλά workflows, τα οποία μπορεί να έχουν εξάρτηση μεταξύ τους, πρόβλημα που δεν εξετάζεται σε αυτήν την εργασία. Οι κύριες παράμετροι του μοντέλου μας είναι η υπολογιστική ισχύς των εξυπηρετητών, η υπολογιστική ισχύς που χρειάζονται οι λειτουργίες για να εκτελεστούν, τα μηνύματα που στέλνονται μεταξύ των λειτουργιών και οι ταχύτητες των συνδέσεων μεταξύ των εξυπηρετητών. Στη συνέχεια, για διαφορετικές τοπολογίες του workflow αλλά και του δικτύου εξυπηρετητών προτείνουμε αλγορίθμους που δίνουν λύση στο πρόβλημα. Πιο συγκεκριμένα, για το workflow, μελετήσαμε την τοπολογία Γραμμή, όπου οι λειτουργίες του workflow εκτελούνται με τη σειρά η μία μετά την άλλη, και την τοπολογία Τυχαίος Γράφος, όπου το workflow είναι ένας τυχαίος κατευθυνόμενος γράφος. Επίσης, για το δίκτυο εξυπηρετητών, μελετήσαμε της εξής τοπολογίες: τοπολογία Γραμμή, όπου οι εξυπηρετητές διατάσσονται σε μία γραμμή και κάθε εξυπηρετητής συνδέεται μόνο με τους γειτονικούς εξυπηρετητές του και τοπολογία Δίαυλος, όπου όλοι οι εξυπηρετητές συνδέονται μεταξύ τους μέσω ενός διαύλου. Οι συνδυασμοί που μελετάμε για τις τοπολογίες (τοπολογία workflow–τοπολογία δικτύου εξυπηρετητών) είναι Γραμμή– Γραμμή, Γραμμή–Δίαυλος, Τυχαίος Γράφος–Δίαυλος. Για την επινόηση των αλγορίθμων κάναμε πρώτα κάποια πειράματα για να βρούμε τις ιδιότητες που έχουν οι καλές λύσεις του προβλήματος. Τις ιδιότητες αυτές, χρησιμοποιούν οι αλγόριθμοι ώστε να «σπρώχνουν» τη λύση τους σε αποδεκτές τιμές.

Για να βρούμε τη βέλτιστη λύση, στο πρόβλημα που περιγράψαμε, μπορούμε να βρούμε όλες τις δυνατές λύσεις, χρησιμοποιώντας έναν εξαντλητικό αλγόριθμο και να επιλέξουμε την καλύτερη. Η πολυπλοκότητα όμως του εξαντλητικού αλγορίθμου είναι εκθετική ως προς τον αριθμό των εξυπηρετητών. Έτσι για μεγάλο αριθμό λειτουργιών και εξυπηρετητών, η εύρεση της βέλτιστης λύσης είναι αδύνατη. Αντίθετα, οι αλγόριθμοι που προτείνουμε έχουν μικρές πολυπλοκότητες. Συγκεκριμένα, για τις τοπολογίες Γραμμή–Γραμμή προτείνουμε έναν γραμμικό αλγόριθμο με τέσσερις παραλλαγές, που αναθέτει τις λειτουργίες στους κόμβους τη

μία μετά την άλλη, με τη σειρά που βρίσκονται στο workflow, αρχίζοντας την ανάθεση από τον εξυπηρετητή που βρίσκεται στο ένα άκρο του δικτύου και προχωρώντας προς τον εξυπηρετητή που βρίσκεται στο άλλο άκρο του δικτύου. Στις τοπολογίες Γραμμή–Δίαυλος και Τυχαίος Γράφος–Δίαυλος ξεκινάμε με τον αλγόριθμο Fair Load, ο οποίος, κάνει την ανάθεση των λειτουργιών προσπαθώντας να ικανοποιήσει τη βέλτιστη κατανομή φορτίου. Στη συνέχεια, δίνουμε δύο παραλλαγές του Fair Load, τους Fair Load–Tie Resolver$_1$ και Fair Load–Tie Resolver$_2$ οι οποίοι, προσπαθούν να βελτιώσουν το χρόνο εκτέλεσης του workflow, χωρίς όμως να χαλάνε την κατανομή του φορτίου. Τέλος, δίνουμε άλλους δύο αλγορίθμους, τον Fair Load–Merge Messages' Ends και τον Heavy Operations–Large Messages. Ο πρώτος διαφοροποιείται από τον Fair Load–Tie Resolver$_2$ στο ότι προσπαθεί να αποτρέψει την αποστολή μηνυμάτων μεγάλου μεγέθους μέσω του διαύλου με τον κίνδυνο να χαλάσει πολύ την κατανομή του φορτίου. Ο δεύτερος διαφέρει από όλους τους υπόλοιπους και σε κάθε βήμα του κάνει είτε μία «εικονική» ομαδοποίηση δύο οι περισσότερων λειτουργιών σε μία ομάδα λειτουργιών, είτε αναθέτει τις λειτουργίες μιας «εικονικής ομάδας» σε έναν εξυπηρετητή. Η πρώτη επιλογή βελτιώνει τη λύση από τη σκοπιά του χρόνου εκτέλεσης ενώ η δεύτερη από τη σκοπιά της δίκαιης κατανομής φορτίου. Ο Fair Load έχει τετραγωνική πολυπλοκότητα ενώ οι υπόλοιποι έχουν κυβική πολυπλοκότητα.

Για την αξιολόγηση των αλγορίθμων κάνουμε πειράματα όπου συγκρίνουμε την επίδοσή τους με το σύνολο των λύσεων που δίνει ο εξαντλητικός αλγόριθμος. Όπου το σύνολο των λύσεων είναι πολύ μεγάλο και είναι αδύνατο να παραχθεί, παράγουμε ένα τυχαίο υποσύνολό του. Τα πειράματα διακρίνονται σε τρεις κατηγορίες ανάλογα με τις παραμέτρους που κρατάμε σταθερές. Στην κατηγορία Α κρατάμε σταθερές την υπολογιστική ισχύ των κόμβων και την υπολογιστική ισχύ που χρειάζονται οι λειτουργίες για την εκτέλεσή τους ενώ μεταβάλουμε τα μεγέθη των μηνυμάτων και τις ταχύτητες των συνδέσεων. Το αντίθετο κάνουμε στην κατηγορία Β, ενώ στην κατηγορία Γ μεταβάλουμε όλες αυτές τις παραμέτρους.

Στα πειράματα που παραθέτουμε, φαίνεται ότι η επίδοση των αλγορίθμων που προτείνουμε είναι καλύτερη από το αν γίνει μία τυχαία κατανομή των λειτουργιών

στους εξυπηρετητές. Η χρήση των αλγορίθμων αυτών κρίνεται απαραίτητη όταν πρόκειται για την κατανομή ροών εργασίας με πολλές λειτουργίες. Επίσης από τα πειράματα βγάζουμε χρήσιμα συμπεράσματα για το ποιοι αλγόριθμοι είναι καταλληλότεροι σε κάθε περίπτωση.

Η κύρια συμβολή της εργασίας συνοψίζεται στις παρακάτω προτάσεις:

- Διαστασιοποίηση του προβλήματος.
- Ορισμός ενός απλού μοντέλου για την επίλυση του προβλήματος.
- Σχεδιασμός αλγορίθμων για την επίλυση του προβλήματος.
- Πειραματική μελέτη της απόδοσης των αλγορίθμων για πολλαπλές διαστάσεις του προβλήματος.

Στη συνέχεια παραθέτουμε την οργάνωση του τόμου που αποτελείται από έξι κεφάλαια. Στο δεύτερο κεφάλαιο παρουσιάζουμε τις εργασίες που σχετίζονται με το αντικείμενο αυτής της εργασίας. Στο τρίτο κεφάλαιο αρχικά δίνουμε τον ορισμό του workflow και ένα παράδειγμα ενός πραγματικού workflow. Στη συνέχεια κάνουμε τη διαστασιοποίηση και τη μοντελοποίηση του προβλήματος. Στην ενότητα της διαστασιοποίησης απαριθμούμε τις παραμέτρους του workflow, του δικτύου των εξυπηρετητών αλλά και τα χαρακτηριστικά με τα οποία μπορεί να γίνει η αξιολόγηση μίας αντιστοίχισης λειτουργιών σε εξυπηρετητές, δηλαδή μιας λύσης του συστήματος. Στη συνέχεια κάνουμε συγκεκριμένο το πρόβλημα δίνοντας το μοντέλο του. Έπειτα αναλύουμε τις ιδιότητες που έχουν οι καλές λύσεις του προβλήματος και μας βοηθάνε στην κατασκευή των αλγορίθμων. Στο τέλος του τρίτου κεφαλαίου υπάρχουν οι συμβολισμοί και οι τύποι που χρησιμοποιούνται στη συνέχεια του τόμου. Στο τέταρτο κεφάλαιο παραθέτουμε τον εξαντλητικό αλγόριθμο ο οποίος δίνει το σύνολο όλων των λύσεων του προβλήματος και τους αλγορίθμους που κατασκευάσαμε για τρεις διαφορετικούς συνδυασμούς στις τοπολογίες του workflow και του δικτύου εξυπηρετητών. Στο πέμπτο κεφάλαιο, παρουσιάζουμε τα πειράματα που κάναμε για όλους τους τύπους των αλγορίθμων και για τις τρεις κατηγορίες διαφορετικών αναθέσεων στις παραμέτρους του συστήματος. Τέλος, στο έκτο κεφάλαιο, ανακεφαλαιώνουμε και συζητάμε πώς μπορεί να συνεχισθεί η έρευνα.

ΚΕΦΑΛΑΙΟ 2. ΣΧΕΤΙΚΕΣ ΕΡΓΑΣΙΕΣ

2.1 Υπηρεσίες Διαδικτύου
2.2 Ανακάλυψη Υπηρεσιών Διαδικτύου με Κατανεμημένο Τρόπο
2.3 Διαχείριση Σύνθετων Υπηρεσιών με Βάση την Ποιότητα της Υπηρεσίας
2.4 Διαχείριση Δυναμικών Σύνθετων Υπηρεσιών σε Δυναμικά Δίκτυα
2.5 Εκτέλεση Σύνθετων Υπηρεσιών με Κατανεμημένο Τρόπο
2.6 Βελτιστοποίηση Ερωτήσεων για Σύνθετες Υπηρεσίες Διαδικτύου
2.7 Τοποθέτηση Αντικειμένων σε Απομακρυσμένη Μνήμη
2.8 Ανακεφαλαίωση

Το συγκεκριμένο κεφάλαιο αναφέρεται συνοπτικά σε εργασίες που είναι σχετικές με το περιεχόμενο της μεταπτυχιακής εργασίας. Οι εργασίες αυτές αφορούν κυρίως σύνθετες υπηρεσίες διαδικτύου και πραγματεύονται τα θέματα της ανακάλυψης, διαχείρισης, και εκτέλεσης σύνθετων υπηρεσιών διαδικτύου με κατανεμημένο τρόπο. Στην αρχή του κεφαλαίου γίνεται μία εισαγωγή για τις υπηρεσίες διαδικτύου ενώ στο τέλος του κεφαλαίου γίνεται μία ανακεφαλαίωση των σχετικών εργασιών και αναλύουμε γιατί και πού υπάρχει χώρος για επιπλέον έρευνα.

2.1. Υπηρεσίες Διαδικτύου

Μία υπηρεσία διαδικτύου [ACKM04][Kreg01] είναι μια διεπαφή που περιγράφει μια συλλογή λειτουργιών που προσφέρονται από το διαδίκτυο και προσπελαύνονται μέσω τυποποιημένων XML μηνυμάτων. Η διεπαφή μιας υπηρεσίας διαδικτύου

κρύβει τις λεπτομέρειες της εφαρμογής της υπηρεσίας, επιτρέποντάς της να χρησιμοποιηθεί ανεξάρτητα από το υλικό και το λογισμικό στο οποίο εφαρμόζεται και επίσης από τη γλώσσα στην οποία είναι γραμμένη. Έτσι οι χρήστες τέτοιων υπηρεσιών δε χρειάζεται να γνωρίζουν τις λεπτομέρειες της εφαρμογής της υπηρεσίας αλλά το μόνο που πρέπει να κάνουν είναι να στέλνουν και να λαμβάνουν μηνύματα. Οι υπηρεσίες διαδικτύου μπορούν να χρησιμοποιηθούν μόνες τους ή σε συνεργασία με άλλες υπηρεσίες για να πραγματοποιήσουν μια συναλλαγή.

Η αρχιτεκτονική των υπηρεσιών διαδικτύου βασίζεται σε τρεις ρόλους: τον παροχέα της υπηρεσίας διαδικτύου (service provider), την υπηρεσία καταλόγου (service registry), όπου δημοσιεύονται υπηρεσίες διαδικτύου και τον χρήστη της υπηρεσίας διαδικτύου (service client). Ο παροχέας μιας υπηρεσίας διαδικτύου δημοσιεύει στην υπηρεσία καταλόγου την διεπαφή της υπηρεσίας σε γλώσσα WSDL. Ο χρήστης μπορεί να ανακαλύψει υπηρεσίες διαδικτύου που τον ενδιαφέρουν από την υπηρεσία καταλόγου και να ανακτήσει την περιγραφή τους. Αφού ο χρήστης ανακτήσει την περιγραφή μιας υπηρεσίας μπορεί να καλέσει την υπηρεσία εγκαθιδρύοντας μία απευθείας σύνδεσή του με τον παροχέα της υπηρεσίας. Η επικοινωνία μεταξύ του χρήστη και του παροχέα της υπηρεσίας γίνεται χρησιμοποιώντας το SOAP πρωτόκολλο, το οποίο βασίζεται στο HTTP πρωτόκολλο.

2.2. Ανακάλυψη Υπηρεσιών Διαδικτύου με Κατανεμημένο Τρόπο

Οι σημερινές μέθοδοι για την ανακάλυψη υπηρεσιών διαδικτύου βασίζονται σε κεντρικοποιημένες προσεγγίσεις όπου οι υπηρεσίες διαδικτύου προσδιορίζονται με βάση την λειτουργικότητά τους και με βάση λέξεις κλειδιά. Όσο όμως ο αριθμός των υπηρεσιών διαδικτύου αυξάνεται, τόσο μεγαλώνει η ανάγκη για κατανομή της ανακάλυψης των υπηρεσιών διαδικτύου, η οποία θα στηρίζεται στη σημασιολογία και στην συμπεριφορά των υπηρεσιών διαδικτύου. Στην ενότητα αυτή παρουσιάζονται μερικά άρθρα που σκοπός τους είναι να κατανείμουν την ανακάλυψη υπηρεσιών διαδικτύου χρησιμοποιώντας τη σημασιολογία τους και τη συμπεριφορά τους.

Στο άρθρο [DHM+04] περιγράφεται μια τεχνική για αναζήτηση παρόμοιων υπηρεσιών διαδικτύου (web services), δηλαδή υπηρεσίες διαδικτύου που δέχονται παρόμοιες εισόδους ή παράγουν παρόμοιες εξόδους και υπηρεσίες διαδικτύου που συνθέτουν την ζητούμενη υπηρεσία αν εκτελεστούν με συγκεκριμένο τρόπο. Μια τέτοια τεχνική είναι χρήσιμη και για την ομαδοποίηση ενός μεγάλου συνόλου από υπηρεσίες διαδικτύου ώστε η τοποθέτησή τους και η αναζήτησή τους να μπορεί να γίνει με κατανεμημένο τρόπο. Ο αλγόριθμος που χρησιμοποιούν οι συγγραφείς για να βρουν παρόμοιες υπηρεσίες διαδικτύου βασίζεται σε μία μέθοδο που ομαδοποιεί τα ονόματα των παραμέτρων μιας συλλογής υπηρεσιών διαδικτύου σε «έννοιες» (semantically meaningful concepts). Υποθέτουν ότι τα ονόματα των παραμέτρων είναι μια ακολουθία από λέξεις η μία δίπλα στην άλλη με την προϋπόθεση ότι το πρώτο γράμμα κάθε λέξης είναι κεφαλαίο ενώ τα υπόλοιπα μικρά. Δύο ή περισσότερες λέξεις που απαντώνται συχνά μαζί (στο ίδιο όνομα παραμέτρου) τείνουν να εκφράσουν την ίδια έννοια. Οπότε η μέθοδος ψάχνει για συσχετιστικούς κανόνες της μορφής $\text{λέξη}_1 \rightarrow \text{λεξη}_2$ (support, confidence). Τέτοιοι συσχετιστικοί κανόνες παράγονται από έναν "apriori" αλγόριθμο. Για την εξαγωγή καλών κανόνων πρέπει να γίνεται προεπεξεργασία των λέξεων (π.χ., να αντιστοιχίζονται στη ρίζα, να αφαιρούνται οι προθέσεις κ.λ.π.). Ο αλγόριθμος που ομαδοποιεί παρόμοιες υπηρεσίες διαδικτύου ταξινομεί τους συσχετιστικούς κανόνες πρώτα ως προς την αξιοπιστία (confidence) και μετά ως προς την υποστήριξη (support) και δουλεύει ως εξής: αρχικά θεωρεί κάθε λέξη ως ξεχωριστή έννοια, δηλαδή, κάθε λέξη αποτελεί μία ομάδα. Στη συνέχεια παίρνει από την ταξινομημένη λίστα έναν-έναν τους συσχετιστικούς κανόνες εφόσον έχουν ικανοποιητική αξιοπιστία και υποστήριξη και αν οι λέξεις δεξιά και αριστερά του κανόνα βρίσκονται σε διαφορετικές ομάδες συνενώνει τις δύο ομάδες σε μία. Υπάρχει περίπτωση ο αλγόριθμος να μη συνενώσει τις δύο ομάδες σε μία αλλά να φτιάξει από αυτές δύο ή τρεις νέες ομάδες. Αυτό γίνεται προκειμένου η ομαδοποίηση που προκύπτει να είναι καλύτερη από την προηγούμενη, δηλαδή, να υπάρχει μεγαλύτερη συσχέτιση μεταξύ των λέξεων μιας ομάδας και μικρότερη συσχέτιση μεταξύ λέξεων διαφορετικών ομάδων από ότι στην προηγούμενη ομαδοποίηση. Όταν εξεταστούν όλοι οι κανόνες, κάθε λέξη αντικαθιστάται από την έννοια στην οποία ανήκει, εξάγονται καινούριοι συσχετιστικοί κανόνες και ξανατρέχει ο αλγόριθμος. Η διαδικασία αυτή

επαναλαμβάνεται έως ότου προκύψουν νέες ομάδες. Τέλος, οι έννοιες που προκύπτουν χρησιμοποιούνται για να βρεθεί η ομοιότητα μεταξύ υπηρεσιών διαδικτύου. Όλες οι έννοιες που σχετίζονται με μια υπηρεσία τοποθετούνται μέσα σε ένα σάκο (bag). Έτσι το μέτρο ομοιότητας μεταξύ δύο υπηρεσιών διαδικτύου είναι η τιμή TF/IDF (Term Frequency / Inverse Document Frequency)που προκύπτει μεταξύ των δύο αντίστοιχων σάκων των υπηρεσιών. Εκτός από σάκους με έννοιες μπορούν να χρησιμοποιηθούν σάκοι με τις λέξεις των ονομάτων των παραμέτρων εισόδου ή εξόδου, ή με τις λέξεις από τις περιγραφές (abstract) των υπηρεσιών. Σε αυτήν την περίπτωση το μέτρο ομοιότητας θα είναι ένας γραμμικός συνδυασμός μεταξύ των TF/IDF τιμών όλων των ζευγών από σάκους.

Στη δημοσίευση [ESAA04] προτείνεται ένα μοντέλο με το οποίο η ανακάλυψη υπηρεσιών διαδικτύου θα γίνεται με μη-κεντρικοποιημένο τρόπο και θα βασίζεται εκτός από την λειτουργικότητα και στην συμπεριφορά των υπηρεσιών. Στο μοντέλο αυτό μία υπηρεσία διαδικτύου χαρακτηρίζεται από ένα πεπερασμένο αυτόματο το οποίο περιγράφει τη ροή της εκτέλεσης της υπηρεσίας, δηλαδή τη συμπεριφορά της. Οι καταστάσεις του αυτόματου αντιστοιχούν στις λειτουργίες της υπηρεσίας διαδικτύου ενώ οι μεταβάσεις του στις δυνατές μεταβάσεις από λειτουργία σε λειτουργία. Το αυτόματο για μια υπηρεσία διαδικτύου μπορεί να εξαχθεί από διάφορες μεθόδους που μετατρέπουν BPEL (Business Process Execution Language for web services) ή DAML-S (DARPA Agent Markup Language – Services) περιγραφές σε πεπερασμένα αυτόματα. Μία εκτέλεση μιας υπηρεσίας διαδικτύου αντιστοιχεί σε ένα μονοπάτι από την αρχική ως την τελική κατάσταση του αυτομάτου της. Αφαιρώντας τους κύκλους από κάθε τέτοιο μονοπάτι παίρνουμε ένα PFA (Path Finite Automaton). Είναι προφανές ότι μία υπηρεσία διαδικτύου μπορεί να έχει πολλά PFA. Το μοντέλο τοποθετεί τις υπηρεσίες διαδικτύου σε ένα δίκτυο Chord [SMKK+01]. Το Chord είναι ένα P2P σύστημα στο οποίο οι κόμβοι εισάγονται πάνω σε ένα δακτύλιο. Εκτός από τις συνδέσεις που υπάρχουν μεταξύ γειτονικών κόμβων στο δακτύλιο υπάρχουν και επιπλέον συνδέσεις για μεγαλύτερη αποτελεσματικότητα στην αναζήτηση. Μία υπηρεσία διαδικτύου τοποθετείται σε τόσους κόμβους στο δακτύλιο του Chord όσα είναι και τα PFA της χρησιμοποιώντας κατακερματισμό. Ως κλειδί για τον κατακερματισμό χρησιμοποιείται η κανονική έκφραση που περιγράφει

το αντίστοιχο PFA. Αν κάποιος χρήστης ψάχνει μία συγκεκριμένη υπηρεσία πρέπει να φτιάξει το αυτόματο που την περιγράφει. Η κανονική έκφραση αυτού του αυτόματου χρησιμοποιείται σαν κλειδί για τη δρομολόγηση της ερώτησης. Ο κόμβος που είναι υπεύθυνος για το κλειδί αυτό θα λάβει την ερώτηση και θα επιστρέψει όποιες υπηρεσίες συμφωνούν με αυτή που περιγράφει ο χρήστης. Στην ίδια δημοσίευση οι συγγραφείς προτείνουν ένα μοντέλο για την διαβάθμιση των υπηρεσιών διαδικτύου με βάση την αξιοπιστία των προμηθευτών και των ποιότητα της υπηρεσίας. Σύμφωνα με αυτό οι χρήστες έχουν τη δυνατότητα να ψηφίζουν για την αξιοπιστία των προμηθευτών και την ποιότητα της υπηρεσίας. Προκύπτουν όμως δύο προβλήματα: μερικοί χρήστες μπορεί να μεροληπτούν, και μερικοί χρήστες μπορεί να ψηφίζουν πολλές φορές για να επηρεάσουν τις εκτιμήσεις. Το πρώτο μπορεί να αντιμετωπιστεί εύκολα λαμβάνοντας υπόψη την αξιοπιστία του χρήστη. Το δεύτερο είναι δυσκολότερο και προτείνεται η θεωρία sketch για την επίλυσή του.

Στο άρθρο [LiZh05] παρουσιάζεται μια άλλη αρχιτεκτονική για την ανακάλυψη υπηρεσιών διαδικτύου που βασίζεται επίσης στο σύστημα Chord. Στη συγκεκριμένη αρχιτεκτονική οι προμηθευτές υπηρεσιών διαδικτύου τοποθετούνται όλοι σε ένα Chord δίκτυο αλλά εκτός από τις συνδέσεις του Chord μεταξύ των κόμβων, τοποθετούνται και επιπλέον συνδέσεις οι οποίες δημιουργούν ένα σημασιολογικό δίκτυο. Στο σημασιολογικό δίκτυο οι συνδέσεις δημιουργούνται με βάση τη λειτουργικότητα (εισόδους-εξόδους) των υπηρεσιών, την κατηγορία τους και των λέξεων κλειδιών τους. Όταν ένας προμηθευτής εισάγεται στο δίκτυο, αρχικά εισάγεται στο δίκτυο Chord σύμφωνα με το κλειδί που παίρνει κατακερματίζοντας με SHA-1 συναρτήσεις κατακερματισμού τα ονόματα των υπηρεσιών που προσφέρει και τις λέξεις κλειδιά που χαρακτηρίζουν την υπηρεσία. Στην συνέχεια, για την εισαγωγή του στο σημασιολογικό δίκτυο, ο προμηθευτής επιλέγει τυχαία έναν κόμβο του δικτύου και μια καινούρια σύνδεση τοποθετείται μεταξύ τους. Η σύνδεση αυτή περιγράφει το πόσο όμοιοι είναι οι δύο κόμβοι σύμφωνα με τη λειτουργικότητα τους, την κατηγορία τους και τις λέξεις κλειδιά των υπηρεσιών διαδικτύου που προσφέρουν. Για την τοποθέτηση κι άλλων συνδέσεων του καινούριου κόμβου με άλλους κόμβους του δικτύου, ο καινούριος κόμβος χρησιμοποιεί τις συνδέσεις του κόμβου που μόλις συνδέθηκε και με την χρήση κανόνων δημιουργεί νέες συνδέσεις

με άλλους κόμβους του δικτύου. Η αναζήτηση μιας υπηρεσίας διαδικτύου γίνεται και στο Chord και στο σημασιολογικό δίκτυο. Στο Chord δίκτυο η ερώτηση προωθείται στον κόμβο που είναι υπεύθυνος για το κλειδί που προκύπτει από τον κατακερματισμό της υπηρεσίας που αναζητείται. Στο σημασιολογικό δίκτυο η ερώτηση προωθείται σε γειτονικούς κόμβους σύμφωνα με τις σημασιολογικές συνδέσεις.

Μια διαφορετική προσέγγιση για ανακάλυψη υπηρεσιών διαδικτύου με κατανεμημένο τρόπο που βασίζεται κι αυτή στο Chord, προτείνεται στη δημοσίευση [ScPa04]. Στην συγκεκριμένη προσέγγιση, μία υπηρεσία διαδικτύου περιγράφεται από ένα σύνολο από λέξεις κλειδιά. Τοποθετώντας τις λέξεις κλειδιά σε έναν πολυδιάστατο χώρο, η υπηρεσία διαδικτύου είναι ένα σημείο του πολυδιάστατο χώρου. Μια ιδιότητα που προκύπτει είναι ότι υπηρεσίες διαδικτύου που έχουν κοινές λέξεις κλειδιά ή λέξεις κλειδιά με μικρή λεξικογραφική απόσταση, βρίσκονται κοντά στον πολυδιάστατο χώρο. Από τον πολυδιάστατο χώρο οι υπηρεσίες διαδικτύου απεικονίζονται στον μονοδιάστατο χώρο διατηρώντας όσο το δυνατόν περισσότερο αυτή την ιδιότητα με τη χρήση μιας SFC (Space Filling Curve) καμπύλης. Έτσι οι υπηρεσίες διαδικτύου είναι πια σημεία πάνω στην SFC καμπύλη. Ένα σύνολο από κοντινές υπηρεσίες διαδικτύου στον πολυδιάστατο χώρο μπορεί να απεικονιστεί σε πολλά τμήματα της SFC καμπύλης τα οποία ονομάζονται clusters. Για την αναζήτηση μιας υπηρεσίας διαδικτύου γίνεται μια ερώτηση που αποτελείται από λέξεις κλειδιά, τμήματα λέξεων κλειδιών ή wildcards η οποία μπορεί να απεικονιστεί σε περιοχές του πολυδιάστατου χώρου και επομένως στα αντίστοιχα clusters της SFC καμπύλης. Οι υπηρεσίες διαδικτύου τοποθετούνται σε ένα Chord δίκτυο αντιστοιχίζοντας την θέση τους στην SFC καμπύλη στον κατάλληλο κόμβο του Chord. Ένα cluster μπορεί να τοποθετείται σε περισσότερους από έναν κόμβους και ένας κόμβος μπορεί να περιέχει περισσότερα του ενός clusters. Στην περίπτωση που η ερώτηση δεν περιέχει τμήματα λέξεων κλειδιών και wildcards αντιστοιχίζεται σε ένα σημείο της SFC καμπύλης και επομένως σε ένα κόμβο, οπότε η αναζήτηση είναι εύκολη και χρειάζεται απλά η δρομολόγηση της ερώτησης στο συγκεκριμένο κόμβο. Δεν συμβαίνει όμως το ίδιο όταν η ερώτηση περιέχει τμήματα λέξεων κλειδιών ή wildcards. Σε αυτήν την περίπτωση η ερώτηση μπορεί να αντιστοιχίζεται σε πολλά

clusters της SFC καμπύλης τα οποία μπορεί να βρίσκονται τοποθετημένα σε πολλούς διασκορπισμένους κόμβους του δικτύου. Όταν συμβαίνει αυτό είναι σημαντικό να γίνεται βελτιστοποίηση της ερώτησης ώστε να μην στέλνεται η ερώτηση σε clusters χωρίς πληροφορία. Τα clusters χωρίς πληροφορία είναι αυτά που δεν έχουν κάποια απάντηση για τον απλό λόγο ότι δεν αντιστοιχούν σε όλα τα σημεία του πολυδιάστατου χώρου λέξεις κλειδιά, έτσι μερικές περιοχές που καλύπτουν τα wildcards ή τα τμήματα λέξεων κλειδιών δεν έχουν καθόλου απαντήσεις. Η βελτιστοποίηση γίνεται κατασκευάζοντας ένα δέντρο με φύλλα τα clusters και στη συνέχεια κλαδεύοντας τα υπό-δέντρα που δεν περιέχουν έγκυρη πληροφορία. Επειδή οι υπηρεσίες διαδικτύου δεν κατανέμονται ομοιόμορφα στον πολυδιάστατο χώρο, η αντιστοίχισή τους στην SFC καμπύλη δεν είναι ομοιόμορφη και επομένως ούτε η κατανομή του φορτίου στους κόμβους του Chord είναι ομοιόμορφη. Για την εξισορρόπηση της κατανομής του φορτίου οι συγγραφείς χρησιμοποιούν δύο τεχνικές. Η πρώτη εφαρμόζεται κατά την εισαγωγή ενός κόμβου στο δίκτυο και τοποθετεί τον καινούριο κόμβο δίπλα σε κόμβους με μεγάλο φόρτο. Η δεύτερη εφαρμόζεται ενώ το δίκτυο βρίσκεται σε λειτουργία με τη χρήση δύο αλγορίθμων. Ο πρώτος μετακινεί φόρτο εργασίας από ένα φορτωμένο κόμβο στους διπλανούς και ο δεύτερος χρησιμοποιεί εικονικούς κόμβους και όταν ένας φυσικός κόμβος υπερβεί ένα όριο μεταφέρει μερικούς εικονικούς του κόμβους στην κατοχή άλλων φυσικών κόμβων με μικρό φόρτο εργασίας.

Μία άλλη προσέγγιση για κατανομή της ανακάλυψης υπηρεσιών διαδικτύου παρουσιάζεται στο άρθρο [DoZV05]. Στην προσέγγιση αυτή η ανακάλυψη μιας υπηρεσίας γίνεται με βάση το context και οι κατάλογοι των υπηρεσιών κατανέμονται σε μία P2P αρχιτεκτονική. Context ονομάζεται η πληροφορία που σχετίζεται με την υπηρεσία αλλά και τον χρήστη της υπηρεσίας και χρησιμοποιείται για την αξιολόγηση της χρησιμότητας των αποτελεσμάτων που επιστρέφει η υπηρεσία. Context μιας υπηρεσίας διαδικτύου μπορεί να είναι για παράδειγμα η τοποθεσία της, η έκδοσή της, η ταυτότητα του προμηθευτή, ο τύπος των αποτελεσμάτων ή το κόστος της υπηρεσίας. Context του χρήστη μπορεί να είναι η τοποθεσία του, χρονικοί περιορισμοί και προτιμήσεις που θέτει ο χρήστης, χαρακτηριστικά της συσκευής. Κατά την αναζήτηση μιας υπηρεσίας διαδικτύου για να επιστραφεί μια υπηρεσία στο

χρήστη πρέπει το context της υπηρεσίας και το context του χρήστη να ταιριάζουν. Στο μοντέλο που χρησιμοποιείται οι υπηρεσίες διαδικτύου κατανέμονται σε κατηγορίες σύμφωνα με το context τους. Η αναπαράσταση των κατηγοριών γίνεται με ένα γράφο ο οποίος διαφέρει από ένα δένδρο μόνο στο ότι μπορεί να περιέχει φύλλα που ανήκουν σε περισσότερα του ενός υπό-δένδρα και επομένως μπορεί να υπάρχουν περισσότερα μονοπάτια από τη ρίζα σε κάποιο φύλλο. Οι υπηρεσίες διαδικτύου αναπαριστώνται ως φύλλα. Το context αναπαριστάται σαν ένα σύνολο από ζεύγη κλειδιού-τιμής. Σε υπηρεσίες διαμοιρασμού εικόνων για παράδειγμα, κλειδιά θα μπορούσαν να είναι ο τύπος της εικόνας (jpg, mpg) η ανάλυση της εικόνας ή η τοποθεσία όπου τραβήχτηκαν. Κάθε εσωτερικός κόμβος του γράφου είναι ένα ζεύγος κλειδιού-τιμής. Έτσι όλα τα ζεύγη κλειδιού-τιμής των κόμβων που ανήκουν σε ένα μονοπάτι από τη ρίζα σε μια υπηρεσία αποτελούν ένα context της υπηρεσίας. Οι συγγραφείς αναφέρονται σε μια αρχιτεκτονική όπου οι υπηρεσίες διαδικτύου προσφέρονται από κινητές συσκευές και οι χρήστες επίσης προσπελαύνουν τις υπηρεσίες μέσω κινητών συσκευών. Για τη διαχείριση αυτών των συσκευών ο γεωγραφικός δισδιάστατος χώρος χωρίζεται σε κελιά και σε κάθε κελί υπάρχει ένας διαχειριστής που είναι υπεύθυνος να παρακολουθεί τις συσκευές μέσα στο συγκεκριμένο κελί. Ένας διαχειριστής διατηρεί έναν κατάλογο με της υπηρεσίες που υπάρχουν στο κελί του με τη μορφή του γράφου όπως περιγράφτηκε προηγουμένως. Όταν ένας χρήστης ψάχνει μία υπηρεσία διαδικτύου, κατασκευάζεται μία ερώτηση που περιέχει το context του η οποία στέλνεται στον διαχειριστή του κελιού στο οποίο βρίσκεται εκείνη τη στιγμή ο χρήστης. Η ερώτηση αποτιμάται τοπικά και στέλνεται επίσης σε γειτονικά κελιά (μέχρι έναν ορίζοντα) για αποτίμηση. Για να μην επιβαρύνεται όμως πολύ το δίκτυο υπάρχει και μια δεύτερη λύση στην αποτίμηση μιας ερώτησης, αυτή του caching. Σε αυτήν την περίπτωση, εκτός από τον κύριο κατάλογο υπηρεσιών που διατηρεί ένας διαχειριστής, διατηρεί και έναν δεύτερο κατάλογο με υπηρεσίες διαδικτύου που του έχουν ζητηθεί αλλά δεν βρίσκονται στο δικό του κελί. Για να επιτευχθεί αυτό πρέπει ο διαχειριστής να χειρίζεται επιπλέον μία λίστα με τους διαχειριστές που έχουν κάνει caching υπηρεσίες για τις οποίες είναι ο ίδιος υπεύθυνος, έτσι ώστε όταν μετακινείται ένας προμηθευτής, να ενημερώνονται όλοι αυτοί οι διαχειριστές που έχουν κάνει cache την υπηρεσία που προσφέρει ο

προμηθευτής. Η αρχιτεκτονική που περιγράφτηκε με τους διαχειριστές των κελιών μπορεί να θεωρηθεί σαν ένα P2P σύστημα.

2.3. Διαχείριση Σύνθετων Υπηρεσιών με Βάση την Ποιότητα της Υπηρεσίας

Με τον όρο ποιότητα υπηρεσίας (QoS), εννοούμε όλα εκείνα τα ποσοτικά και ποιοτικά χαρακτηριστικά, λειτουργικές και μη λειτουργικές απαιτήσεις, σύμφωνα με τα οποία συμφωνούν ο προμηθευτής και ο πελάτης της υπηρεσίας. Η ποιότητα μιας υπηρεσίας μπορεί να προσδιοριστεί από πολλές παραμέτρους ή αλλιώς διαστάσεις. Στη συνέχεια παρουσιάζονται δύο προσεγγίσεις όπου η διαχείριση μιας σύνθετης υπηρεσίας βασίζεται στην ποιότητα της υπηρεσίας. Μια σύνθετη υπηρεσία αναπαριστάται με ένα workflow από υπηρεσίες διαδικτύου ή από εργασίες γενικότερα. Η διαχείριση ενός workflow έχει ως σκοπό να εγγυάται την ποιότητα υπηρεσίας του workflow στο χρήστη.

Στη δημοσίευση [CSMA+04] οι συγγραφείς προτείνουν ένα μοντέλο που περιγράφει την ποιότητα μιας υπηρεσίας ενός workflow με τρεις διαστάσεις: το χρόνο απόκρισης, το κόστος και την αξιοπιστία. Ο χρόνος απόκρισης μιας εργασίας ορίζεται ως το χρόνο που απαιτείται για να τεθεί σε λειτουργία η εργασία, να εκτελεστεί και να περιμένει σε ουρές πριν την εκτέλεσή της. Το κόστος μιας εργασίας περιλαμβάνει τα κόστη για την διαχείριση και παρακολούθηση των workflows, κόστη για υλικά και μηχανήματα, καθώς και κόστη για ανθρώπινη εργασία. Η αξιοπιστία της υπηρεσίας συσχετίζεται με τις αποτυχίες του συστήματος (λειτουργικό σύστημα, πρωτόκολλο επικοινωνίας, υλικό) και τις αποτυχίες που προκύπτουν από εξαιρέσεις κατά τη διάρκεια της διαδικασίας. Οι τιμές που μπορούν να πάρουν οι τρεις διαστάσεις του μοντέλου που μόλις αναφέρθηκαν κατατάσσονται σε δύο κατηγορίες ανάλογα με την πληροφορία που δίνουν: στη βασική κατηγορία περιλαμβάνονται η μέγιστη τιμή, η ελάχιστη τιμή και ο μέσος όρος και στην κατηγορία που περιγράφει την κατανομή μίας παραμέτρου περιλαμβάνονται οι συναρτήσεις κατανομής όπως κανονική, εκθετική ή ομοιόμορφη, ή εναλλακτικά τα ιστογράμματα. Η δομή ενός workflow μπορεί να περιέχει διακλαδώσεις τύπου "and" (παράλληλη εκτέλεση πολλών εργασιών) και διακλαδώσεις τύπου "or" (επιλογή εκτέλεσης μιας από τις εργασίες της

διακλάδωσης με συγκεκριμένη πιθανότητα). Το μοντέλο όπως περιγράφτηκε χρησιμοποιείται σε ένα σύστημα, το METEOR που είναι υπεύθυνο για την διαχείριση workflows που επεξεργάζονται ακολουθίες DNA. Κατά την εκτέλεση ενός workflow το METEOR παρακολουθεί τις τιμές που παίρνουν οι παράμετροι ώστε να υπολογίζει περιοδικά την ποιότητα της υπηρεσίας του workflow και να προειδοποιεί όταν η ποιότητα δεν είναι καλή και πλησιάζει τα κατώφλια που έχουν τεθεί αρχικά για τις απαιτήσεις του συστήματος. Όταν πρέπει να υπολογιστεί η ποιότητα ενός workflow που πρόκειται να εκτελεστεί τότε το METEOR λαμβάνει υπόψη εκτός από τις αρχικές προσεγγίσεις, τις μετρήσεις ποιότητας που έγιναν στις επιμέρους εργασίες του workflow σε προηγούμενες εκτελέσεις. Αν μια εργασία δεν έχει εκτελεστεί ακόμα και δεν υπάρχουν μετρήσεις τότε λαμβάνονται υπόψη μόνο οι αρχικές προδιαγραφές ποιότητας (π.χ., μέγιστος χρόνος απόκρισης της εργασίας) που έθεσε αρχικά ο σχεδιαστής του συστήματος. Σημειώνεται ότι και οι πιθανότητες μετάβασης στις διαφορετικές εργασίες μιας "or" διακλάδωσης αλλάζουν ώστε να συμφωνούν με τις μετρήσεις από τις εκτελέσεις. Για τον υπολογισμό της ποιότητας ενός workflow χρησιμοποιείται ο αλγόριθμος Stochastic Workflow Reduction. Ο αλγόριθμος εφαρμόζει επαναληπτικά ένα σύνολο από κανόνες μείωσης του workflow οι οποίοι αλλάζουν την δομή του έως ότου μπορεί να αναπαρασταθεί με μία απλή εργασία. Κατά την εφαρμογή ενός κανόνα μείωσης, δύο οι περισσότερες εργασίες του workflow αντικαθιστώνται από μία απλή εργασία στην οποία ανατίθεται ο χρόνος απόκρισης, το κόστος και η αξιοπιστία όλων των εργασιών που αντικατέστησε. Οι κανόνες μείωσης μετατρέπουν σε μία απλή εργασία τις ακόλουθες δομές: ακολουθίες εργασιών, διακλαδώσεις "and", διακλαδώσεις "or", κύκλους και άλλες δομές.

Στο άρθρο [GiWW02] παρουσιάζεται ένα εργαλείο για την αυτόματη ρύθμιση των παραμέτρων ενός συστήματος διαχείρισης workflows. Ένα τέτοιο σύστημα είναι ένα κατανεμημένο σύστημα που αποτελείται από workflow servers, application servers και communication middleware servers. Το ζητούμενο είναι να γίνει μία καλή ανάθεση στις τιμές των παραμέτρων του συστήματος ώστε να είναι εγγυημένη η ποιότητα των υπηρεσιών, δηλαδή, καλός χρόνος απόκρισης και ρυθμοαπόδοση του workflow και μεγάλη διαθεσιμότητα. Οι παράμετροι που καθορίζουν το πρόβλημα είναι πολλές και αφορούν εκτός από τα κατώφλια που βάζει ο χρήστης του εργαλείου

για χρόνο απόκρισης, ρυθμοαπόδοσης και διαθεσιμότητα, τους servers και τα workflows. Οι παράμετροι σχετικοί με τους servers είναι ο αριθμός των αντιγράφων κάθε τύπου server (workflow, application, communication) που πρέπει να δημιουργηθούν για την αύξηση της διαθεσιμότητας και της αποτελεσματικότητας, το φορτίο του κάθε server, ο μέσος χρόνος για να αποτύχει ένας server, ο μέσος χρόνος για να επιδιορθωθεί, ο μέσος χρόνος για την εξυπηρέτηση μιας αίτησης τόσο την πρώτη φορά, όσο και τις επόμενες φορές που γίνεται η αίτηση και ο μέσος χρόνος παραμονής σε κάθε server. Οι παράμετροι που έχουν να κάνουν με τα workflows είναι ο ρυθμός άφιξης των workflows στο σύστημα διαχείρισης και οι παράμετροι του εκάστοτε workflow. Οι παράμετροι ενός συγκεκριμένου workflow είναι ο αριθμός των αιτήσεων που παράγονται σε κάθε κατάσταση (activity) του workflow, ο μέσος χρόνος αναμονής σε κάθε κατάσταση και οι πιθανότητες που αντιστοιχούν στις μεταβάσεις από κατάσταση σε κατάσταση. Για την περιγραφή της ροής της διαδικασίας των workflows (συμπεριλαμβανομένων και των κύκλων) χρησιμοποιούνται μοντέλα Markov. Επίσης, μοντέλα Markov μπορούν να χρησιμοποιηθούν για τον υπολογισμό του μέσου χρόνου αποτυχίας και επιδιόρθωσης μιας activity. Σε αντίθετη περίπτωση οι συγκεκριμένοι παράμετροι μπορούν να προσεγγισθούν με μία κατανομή (π.χ., εκθετική). Για να βρεθεί η βέλτιστη λύση στο πρόβλημα της ανάθεσης όλων αυτών των παραμέτρων πρέπει να ερευνηθούν όλες οι δυνατές αναθέσεις, επειδή όμως αυτό είναι πολύ ακριβό, οι συγγραφείς χρησιμοποιούν μια «άπληστη» ευρετική προσέγγιση.

Στη δημοσίευση [ZBD+03] προτείνεται ένα μοντέλο για την ποιότητα μιας υπηρεσίας διαδικτύου και μία προσέγγιση για την επιλογή απλών υπηρεσιών διαδικτύου που να συνθέτουν μία σύνθετη η οποία βασίζεται σε αυτό το μοντέλο. Για την εκτέλεση μιας εργασίας της σύνθετης υπηρεσίας επιλέγεται μία απλή υπηρεσία διαδικτύου από ένα σύνολο απλών υπηρεσιών διαδικτύου οι οποίες έχουν την ίδια λειτουργικότητα αλλά δεν έχουν την ίδια ποιότητα υπηρεσίας. Οι συγγραφείς περιγράφουν μία σύνθετη υπηρεσία διαδικτύου με ένα statechart. Ένα statechart αποτελείται από καταστάσεις και μεταβάσεις. Οι μεταβάσεις προσδιορίζουν τη ροή δεδομένων και ελέγχου της σύνθετης υπηρεσίας. Οι καταστάσεις προσδιορίζουν τις λειτουργίες της σύνθετης υπηρεσίας και διακρίνονται σε βασικές και σε σύνθετες. Μία βασική κατάσταση

μπορεί να είναι είτε μία απλή υπηρεσία διαδικτύου, είτε μία σύνθετη υπηρεσία διαδικτύου, είτε μια κοινότητα υπηρεσιών διαδικτύου. Η έννοια της κοινότητας βασίζεται στην ιδέα μιας μεγάλης συλλογής από υπηρεσίες διαδικτύου οι οποίες έχουν την ίδια λειτουργικότητα αλλά έχουν διαφορετικές μη-λειτουργικές απαιτήσεις. Όταν μία κοινότητα δέχεται μία αίτηση για την εκτέλεση μιας λειτουργίας, η κοινότητα επιλέγει μια από της υπηρεσίες της που πληροί της ποιοτικές προϋποθέσεις για να εκτελέσει την αίτηση. Μία σύνθετη κατάσταση περιέχει ένα statechart, ή περισσότερα του ενός statecharts, τα οποία, είτε εκτελούνται παράλληλα, είτε εκτελείται ένα από αυτά. Επειδή σε ένα statechart υπάρχουν διακλαδώσεις, υπάρχουν πολλοί τρόποι για να εκτελεστεί. Κάθε διαφορετικός τρόπος εκτέλεσης αποτελεί και ένα διαφορετικό μονοπάτι εκτέλεσης. Με την προϋπόθεση ότι στο statechart δεν υπάρχουν κύκλοι τότε ένα μονοπάτι εκτέλεσης είναι ένας άκυκλος κατευθυνόμενος γράφος με κόμβους τις εργασίες του statechart και ακμές τις μεταβάσεις του statechart. Αν στις λειτουργίες ενός μονοπατιού εκτέλεσης ανατεθούν απλές υπηρεσίες διαδικτύου προκύπτει ένα πλάνο εκτέλεσης το οποίο μπορεί να αποτιμηθεί με βάση το μοντέλο ποιότητας που προτείνεται. Στο μοντέλο αυτό, τόσο οι απλές όσο και οι σύνθετες υπηρεσίες διαδικτύου προσδιορίζονται από τα εξής κριτήρια ποιότητας: κόστος εκτέλεσης, διάρκεια εκτέλεσης, αξιοπιστία, διαθεσιμότητα και υπόληψη. Όποιο πλάνο εκτέλεσης έχει καλύτερη ποιότητα με βάση τα παραπάνω κριτήρια είναι βέλτιστο. Επειδή η παραγωγή όλων των πλάνων εκτέλεσης έχει μεγάλη πολυπλοκότητα, οι συγγραφείς προτείνουν έναν αλγόριθμο γραμμικού προγραμματισμού. Ο αλγόριθμος αυτός δέχεται είσοδο μία γραμμική συνάρτηση η οποία εκφράζει την ποιότητα της ζητούμενης σύνθετης υπηρεσίας διαδικτύου μαζί με τους περιορισμούς, ελαχιστοποιεί αυτή τη συνάρτηση και δίνει έξοδο τις αναθέσεις απλών υπηρεσιών διαδικτύου που πρέπει να γίνουν σε κάθε λειτουργία της ζητούμενης σύνθετης υπηρεσίας για να λάβουμε ένα βέλτιστο πλάνο.

2.4. Διαχείριση Δυναμικών Σύνθετων Υπηρεσιών σε Δυναμικά Δίκτυα

Μια σύνθετη υπηρεσία (workflow) αποτελείται από επιμέρους απλές υπηρεσίες ή αλλιώς εργασίες που συνεργάζονται μεταξύ τους για να διεκπεραιώσουν τη λειτουργία της σύνθετης υπηρεσίας. Οι εργασίες μιας σύνθετης υπηρεσίας μπορεί να

είναι τοποθετημένες σε διαφορετικούς κόμβους ενός δικτύου. Υπάρχει περίπτωση το υπολογιστικό κόστος των εργασιών, τα μηνύματα που ανταλλάσσουν μεταξύ τους και ο χρόνος επικοινωνίας μεταξύ των κόμβων να μεταβάλλονται με το χρόνο, πρόκειται δηλαδή για ένα δυναμικό σύστημα.

Στο άρθρο [SaZh04] προτείνεται μία τακτική για την ανακατανομή των εργασιών στους κόμβους του δικτύου ώστε να λαμβάνονται υπόψη αυτές οι μεταβολές του συστήματος και να εκτελείται γρηγορότερα η σύνθετη υπηρεσία. Αρχικά, θεωρείται ότι οι εργασίες της σύνθετης υπηρεσίας κατανέμονται στο δίκτυο σύμφωνα με εκτιμήσεις για τις τιμές του συστήματος και με τη χρήση κάποιου αλγορίθμου χρονοπρογραμματισμού ο οποίος ορίζει για κάθε εργασία τον κόμβο που πρέπει να εκτελεστεί, το χρόνο εκκίνησης και το χρόνο τερματισμού. Στη συνέχεια, κατά την εκτέλεση της σύνθετης υπηρεσίας, ξανακαλείται ο αλγόριθμος, όταν είναι απαραίτητο, για ανακατανομή των εργασιών. Το ερώτημα είναι αν μετά από μια αλλαγή στις τιμές του συστήματος πρέπει να ξανά-εκτελεστεί ο αλγόριθμος χρονοπρογραμματισμού. Οι συγγραφείς προτείνουν μία μέθοδο ώστε να εκτελείται ο αλγόριθμος μόνο όταν οι αλλαγές στις τιμές του συστήματος είναι σημαντικές ώστε να χρειάζεται ανακατανομή. Η ιδέα είναι να υπολογίζεται ο χρόνος που μπορεί να καθυστερήσει μία εργασία ή ένα μήνυμα χωρίς να επηρεάζεται ο συνολικός χρόνος εκτέλεσης του workflow και όταν μία εργασία ή ένα μήνυμα καθυστερεί περισσότερο από αυτό το χρόνο τότε μόνο πρέπει να γίνεται εκτέλεση του αλγορίθμου χρονοπρογραμματισμού. Μετά από μία ανακατανομή των εργασιών το workflow συνεχίζει να εκτελείται από την εργασία από όπου σταμάτησε σύμφωνα με τους καινούριους χρόνους που μπορούν να καθυστερήσουν οι εργασίες και τα μηνύματα που υπολογίζονται από το καινούριο χρονοπρόγραμμα.

2.5. Εκτέλεση Σύνθετων Υπηρεσιών με Κατανεμημένο Τρόπο

Συνήθως η εκτέλεση μιας σύνθετης υπηρεσίας ή αλλιώς ενός workflow γίνεται με κεντρικοποιημένο τρόπο είτε στην πλευρά του χρήστη, ο οποίος εκτελεί μία-μία τις υπηρεσίες του workflow, είτε σε έναν server που δρα σαν workflow execution engine, ο οποίος αναλαμβάνει εξολοκλήρου την εκτέλεση του workflow. Και στις δυο αυτές

περιπτώσεις τα ενδιάμεσα αποτελέσματα που παράγονται από τις υπηρεσίες στέλνονται σε ένα κεντρικό σημείο (στον χρήστη ή στην workflow execution engine). Αυτές όμως οι προσεγγίσεις πάσχουν από το πρόβλημα του ενός κεντρικού σημείου αποτυχίας.

Στο άρθρο [CoBF05] παρουσιάζεται μια προσέγγιση για την εκτέλεση μιας σύνθετης υπηρεσίας διαδικτύου με κατανεμημένο τρόπο. Μία σύνθετη υπηρεσία διαδικτύου αναπαριστάται με ένα workflow. Στην προσέγγιση που προτείνεται, τα ενδιάμεσα αποτελέσματα που παράγονται από μία υπηρεσία διαδικτύου στέλνονται απευθείας στην αμέσως επόμενη στο workflow υπηρεσία διαδικτύου. Για την ακρίβεια στέλνονται σε ένα execution site το οποίο βρίσκεται «κοντά» (π.χ., γεωγραφικά) στην συγκεκριμένη υπηρεσία και είναι υπεύθυνο για τον έλεγχο της ροής δεδομένων αλλά και ελέγχου. Ένα execution site μπορεί να είναι υπεύθυνο για πολλές υπηρεσίες διαδικτύου. Κάθε υπηρεσία διαδικτύου για να κληθεί χρειάζεται έναν trigger στο execution site που είναι υπεύθυνο για αυτή. Ένας trigger κάνει τις εξής ενέργειες: συλλέγει όλες τις τιμές των εισόδων που χρειάζονται για την κλήση της υπηρεσίας και όταν όλες είναι διαθέσιμες καλεί την υπηρεσία. Στη συνέχεια είναι υπεύθυνος για να τροφοδοτήσει τον κατάλληλο trigger με τις τιμές της εξόδου που επέστρεψε η υπηρεσία. Το πρόβλημα που προκύπτει είναι το εξής: δοθέντος ενός workflow και ενός συνόλου διαθέσιμων execution sites, σε ποια execution sites πρέπει να τοποθετηθούν οι triggers των υπηρεσιών διαδικτύου του workflow, ώστε η συνάρτηση κόστους να ελαχιστοποιηθεί. Για τον ορισμό της συνάρτησης κόστους χρησιμοποιούνται τρεις παράμετροι. Το κόστος για την ενεργοποίηση των υπηρεσιών διαδικτύου, το κόστος για την εκτέλεση των υπηρεσιών διαδικτύου και το κόστος για τη μεταφορά των παραμέτρων. Η διαδικασία επίλυσης του προβλήματος έχει ως εξής: το αρχικό workflow στέλνεται σε όλα τα σχετικά execution sites και κάθε ένα από αυτά επιστρέφει όλες τις αναθέσεις των triggers στα execution sites, που μπορούν να γίνουν σύμφωνα με τη δική του προοπτική, μαζί με το κόστος της κάθε μιας ανάθεσης. Οι αναθέσεις και τα αντίστοιχα κόστη που προτείνουν όλα τα execution sites συλλέγονται και κατασκευάζεται ένα πρόβλημα βελτιστοποίησης με περιορισμούς που θέτει ο χρήστης. Με την επίλυση αυτού του προβλήματος προκύπτει η καλύτερη ανάθεση triggers σε execution sites.

2.6. Βελτιστοποίηση Ερωτήσεων για Σύνθετες Υπηρεσίες Διαδικτύου

Όταν ένα σύνολο από απλές υπηρεσίες διαδικτύου συνεργάζεται για να εκτελέσει μία εργασία, τότε αυτό το σύνολο αποτελεί μία σύνθετη υπηρεσία διαδικτύου. Όσον αφορά την ακολουθία με την οποία εκτελούνται οι απλές υπηρεσίες διαδικτύου υπάρχει περίπτωση να εκτελούνται με πολλούς τρόπους οι οποίοι δίνουν βέβαια το ίδιο αποτέλεσμα αλλά σε διαφορετικό χρόνο.

Στο άρθρο [SWMM05] παρουσιάζεται ένας αλγόριθμος για την βελτιστοποίηση ερωτήσεων για σύνθετες υπηρεσίες διαδικτύου ώστε να ελαχιστοποιείται ο χρόνος εκτέλεσης τους. Η βελτιστοποίηση γίνεται από ένα κεντρικό σύστημα διαχείρισης το οποίο είναι υπεύθυνο και για τον συντονισμό της ροής της εκτέλεσης. Το σύστημα αυτό χειρίζεται μια σύνθετη υπηρεσία διαδικτύου ως ένα workflow ή ως μία σωλήνωση (pipeline). Η πρώτη βελτιστοποίηση που γίνεται είναι η παράλληλη εκτέλεση όλων των υπηρεσιών του workflow, με την τροφοδότηση των ήδη επεξεργασμένων δεδομένων από μια υπηρεσία στην επόμενή της χωρίς να έχει τελειώσει ακόμα με το σύνολο των δεδομένων. Η δεύτερη βελτιστοποίηση έχει να κάνει με την εύρεση της καταλληλότερης σειράς που πρέπει να εκτελεστούν οι υπηρεσίες διαδικτύου χωρίς να παραβιάζονται περιορισμοί προτεραιότητας. Η βασική ιδέα της είναι ότι η ελαχιστοποίηση του χρόνου εκτέλεσης μιας σωλήνωσης ισοδυναμεί με την μεγιστοποίηση του ρυθμού που επεξεργάζονται τα δεδομένα της εισόδου, στην φάση όπου όλες οι υπηρεσίες διαδικτύου της σωλήνωσης επεξεργάζονται δεδομένα. Σε μία σωλήνωση, αυτόν τον ρυθμό καθορίζει η πιο αργή υπηρεσία διαδικτύου . Έτσι είναι προφανές ότι στην πιο αργή υπηρεσία πρέπει να τροφοδοτηθούν όσο γίνεται λιγότερα δεδομένα. Γενικεύοντας το παραπάνω, όσο αργή είναι μια υπηρεσία τόσο λιγότερα δεδομένα πρέπει να επεξεργάζεται. Στο σημείο αυτό διακρίνονται δύο κατηγορίες υπηρεσιών διαδικτύου: υπηρεσίες διαδικτύου με επιλεκτικότητα μικρότερη ή ίση του ένα (δηλαδή παράγουν λιγότερα ή ίσα δεδομένα από ότι επεξεργάζονται) και υπηρεσίες διαδικτύου με επιλεκτικότητα μεγαλύτερη του ένα (δηλαδή παράγουν περισσότερα δεδομένα από αυτά που επεξεργάζονται). Για υπηρεσίες διαδικτύου που ανήκουν στην πρώτη κατηγορία, ο

βέλτιστος τρόπος για να εκτελεστούν είναι ακολουθιακά αρχίζοντας από την πιο γρήγορη και προχωρώντας προς την πιο αργή, με την προϋπόθεση ότι ικανοποιούνται οι περιορισμοί προτεραιότητας. Για υπηρεσίες διαδικτύου που ανήκουν στην δεύτερη κατηγορία ο βέλτιστος τρόπος για να εκτελεστούν είναι να εκτελεστούν όλες μαζί συγχρόνως (παράλληλα) και τα αποτελέσματα να συνενωθούν. Αν υπάρχουν περιορισμοί προτεραιότητας, τότε δεν εκτελούνται όλες οι υπηρεσίες συγχρόνως αλλά προκύπτει ένα δενδροειδές πλάνο. Όταν το workflow αποτελείται από υπηρεσίες διαδικτύου και των δύο κατηγοριών τότε στο βέλτιστο πλάνο εκτέλεσης αρχικά εκτελούνται οι υπηρεσίες διαδικτύου με επιλεκτικότητα μικρότερη ή ίση του ένα και στη συνέχεια οι υπηρεσίες διαδικτύου με επιλεκτικότητα μεγαλύτερη του ένα, με τον τρόπο που περιγράφτηκε παραπάνω. Όμως υπάρχει περίπτωση, υπηρεσίες με επιλεκτικότητα μικρότερη ή ίση του ένα να πρέπει να εκτελεστούν μετά από την εκτέλεση υπηρεσιών με επιλεκτικότητα μεγαλύτερη του ένα, λόγω περιορισμών προτεραιότητας. Κάτι τέτοιο όμως απαγορεύει τη χρήση του βέλτιστου πλάνου και απαιτείται κάτι ακόμα. Απαιτείται ομαδοποίηση υπηρεσιών διαδικτύου, ώστε στην αρχή του workflow να υπάρχουν μόνο υπηρεσίες διαδικτύου με επιλεκτικότητα μικρότερη ίση του ένα και να ακολουθούν υπηρεσίες διαδικτύου με επιλεκτικότητα μεγαλύτερη από ένα. Στο άρθρο υπολογίζεται επίσης και το βέλτιστο μέγεθος δεδομένων (chunk size) που πρέπει να στέλνει μία υπηρεσία διαδικτύου στην επόμενή της στο workflow.

2.7. Τοποθέτηση Αντικειμένων σε Απομακρυσμένη Μνήμη

Στην ενότητα αυτή θα περιγραφεί ένα πρόβλημα που δεν εντάσσεται στην ίδια κατηγορία με το πρόβλημα της εργασίας, παρόλο αυτά τα δύο προβλήματα έχουν κοινά σημεία. Απομακρυσμένη μνήμη είναι ένα νέο είδος μνήμης που χρησιμοποιείται σε κατανεμημένα συστήματα και ορίζεται ως η μνήμη ενός κόμβου του κατανεμημένου συστήματος που χρησιμοποιείται από άλλους κόμβους και προσπελαύνεται μέσω του δικτύου. Η απομακρυσμένη μνήμη είναι πιο αργή από την τοπική μνήμη αλλά μπορεί να είναι πιο γρήγορη από το δίσκο. Η αναζήτηση ενός αντικειμένου από έναν κόμβο γίνεται πρώτα στην τοπική μνήμη μετά στην απομακρυσμένη και μετά στο δίσκο. Το πρόβλημα που παρουσιάζεται στην

αρχιτεκτονική αυτή είναι το εξής: ποια αντικείμενα πρέπει να τοποθετηθούν και σε ποιο κόμβο (τοπική μνήμη του κόμβου) του συστήματος ώστε να αξιοποιείται στο μέγιστο η αρχιτεκτονική αυτή και να μεγιστοποιείται το κέρδος. Το κέρδος όμως είναι μία σχετική ποσότητα και υπάρχουν πολλοί τρόποι για να ορισθεί. Παρομοίως, το πρόβλημα της εργασίας έχει να κάνει με την τοποθέτηση αντικειμένων (υπηρεσιών διαδικτύου) σε ένα δίκτυο εξυπηρετητών όπου το κέρδος μπορεί να ορισθεί με πολλούς τρόπους.

Στο άρθρο [LeWY93] μελετάται το πρόβλημα τοποθέτησης αντικειμένων σε αρχιτεκτονική που χρησιμοποιεί απομακρυσμένη μνήμη. Κατά την διαστασιοποίηση του προβλήματος διευκρινίζονται οι διαφορετικοί τρόποι με τους οποίους μπορεί να ορισθεί το κέρδος μιας τοποθέτησης. Από τη μία, μπορεί ο στόχος να είναι η επίδοση του συστήματος και από την άλλη η δικαιοσύνη. Στην πρώτη περίπτωση μπορεί οι κόμβοι είτε να ενδιαφέρονται μόνο για την δική τους επίδοση ξεχωριστά (local performance), είτε να ενδιαφέρονται για την επίδοση του συστήματος συνολικά (global performance). Στην δεύτερη περίπτωση ενδιαφερόμαστε όλοι οι κόμβοι να έχουν περίπου την ίδια επίδοση. Για τους διαφορετικούς αυτούς ορισμούς του κέρδους υπάρχουν τρεις στρατηγικές για το ποια αντικείμενα πρέπει να αποθηκεύει ένας κόμβος: κεντρικοποιημένες κατανεμημένες και απομονωμένες. Στις κεντρικοποιημένες στρατηγικές υπάρχει κάποιος κόμβος που ελέγχει όλους τους κόμβους και έτσι μπορεί να υπολογίσει την βέλτιστη λύση ως προς τη δικαιοσύνη ή ως προς την καθολική απόδοση (global performance). Οι στρατηγικές αυτές δεν μπορούν όμως να εφαρμοστούν γιατί απαιτείται πολύ μεγάλο bandwidth και χρησιμοποιούνται μόνο για την εύρεση του πάνω ορίου στο κέρδος. Στις απομονωμένες στρατηγικές, κάθε κόμβος αποθηκεύει στη τοπική μνήμη του τα αντικείμενα που προσπελαύνει ο ίδιος συχνότερα αδιαφορώντας για το ποια αντικείμενα είναι αποθηκευμένα στους άλλους κόμβους. Με τον τρόπο αυτό επιτυγχάνεται τοπικά βέλτιστη λύση στον κάθε ένα κόμβο ξεχωριστά. Στις κατανεμημένες στρατηγικές οι κόμβοι δεν γνωρίζουν τα αντικείμενα όλων των κόμβων αλλά τα αντικείμενα που βρίσκονται σε μια μερίδα των κόμβων. Έτσι το πρόβλημα χωρίζεται σε υπό-προβλήματα, το καθένα από αυτά βελτιστοποιείται και η συνένωση τους ελπίζουμε να μας δώσει μία λύση κοντά στην βέλτιστη.

Μία κατανεμημένη στρατηγική παρουσιάζεται στο άρθρο [LTZS05]. Ο αλγόριθμος που προτείνεται αρχικά δίνει τοπικά βέλτιστη λύση και στη συνέχεια βελτιώνει αυτή τη λύση ως προς το καθολικό κέρδος χωρίς όμως να μειώνεται η απόδοση κανενός κόμβου. Ο χώρος αυτός για βελτίωση της τοπικά βέλτιστης λύσης ώστε να βελτιωθεί και το καθολικό κέρδος υπάρχει για τον εξής λόγο: όταν όλοι οι κόμβοι αποθηκεύουν τα ίδια αντικείμενα στην τοπική τους μνήμη τότε σπαταλούν όλοι την τοπική τους μνήμη για τα ίδια αντικείμενα. Αυτή τη μνήμη θα μπορούσαν να την αξιοποιήσουν αποτελεσματικότερα αν κόμβοι επικοινωνούσαν μεταξύ τους, βελτιώνοντας όλοι την απόδοσή τους, βελτιώνοντας παράλληλα και την καθολική απόδοση του συστήματος.

2.8. Ανακεφαλαίωση

Στο κεφάλαιο αυτό είδαμε εργασίες σχετικές με τη μεταπτυχιακή εργασία. Οι εργασίες αυτές έχουν δύο κύριες συνιστώσες, τις σύνθετες υπηρεσίες και την ποιότητα υπηρεσίας. Όσον αφορά τις σύνθετες υπηρεσίες, το ενδιαφέρον βρίσκεται στην κατανομή της διαχείρισης και της εκτέλεσης τους και επίσης στην κατανομή της ανακάλυψής τους όταν πρόκειται για υπηρεσίες διαδικτύου.

Τα άρθρα που ασχολούνται με την κατανομή της ανακάλυψης υπηρεσιών διαδικτύου χρησιμοποιούν την σημασιολογία των υπηρεσιών διαδικτύου. Στο άρθρο [DHM+04] προτείνεται ένας αλγόριθμος για ομαδοποίηση υπηρεσιών διαδικτύου που χειρίζεται σημασιολογικές «έννοιες» οι οποίες προκύπτουν από την ομαδοποίηση των ονομάτων, των παραμέτρων και των περιγραφών των υπηρεσιών διαδικτύου. Στο άρθρο [DoZV05], η ανακάλυψη υπηρεσιών διαδικτύου γίνεται με τη βοήθεια ενός γράφου ο οποίος κατηγοριοποιεί τις υπηρεσίες και τις ερωτήσεις με βάση το context τους. Στην ενότητα της ανακάλυψης υπηρεσιών διαδικτύου εντάσσονται άλλα τρία άρθρα που χρησιμοποιούν το σύστημα Chord για την κατανομή των υπηρεσιών. Στο άρθρο [ESAA04], για να βρεθεί η θέση μιας υπηρεσίας στο δακτύλιο του Chord, χρησιμοποιείται ως κλειδί κατακερματισμού το πεπερασμένο αυτόματο της υπηρεσίας. Αντίθετα, στα άρθρα [LiZh05] και [ScPa04] χρησιμοποιούνται ως κλειδί κατακερματισμού οι λέξεις κλειδιά των υπηρεσιών. Πιο συγκεκριμένα, στο πρώτο

άρθρο, το κλειδί προκύπτει κατακερματίζοντας τις λέξεις κλειδιά της υπηρεσίας με SHA-1 συναρτήσεις κατακερματισμού. Εκτός όμως από το δίκτυο Chord, οι υπηρεσίες σχηματίζουν και ένα σημασιολογικό δίκτυο. Οι συνδέσεις σε αυτό το δίκτυο δημιουργούνται κατά την εισαγωγή τους στο δίκτυο και συνδέουν υπηρεσίες με παρόμοια λειτουργικότητα, κατηγορία και λέξεις κλειδιά. Στο δεύτερο άρθρο, οι λέξεις κλειδιά μιας υπηρεσίας θεωρούνται ένα σημείο στον πολυδιάστατο χώρο. Από τον πολυδιάστατο χώρο το σημείο αυτό απεικονίζεται στον μονοδιάστατο χώρο με τη χρήση μιας SFC καμπύλης και από αυτήν την καμπύλη απεικονίζονται στη συνέχεια στο δακτύλιο του Chord.

Τα άρθρα που αναφέρθηκαν και ασχολούνται με την ανακάλυψη υπηρεσιών διαδικτύου είναι σχετικά με το πρόβλημα της εργασίας γιατί προτείνουν ομαδοποιήσεις των λειτουργιών. Αυτό που λείπει όμως από τα άρθρα αυτά είναι ότι τα πρώτα δύο δεν λένε πως πρέπει να κατανεμηθούν οι ομάδες των λειτουργιών που προκύπτουν, ενώ τα τρία τελευταία κατανέμουν τις περιγραφές των λειτουργιών σε συστήματα Chord κι όχι τις ίδιες τις λειτουργίες.

Σημαντικό θέμα στις προσφερόμενες υπηρεσίες είναι η ποιότητα τους και ο τρόπος που μπορούμε να διαχειριστούμε και να διασφαλίσουμε την ποιότητα. Στις εργασίες που αναφέρθηκαν έχουν προταθεί πολλά κριτήρια αξιολόγησης της ποιότητας μιας σύνθετης υπηρεσίας. Τα κυριότερα είναι ο χρόνος απόκρισης, η ρυθμοαπόδοση, το κόστος και η διαθεσιμότητα. Ο χρόνος απόκρισης είναι ο χρόνος που περιμένει ο χρήστης για την εκτέλεση της υπηρεσίας. Η ρυθμοαπόδοση είναι ο ρυθμός που το σύστημα εκτελεί workflows. Το κόστος μπορεί να συμπεριλαμβάνει πολλά κόστη, όπως το κόστος για τα υλικά του συστήματος και το κόστος για τη διαχείριση και την εκτέλεση του συστήματος. Η διαθεσιμότητα είναι το ποσοστό του χρόνου που το σύστημα λειτουργεί κανονικά και τα workflows μπορούν να εκτελεστούν.

Στα άρθρα [CSMA+04], [GiWW02], [ZBD+03] και [SaZh04] οι συγγραφείς ασχολούνται με την διαχείριση σύνθετων υπηρεσιών διαδικτύου ώστε να διασφαλίζεται η ποιότητα της σύνθετης υπηρεσίας (workflow). Στο πρώτο η ποιότητα του workflow υπολογίζεται με έναν αλγόριθμο που εφαρμόζει

επαναληπτικά ένα σύνολο από κανόνες μείωσης του workflow, οι οποίοι αλλάζουν τη δομή του έως ότου να μπορεί να αναπαρασταθεί με μία απλή εργασία. Στο δεύτερο η ποιότητα υπηρεσίας ενός workflow υπολογίζεται με τη βοήθεια μοντέλων Markov, και με βάση αυτά, υπολογίζεται πόσοι εξυπηρετητές και τι τύπου (workflow severs, application servers, communication servers) πρέπει να χρησιμοποιηθούν. Στο τρίτο άρθρο, προτείνεται μία προσέγγιση όπου, δοθείσας την ποιότητα της σύνθετης υπηρεσίας που θέλουμε, επιλέγονται οι καταλληλότερες απλές υπηρεσίες για να συνθέσουν το workflow. Η επιλογή απλών υπηρεσιών για μία εργασία του workflow γίνεται μεταξύ υπηρεσιών με την ίδια λειτουργικότητα. Στο τέταρτο άρθρο παρουσιάζεται μία απλή μέθοδος για την παρακολούθηση των αλλαγών των παραμέτρων του συστήματος η οποία αποφασίζει πότε οι αλλαγές αυτές επηρεάζουν το συνολικό χρόνο εκτέλεσης ενός workflow. Το πρώτο και το τέταρτο άρθρο προτείνουν λύσεις για να ελέγχεται η ποιότητα υπηρεσίας ενός workflow, αλλά δεν προτείνουν λύση για το πώς πρέπει να κατανεμηθεί ένα workflow. Το δεύτερο άρθρο χρησιμοποιεί ένα μοντέλο που καλύπτει πολλές παραμέτρους του προβλήματός μας, προσεγγίζει όμως το πρόβλημα διαφορετικά, αφού «φτιάχνει» αντίγραφα εξυπηρετητών. Το τρίτο άρθρο μοιάζει με το πρόβλημα της εργασίας μας, αλλά θεωρεί κοινότητες υπηρεσιών με ίδια λειτουργικότητα και δεν εξετάζει την κατανομή του φορτίου.

Τα άρθρα [CoBF05] και [SWMM05] ασχολούνται με την εκτέλεση σύνθετων υπηρεσιών διαδικτύου. Στο πρώτο από αυτά προτείνεται μία προσέγγιση για κατανομή της εκτέλεσης με την βοήθεια triggers οι οποίοι τοποθετούνται στους κόμβους και συλλέγουν τις εισόδους για τις υπηρεσίες που είναι υπεύθυνοι και καλούν οι ίδιοι τις υπηρεσίες. Το πρόβλημα που επιλύεται σε αυτό το άρθρο είναι: δοθείσας μιας συνάρτησης κόστους, ποια είναι η καλύτερη ανάθεση των triggers στους κόμβους. Η προσέγγιση αυτή διαφέρει από τη δική μας γιατί θεωρεί τις υπηρεσίες σταθερές στους κόμβους που βρίσκονται. Αντιθέτως στη δική μας προσέγγιση, οι λειτουργίες μπορούν να τοποθετηθούν σε οποιονδήποτε κόμβο. Στο δεύτερο άρθρο συζητείται με ποια σειρά πρέπει να εκτελεστούν οι υπηρεσίες διαδικτύου ενός workflow ώστε να βελτιστοποιηθεί η ρυθμοαπόδοση της εκτέλεσης του. Οι συγγραφείς κάνουν την ισχυρή υπόθεση ότι το workflow εκτελείται

συνεχόμενα, δηλαδή, υπάρχουν πάντα δεδομένα εισόδου στην αρχή του workflow. Η προσέγγιση που προτείνεται μπορεί να εφαρμοστεί στο δικό μας πρόβλημα αν στο μοντέλο μας προσθέσουμε τη ρυθμοαπόδοση του workflow και κάνοντας επίσης την υπόθεση ότι το workflow εκτελείται συνεχόμενα.

Τέλος τα άρθρα [LeWY93] και [LTZS05] πραγματεύονται ένα πρόβλημα που μοιάζει πολύ με το πρόβλημα της μεταπτυχιακής εργασίας, την τοποθέτηση αντικειμένων σε ένα είδος μνήμης, την απομακρυσμένη μνήμη. Το ενδιαφέρον σε αυτά τα άρθρα είναι ο τρόπος που γίνεται η διαστασιοποίηση του προβλήματος και οι διαφορετικοί τρόποι που μπορεί να ορισθεί το κέρδος μιας τοποθέτησης.

ΚΕΦΑΛΑΙΟ 3. ΓΕΝΙΚΟΣ ΟΡΙΣΜΟΣ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ

3.1 Ορισμός του Workflow
3.2 Παράδειγμα
3.3 Διαστασιοποίηση του Προβλήματος
3.4 Μοντελοποίηση του Προβλήματος
3.5 Ιδιότητες Μιας Καλής Λύσης του Προβλήματος
3.6 Συμβολισμοί, Τύποι και Φόρμουλες

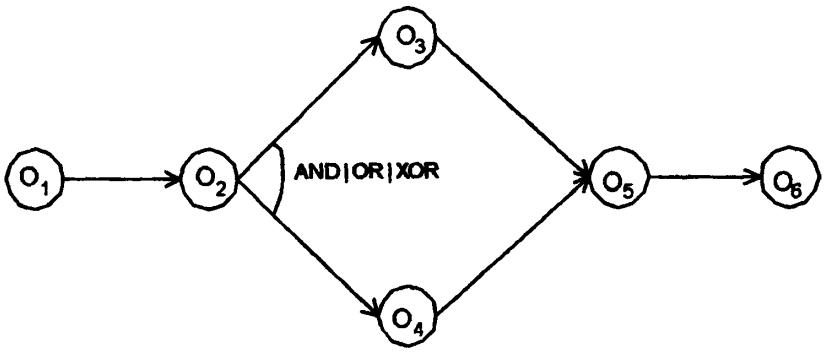
Στο συγκεκριμένο κεφάλαιο, αρχικά θα δούμε τι είναι ένα workflow και θα παρουσιάσουμε ένα παράδειγμα ενός πραγματικού workflow. Στη συνέχεια, ορίζουμε το πρόβλημα δίνοντας το μοντέλο που χρησιμοποιούμε και αναλύοντας τις διαφορετικές διαστάσεις του προβλήματος. Έπειτα, περιγράφουμε τις ιδιότητες που πρέπει να έχει μία λύση του προβλήματος ώστε να είναι καλή και τέλος δίνουμε τους συμβολισμούς, τους τύπους και τις φόρμουλες που χρησιμοποιούμε.

3.1. Ορισμός του Workflow

Ένα workflow, ή αλλιώς μια ροή εργασιών, είναι ένας τυπικός τρόπος για την περιγραφή της εκτέλεσης μιας σύνθετης εργασίας. Πιο συγκεκριμένα είναι ένας γράφος με κόμβους τις εργασίες της σύνθετης εργασίας και ακμές τα μηνύματα που στέλνουν οι εργασίες μεταξύ τους. Τα μηνύματα αυτά αποτελούν τα δεδομένα που

χρειάζονται για την εκτέλεση της σύνθετης εργασίας αλλά και για τον έλεγχο της ροής της εκτέλεσης. Στην εργασία αυτή ασχολούμαστε με workflows υπηρεσιών διαδικτύου. Αυτό σημαίνει ότι οι κόμβοι του workflow είναι λειτουργίες υπηρεσιών διαδικτύου και οι ακμές είναι XML μηνύματα. Οι λειτουργίες συνεργάζονται μεταξύ τους και όλες μαζί αποτελούν μία σύνθετη υπηρεσία διαδικτύου.

Στη περίπτωση που στο workflow υπάρχουν διακλαδώσεις, οι λειτουργίες του workflow διακρίνονται σε δύο κατηγορίες, στις *conditional* και στις *operational*. Οι πρώτες είναι αυτές που ελέγχουν τη ροή της διαδικασίας ενώ οι δεύτερες είναι αυτές που εκτελούν τις κύριες λειτουργίες του workflow. Οι conditional λειτουργίες χρειάζονται πολύ λιγότερη υπολογιστική ισχύ για να εκτελεστούν από ότι οι operational λειτουργίες. Κάθε διακλάδωση αρχίζει από μία conditional λειτουργία και τελειώνει επίσης σε μία conditional λειτουργία, ενώ μπορούν να υπάρχουν τρεις διαφορετικοί τύποι διακλαδώσεων. Στον πρώτο τύπο διακλάδωσης (τύπος and) εκτελούνται όλοι οι κλάδοι και για να συνεχιστεί η εκτέλεση του workflow πρέπει να έχουν τελειώσει την εκτέλεσή τους όλοι οι κλάδοι. Στον δεύτερο τύπο διακλάδωσης (τύπος or) εκτελούνται επίσης όλοι οι κλάδοι αλλά για τη συνέχιση της εκτέλεσης του workflow αρκεί να τελειώσει την εκτέλεσή του ένας μόνο κλάδος. Στον τρίτο τύπο διακλάδωσης (τύπος xor) επιλέγεται μόνο ένας κλάδος να εκτελεστεί. Στο Σχήμα 3.1, βλέπουμε ένα workflow με έξι λειτουργίες. Οι λειτουργίες O_2 και O_5 είναι conditional, ενώ οι λειτουργίες O_1, O_3, O_4 και O_6 είναι operational.



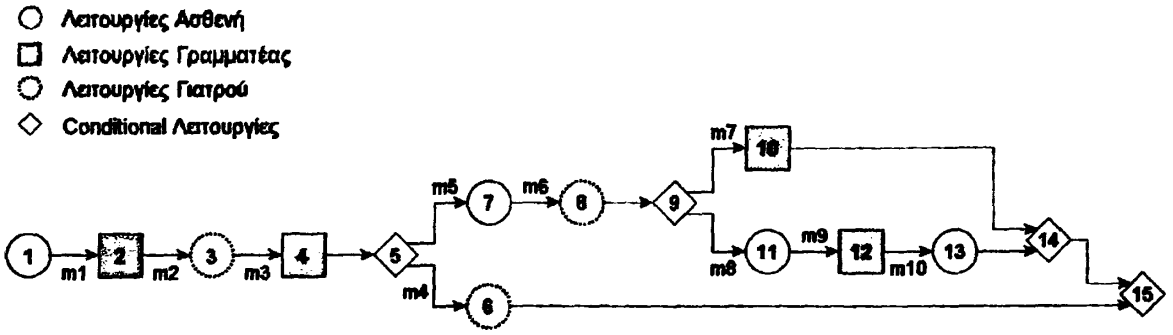
Σχήμα 3.1 Παράδειγμα Ενός Workflow. Ο Τύπος της Διακλάδωση Μπορεί να Είναι Ένας από τους Τρεις που Απεικονίζονται.

3.2. Παράδειγμα

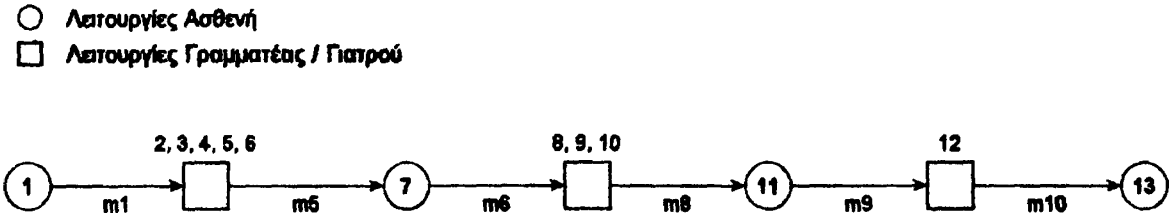
Στην ενότητα αυτή, περιγράφεται ένα παράδειγμα πραγματικού workflow, όπου μπορεί να ορισθεί και να λυθεί το πρόβλημα της ομαδοποίησης των υπηρεσιών διαδικτύου. Αυτό θα έχει νόημα αν για παράδειγμα όλες οι υπηρεσίες διαδικτύου του workflow εκτελούνται σε έναν δικτυακό τόπο που διαθέτει ένας πλήθος από εξυπηρετητές. Το πρόβλημα τότε είναι πώς πρέπει να κατανεμηθούν οι υπηρεσίες διαδικτύου του workflow στους εξυπηρετητές του δικτυακού τόπου, ώστε να ικανοποιούνται τα κριτήρια που θέτει ο διαχειριστής του συστήματος.

Το παράδειγμα αφορά την διαδικασία της επικοινωνίας ενός ασθενή με το γιατρό του [OMG06]. Το workflow του παραδείγματος αυτού φαίνεται στο Σχήμα 3.2. Οι operational λειτουργίες του workflow χωρίζονται σε τρεις κατηγορίες ανάλογα με το ποιος τις χρησιμοποιεί. Έτσι υπάρχουν λειτουργίες που χρησιμοποιούνται από τον ασθενή, λειτουργίες που χρησιμοποιούνται από τον γιατρό και λειτουργίες που χρησιμοποιούνται από τη γραμματέα του γιατρού. Οι conditional λειτουργίες του workflow (5 και 9) είναι τύπου and και έτσι εκτελούνται και οι δύο κλάδοι που ξεκινάνε από την κάθε μία. Η λειτουργικότητα των λειτουργιών του workflow καθώς και τα μηνύματα που ανταλλάσσουν μεταξύ τους φαίνονται στους πίνακες 3.1 και 3.2 αντίστοιχα. Η λειτουργία 4 στέλνει στην λειτουργία 5 τα μηνύματα m_4 και m_5, και η λειτουργία 8 στέλνει στην λειτουργία 9 τα μηνύματα m_7 και m_8. Επίσης τα μηνύματα που στέλνονται στις conditional λειτουργίες 14 και 15 είναι μικρά μηνύματα που επιβεβαιώνουν την περάτωση των operational λειτουργιών που τα στέλνουν.

Αν θεωρήσουμε ότι οι λειτουργίες του γιατρού και της γραμματέως είναι κοινές και τις χρησιμοποιεί ένας από τους δύο, τότε το workflow του Σχήματος 3.2 απλοποιείται και προκύπτει το workflow του Σχήματος 3.3. Σε αυτή την περίπτωση οι λειτουργίες 2 έως 6 συνενώνονται σε μία λειτουργία η οποία έχει την ίδια λειτουργικότητα με αυτή που είχαν συνολικά οι λειτουργίες από 2 έως 6. Το ίδιο ισχύει και για τις λειτουργίες 8, 9 και 10, οι οποίες στο καινούριο workflow αποτελούν μία λειτουργία.



Σχήμα 3.2 Παράδειγμα Workflow με Διακλαδώσεις.



Σχήμα 3.3 Παράδειγμα Workflow χωρίς Διακλαδώσεις.

Αν υποθέσουμε τώρα ότι το workflow του Σχήματος 3.2 φιλοξενείται σε ένα δικτυακό τόπο, ο οποίος διαθέτει πέντε εξυπηρετητές, τότε, όλες οι δυνατές αναθέσεις θα ήταν 5^{15} (= 30.517.587.125). Για να βρούμε ποια ανάθεση είναι καλύτερη, πρέπει να ελέγξουμε όλες αυτές τις αναθέσεις και να επιλέξουμε αυτή που ικανοποιεί τις απαιτήσεις μας. Όμως, ο υπολογισμός όλων αυτών των αναθέσεων είναι ακριβός και πρέπει να βρεθεί ένας τρόπος ώστε η ανάθεση στον δικτυακό τόπο να γίνεται γρήγορα. Η ανάγκη για μια γρήγορη ανάθεση των λειτουργιών στους κόμβους γίνεται πιο επιτακτική αν σκεφτούμε ότι ο δικτυακός τόπος μπορεί να φιλοξενεί χιλιάδες workflows και ότι κάθε workflow μπορεί να είναι δυναμικό, δηλαδή, μπορεί να αλλάζουν οι παράμετροί του και να απαιτείται καινούρια ανάθεση των λειτουργιών του στους εξυπηρετητές.

Με την κατανομή του workflow του Σχήματος 3.2 μπορούμε να πετύχουμε παραλληλισμό των λειτουργιών αλλά και κατανομή του κόστους λειτουργίας του. Για παράδειγμα, οι λειτουργίες που μπορούν να εκτελεστούν παράλληλα σε διαφορετικούς εξυπηρετητές είναι η 6 με την 7, οι οποίες αποτελούν λειτουργίες του

γιατρού και του ασθενή αντίστοιχα, όπως, και η 10 με την 11, οι οποίες αποτελούν λειτουργίες της γραμματέας και του ασθενή αντίστοιχα. Στις περιπτώσεις αυτές υπάρχουν περιορισμοί που απαγορεύουν την τοποθέτηση των λειτουργιών 6 και 7 στον ίδιο εξυπηρετητή, καθώς και την τοποθέτηση των λειτουργιών 10 και 11 στον ίδιο εξυπηρετητή.

Πίνακας 3.1 Λειτουργικότητα των Λειτουργιών του Workflow του Σχήματος 3.2 .

Λειτουργία	Λειτουργικότητα
1	Στέλνει μία αίτηση για εύρεση διαθέσιμου γιατρού.
2	Λαμβάνει μία αίτηση για εύρεση διαθέσιμου γιατρού και στη συνέχεια στέλνει μία αίτησης προς μία λίστα ιατρών για το αν είναι διαθέσιμοι.
3	Λαμβάνει μία αίτηση για διαθέσιμο γιατρό και στη συνέχεια στέλνει καταφατική ή αρνητική απάντηση.
4	Λαμβάνει απαντήσεις από τους γιατρούς και είναι υπεύθυνη για να επικοινωνήσει ο ασθενής με κάποιον γιατρό.
5	Πρόκειται για conditional λειτουργία που μπορεί να θεωρηθεί κομμάτι της λειτουργίας 4 και είναι υπεύθυνη για την αποστολή δύο μηνυμάτων προς ασθενή και γιατρό.
6	Λαμβάνει τα στοιχεία του πελάτη ώστε να γίνει η σύνδεση μεταξύ ασθενή και γιατρού.
7	Λαμβάνει τα στοιχεία του γιατρού που χρειάζονται για την επικοινωνία και στέλνει τα συμπτώματα του ασθενή στον γιατρό.
8	Λαμβάνει τα συμπτώματα από τον ασθενή και στέλνει την συνταγή στη γραμματέα για να την ετοιμάσει και στον ασθενή για να την παραλάβει.
9	Πρόκειται για conditional λειτουργία που μπορεί να θεωρηθεί κομμάτι της λειτουργίας 8 και είναι υπεύθυνη για την αποστολή δύο μηνυμάτων προς ασθενή και γραμματέα.
10	Λαμβάνει μία συνταγή γιατρού για να την προετοιμάσει η γραμματέας.
11	Λαμβάνει μία συνταγή γιατρού και στέλνει μία αίτηση για την αγορά των φαρμάκων της συνταγής.
12	Λαμβάνει μία αίτηση για αγορά φαρμάκων και επιστρέφει μία απάντηση για τον τρόπο που ο ασθενής θα τα λάβει.
13	Λαμβάνει μία απάντηση που περιγράφει πως ο ασθενής θα λάβει τα φάρμακά του.
14	Πρόκειται για conditional λειτουργία η οποία περιμένει να τελειώσουν οι δύο κλάδοι του workflow που αρχίζουν από την λειτουργία 9 ώστε να συνεχιστεί η ροή της διαδικασίας.
15	Πρόκειται για conditional λειτουργία η οποία περιμένει να τελειώσουν οι δύο κλάδοι του workflow που αρχίζουν από την λειτουργία 5 ώστε να συνεχιστεί η ροή της διαδικασίας, δηλαδή να ολοκληρωθεί η εκτέλεση του workflow.

Πίνακας 3.2 Περιγραφή των Μηνυμάτων του Workflow του Σχήματος 3.2 .

Μήνυμα	Περιγραφή	Διαδρομή Μηνύματος
m_1	«Θέλω να δω ένα γιατρό»	Ο ασθενής στέλνει στην γραμματέα.
m_2	«Είστε διαθέσιμος;»	Η γραμματέας στέλνει σε μία λίστα από γιατρούς.
m_3	«Είμαι διαθέσιμος»	Ο γιατρός στέλνει στη γραμματέα.
m_4	«Θα σας κλείσω ραντεβού»	Η γραμματέας στέλνει στο γιατρό.
m_5	«Θα σας κλείσω ραντεβού»	Η γραμματέας στέλνει στον ασθενή.
m_6	Συμπτώματα της ασθένειας	Ο ασθενής στέλνει στο γιατρό.
m_7	«Ετοίμασε αυτή τη συνταγή»	Ο γιατρός στέλνει στη γραμματέα.
m_8	«Ορίστε η συνταγής σας»	Ο γιατρός στέλνει στον ασθενή.
m_9	«Χρειάζομαι το φάρμακό μου»	Ο ασθενής στέλνει στη γραμματέα.
m_{10}	Περιέχει τον τρόπο που θα αποκτήσει το φάρμακό του.	Η γραμματέας στέλνει στον ασθενή.

3.3. Διαστασιοποίηση του Προβλήματος

Οι παράμετροι που επηρεάζουν το πρόβλημα της κατανομής των υπηρεσιών διαδικτύου είναι πολλές και αυτό κάνει δύσκολη την μοντελοποίηση του προβλήματος. Στην συγκεκριμένη ενότητα, περιγράφουμε τις παραμέτρους που υπεισέρχονται στο workflow και στο δίκτυο εξυπηρετητών σύμφωνα με τις οποίες μπορεί να οριστεί το πρόβλημα.

Για ένα workflow πρέπει να είναι γνωστός ο αριθμός των λειτουργιών που έχει και για κάθε λειτουργία πρέπει να ορίσουμε τους κύκλους που χρειάζεται για να εκτελεστεί. Ορίζουμε τρεις κατηγορίες «όγκου εργασίας» για τις λειτουργίες, μεγάλος, μεσαίος και μικρός όγκος εργασίας. Κάθε λειτουργία ανήκει σε μία από τις τρεις κατηγορίες ανάλογα με τους κύκλους που χρειάζεται για να εκτελεστεί. Οι λειτουργίες που χρειάζονται πλήθος κύκλων της τάξης του 10^6 έχουν μικρό όγκο εργασίας, αυτές που χρειάζονται πλήθος κύκλων της τάξης του 10^7 έχουν μεσαίο όγκο εργασίας και αυτές που χρειάζονται πλήθος κύκλων της τάξης του 10^8 έχουν μεγάλο όγκο εργασίας. Στον Πίνακα 3.3 φαίνονται τρεις διαφορετικές περιπτώσεις που μελετάμε για τον όγκο εργασίας των λειτουργιών. Μια άλλη παράμετρος που πρέπει να γνωρίζουμε για το workflow είναι τα μεγέθη των μηνυμάτων που ανταλλάσσουν μεταξύ τους οι λειτουργίες. Τα μεγέθη των μηνυμάτων που

ανταλλάσσουν οι λειτουργίες ανήκουν κι αυτά σε τρεις κατηγορίες: μικρά, μεσαία και μεγάλα. Στον Πίνακα 3.4 φαίνονται τρεις διαφορετικές περιπτώσεις που μελετάμε για το μέγεθος των μηνυμάτων. Επίσης, μία παράμετρος είναι η τοπολογία του workflow, η οποία μπορεί να είναι γραμμή ή ένας τυχαίος γράφος. Στην τελευταία περίπτωση πρέπει να ορισθεί το fan-out του γράφου, οι τύποι των διακλαδώσεων που μπορεί να περιέχει (and, or, xor) και η συχνότητα κάθε τύπου διακλάδωσης. Όσον αφορά το fan-out, εμείς θεωρούμε τρεις τύπους workflow: θαμνώδη (bushy), επιμήκη (lengthy) και ένα υβριδικά (hybrid). Το ποσοστό conditional-operational λειτουργιών στα bushy workflows είναι περίπου 50%-50%, στα lengthy workflows περίπου 16%-84% και στα hybrid workflows περίπου 35%-65%.

Πίνακας 3.3 Περιπτώσεις που Μελετάμε για τον Όγκο Εργασίας των Λειτουργιών του Workflow.

Όγκος εργασίας (κύκλοι) / Περιπτώσεις που μελετάμε	**Μικρός** *25% των μηνυμάτων*	**Μεσαίος** *50% των μηνυμάτων*	**Μεγάλος** *25% των μηνυμάτων*
1η περίπτωση	3×10^6	5×10^7	7×10^8
2η περίπτωση	30×10^6	50×10^7	70×10^8
3η περίπτωση	300×10^6	500×10^7	700×10^8

Πίνακας 3.4 Περιπτώσεις που Μελετάμε για το Μέγεθος των Μηνυμάτων του Workflow.

Μέγεθος Μηνυμάτων / Πιθανές Περιπτώσεις	**Μικρό** *25% των μηνυμάτων*	**Μεσαίο** *50% των μηνυμάτων*	**Μεγάλο** *25% των μηνυμάτων*
1η περίπτωση	0,002 Mbits	0,006 Mbits	0,01 Mbits
2η περίπτωση	0,02 Mbits	0,06 Mbits	0,1 Mbits
3η περίπτωση	0,1 Mbits	0,2 Mbits	0,3 Mbits

Για το δίκτυο εξυπηρετητών πρέπει να γνωρίζουμε τον αριθμό των εξυπηρετητών και για κάθε εξυπηρετητή μπορούν να ορισθούν οι εξής παράμετροι: η υπολογιστική του ισχύς (σε κύκλους το δευτερόλεπτο), ο μέσος χρόνος για αποτυχία και ο μέσος χρόνος για εκκίνηση ή επανεκκίνηση του εξυπηρετητή. Μία σημαντική παράμετρος που πρέπει να γνωρίζουμε είναι η τοπολογία του δικτύου που συγκροτούν οι εξυπηρετητές. Η τοπολογία μπορεί να είναι γραμμή, δίαυλος ή ένας τυχαίος γράφος. Για κάθε σύνδεση μεταξύ δύο εξυπηρετητών πρέπει να ορισθεί η ταχύτητα της

σύνδεσης και ο χρόνος διάδοσης της σύνδεσης. Εμείς θεωρούμε ότι υπάρχουν τρεις διαφορετικές ταχύτητες σύνδεσης: 10 Mbps, 100 Mbps και 1000 Mbps. Για όλες τις συνδέσεις θεωρούμε ότι ο χρόνος διάδοσης είναι 0,0001 sec. Επίσης, μια παράμετρος που μπορεί να επηρεάσει την κατανομή των λειτουργιών του workflow είναι η εξάρτηση λειτουργιών από πόρους που μπορεί να υπάρχουν στους εξυπηρετητές. Για παράδειγμα, μία λειτουργία μπορεί να χρησιμοποιεί μία βάση δεδομένων ή ένα συγκεκριμένο αρχείο. Στην εργασία αυτή δεν θεωρούμε ότι υπάρχουν τέτοιου είδους εξαρτήσεις και περιορισμοί. Όσον αφορά την υπολογιστική ισχύ των εξυπηρετητών ορίζουμε τρεις κατηγορίες δυναμικότητας: μεγάλη, μεσαία και μικρή. Ένας εξυπηρετητής βρίσκεται στην μεγάλη κατηγορία δυναμικότητας αν η υπολογιστική του ισχύς είναι 3 GHz, στη μεσαία κατηγορία δυναμικότητας αν η υπολογιστική του ισχύς είναι 2 GHz και στην μικρή κατηγορία δυναμικότητας αν η υπολογιστική του ισχύς είναι 1 GHz. Τέσσερις διαφορετικές περιπτώσεις που θα μπορούσαν να ισχύουν την υπολογιστική ισχύ των εξυπηρετητών φαίνονται στον Πίνακα 3.5.

Πίνακας 3.5 Πιθανές Περιπτώσεις για την Υπολογιστική Ισχύ των Εξυπηρετητών.

Υπολογιστική ισχύς / Πιθανές Περιπτώσεις	**Μικρή** *1 GHz*	**Μεσαία** *2 GHz*	**Μεγάλη** *3 GHz*
1η περίπτωση	Όλοι οι εξυπηρετητές		
2η περίπτωση		Όλοι οι εξυπηρετητές	
3η περίπτωση			Όλοι οι εξυπηρετητές
4η περίπτωση	25% των εξυπηρετητών	50% των εξυπηρετητών	25% των εξυπηρετητών

Για τον αριθμό των λειτουργιών σε σχέση με τους εξυπηρετητές θέλουμε να μελετήσουμε διάφορες αναλογίες. Για το λόγο αυτό ορίζουμε τον παράγοντα ομαδοποίησης K ως έναν θετικό ακέραιο αριθμό που εκφράζει πόσες φορές μεγαλύτερος είναι ο αριθμός των λειτουργιών από τον αριθμό των εξυπηρετητών.

Η αξιολόγηση μιας κατανομής των λειτουργιών μιας σύνθετης υπηρεσίας διαδικτύου σε ένα δίκτυο εξυπηρετητών μπορεί να αξιολογηθεί με το χρόνο εκτέλεσης της υπηρεσίας, τη ρυθμοαπόδοση και τη διαθεσιμότητα της υπηρεσίας αλλά και την κατανομή του φορτίου στους εξυπηρετητές του δικτύου. Ο χρόνος εκτέλεσης της υπηρεσίας είναι ο χρόνος που απαιτείται από τη στιγμή που όλα τα δεδομένα εισόδου είναι διαθέσιμα και ξεκινάει την εκτέλεσή της η υπηρεσία μέχρι τη στιγμή που

τελειώνει η εκτέλεση της υπηρεσίας και τα αποτελέσματα βρίσκονται στην έξοδο. Η ρυθμοαπόδοση της υπηρεσίας είναι ο αριθμός των διαφορετικών αιτήσεων που μπορούν να εκτελεστούν από την υπηρεσία στη μονάδα του χρόνου. Διαθεσιμότητα είναι το ποσοστό του χρόνου που η υπηρεσία είναι διαθέσιμη στους χρήστες. Τέλος, η κατανομή του φορτίου είναι ένα μέτρο που δείχνει πόσο δίκαια κατανέμεται το φορτίο της εκτέλεση της υπηρεσίας στους εξυπηρετητές.

Όλες οι παράμετροι με βάση τις οποίες μπορεί να ορισθεί το παραπάνω πρόβλημα συγκεντρώνονται στις τρεις επόμενες παραγράφους.

3.3.1. Παράμετροι για το Workflow

Συνοπτικά οι παράμετροι που χαρακτηρίζουν ένα workflow συνοψίζονται στην παρακάτω λίστα:

- Αριθμός λειτουργιών
- Κύκλοι που χρειάζεται να εκτελεστεί κάθε λειτουργία
- Μέγεθος μηνυμάτων που ανταλλάσσουν οι λειτουργίες
- Τοπολογία:
    - Fan-out
    - Τύπος διακλαδώσεων (and, or, xor)
    - Συχνότητα κάθε τύπου διακλαδώσεων

3.3.2. Παράμετροι για το Δίκτυο Εξυπηρετητών

Στην παρακάτω λίστα συγκεντρώνονται οι παράμετροι που πρέπει να ορίζονται για ένα δίκτυο εξυπηρετητών:

- Αριθμός των εξυπηρετητών
- Υπολογιστική ισχύς του κάθε εξυπηρετητή
- Μέσος χρόνος για αποτυχία
- Μέσος χρόνος για εκκίνηση-επανεκκίνηση
- Τοπολογία των εξυπηρετητών: γραμμή, δίαυλος, τυχαίος γράφος, πλήρης γράφος
- Ταχύτητα συνδέσεων

- Τοποθεσία πόρων που χρησιμοποιούνται από τις λειτουργίες

3.3.3. Χαρακτηριστικά Αξιολόγησης Μιας Αντιστοίχισης

Σε αυτήν την παράγραφο συνοψίζονται τα χαρακτηριστικά με βάση τα οποία μπορεί να γίνει η αξιολόγηση μιας αντιστοίχισης:

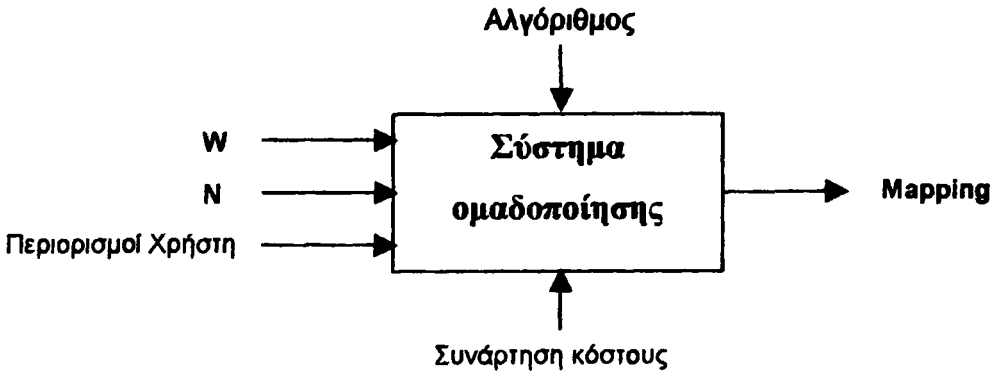
- χρόνος εκτέλεσης ενός workflow
- κατανομή του φορτίου στους εξυπηρετητές
- ρυθμοαπόδοση ενός workflow
- διαθεσιμότητα ενός workflow

3.4. Μοντελοποίηση του Προβλήματος

Αφού είδαμε στην προηγούμενη ενότητα τις διαστάσεις του προβλήματος, προχωράμε στον αυστηρό ορισμό και τη μοντελοποίηση του προβλήματος. Για να κάνουμε πιο συγκεκριμένο το πρόβλημα δεν μελετάμε όλες τις διαστάσεις του προβλήματος και επικεντρωνόμαστε μόνο σε μερικές παραμέτρους που θεωρούμε σημαντικές. Αυτές είναι η υπολογιστική ισχύς των εξυπηρετητών, οι κύκλοι που χρειάζονται οι λειτουργίες για να εκτελεστούν, η ταχύτητα των συνδέσεων μεταξύ των εξυπηρετητών και τα μεγέθη των μηνυμάτων που ανταλλάσσουν οι λειτουργίες μεταξύ τους.

Ο στόχος της εργασίας είναι η κατανομή υπηρεσιών διαδικτύου σε ένα δίκτυο εξυπηρετητών. Θέλουμε να κατασκευάσουμε, ένα σύστημα ομαδοποίησης που θα δέχεται ως είσοδο ένα workflow από υπηρεσίες διαδικτύου, ένα δίκτυο εξυπηρετητών και ένα σύνολο περιορισμών που θέτει ο χρήστης και να επιστρέφει μία αντιστοίχιση των υπηρεσιών διαδικτύου στους εξυπηρετητές. Ορίζουμε με $\mathbf{O} = \{O_1, O_2, \ldots, O_M\}$ ένα σύνολο από λειτουργίες και με $\mathbf{S} = \{S_1, S_2, \ldots, S_N\}$ ένα σύνολο από εξυπηρετητές. Ένα workflow είναι ένας γράφος από λειτουργίες: $\mathbf{W}(O, E)$, όπου $E=\{(o_p, o_n) \mid o_p, o_n \in O, \exists$ μετάβαση από την o_p στην $o_n\}$. Μία μετάβαση (o_p, o_n) είναι ένα μήνυμα που στέλνει η λειτουργία o_p στην λειτουργία o_n δηλαδή, τα δεδομένα εξόδου της o_p που εισέρχονται ως δεδομένα εισόδου στην o_n. Ένα δίκτυο εξυπηρετητών είναι ένας γράφος από

εξυπηρετητές: **N(S, L)** όπου $L=\{(s_i, s_j) \mid s_i, s_j \in S, \exists$ σύνδεση του εξυπηρετητή s_i με τον εξυπηρετητή $s_j\}$. Το ζητούμενο είναι μία αντιστοίχιση των στοιχείων του συνόλου O σε στοιχεία του συνόλου S. Η αντιστοίχιση αυτή μοντελοποιείται με ένα σύνολο **Mapping** = $\{r_1, r_2, \ldots, r_M \mid \forall i=1,2,\ldots,M$: r_i κανόνας της μορφής $o \rightarrow s$, $o \in O$ και $s \in S$ (η λειτουργία "o" τοποθετείται στον εξυπηρετητή "s")$\}$. Η αντιστοίχιση που επιστρέφει το σύστημα ομαδοποίησης θέλουμε να ικανοποιεί τους περιορισμούς που θέτει ο χρήστης. Οι περιορισμοί αυτοί μπορούν να αφορούν το χρόνο εκτέλεσης, τη ρυθμοαπόδοση και το ποσοστό επιτυχίας του workflow, όπως και την κατανομή φορτίου στους εξυπηρετητές. Το σύστημα ομαδοποίησης θα χρησιμοποιεί μία συνάρτηση κόστους η οποία θα αξιολογεί την ποιότητα της ομαδοποίησης. Η αντιστοίχιση που θα επιστρέφει το σύστημα θέλουμε να έχει όσο το δυνατόν λιγότερο κόστος σύμφωνα με τη συνάρτηση κόστους. Δηλαδή, δοθέντος (α) ένα workflow **W**, (β) ένα δίκτυο εξυπηρετητών **S** και (γ) ένα σύνολο περιορισμών από τον χρήστη, το σύστημα ομαδοποίησης θα πρέπει να υπολογίζει μία αντιστοίχιση **Mapping**, της οποίας το κόστος είναι ελάχιστο. Επειδή οι παράμετροι του προβλήματος είναι πολλές, το σύστημα πρέπει να χρησιμοποιεί μία σειρά από αλγορίθμους ώστε να επιλέγει για κάθε διαφορετική ανάθεση παραμέτρων τον καταλληλότερο αλγόριθμο, με τον οποίο θα κατασκευάσει το σύνολο **Mapping**. Σχηματικά, το σύστημα ομαδοποίησης που θέλουμε να υλοποιήσουμε φαίνεται στο Σχήμα 3.4.



Σχήμα 3.4 Το Σύστημα Ομαδοποίησης Δέχεται ως Είσοδο Ένα Workflow, Ένα Δίκτυο Εξυπηρετητών και Ένα Σύνολο Περιορισμών, Χρησιμοποιεί Έναν από τους Αλγορίθμους και τη Συνάρτηση Κόστους και Δίνει Έξοδο μία Αντιστοίχιση των Λειτουργιών του Workflow στους Εξυπηρετητές.

3.5. Ιδιότητες Μιας Καλής Λύσης του Προβλήματος

Αρχικά, σε μια πρώτη προσέγγιση, τα χαρακτηριστικά μιας λύσης στο πρόβλημά μας, τα οποία μας ενδιαφέρουν, είναι ο χρόνος εκτέλεσης του workflow και η κατανομή του φορτίου. Από το σημείο αυτό και στο εξής θεωρούμε ότι τα δύο αυτά χαρακτηριστικά ενδιαφέρουν με την ίδια βαρύτητα το χρήστη. Όσον αφορά την κατανομή του φορτίου στους εξυπηρετητές μας ενδιαφέρει ως χρήστες να υπάρχει *δίκαιη κατανομή του φορτίου*. Δηλαδή, όσο μεγαλύτερη υπολογιστική ισχύ έχει ένας εξυπηρετητής, τόσο περισσότερο φορτίο (κύκλους) πρέπει να αναλάβει. Με άλλα λόγια θέλουμε όλοι οι εξυπηρετητές να δαπανούν ακριβώς τον ίδιο χρόνο για την εκτέλεση των λειτουργιών που αναλαμβάνουν. Αυτή είναι και η πιο ιδανική λύση για μια δίκαιη κατανομή φορτίου. Όσον αφορά το χρόνο εκτέλεσης, είναι προφανές, ότι θέλουμε *όσο το δυνατόν μικρότερο χρόνο εκτέλεσης* του workflow. Διαπιστώνεται εύκολα ότι προσπαθώντας να βελτιώσουμε ένα από τα δύο χαρακτηριστικά μπορεί να χαλάει το άλλο. Για παράδειγμα, αν ανατεθούν όλες οι λειτουργίες του workflow στον εξυπηρετητή με τη μεγαλύτερη υπολογιστική ισχύ, μπορεί ο χρόνος εκτέλεσης του workflow να ελαχιστοποιείται αλλά η κατανομή φορτίου είναι πολύ κακή. Από την άλλη, είναι δυνατόν μια πολύ δίκαιη κατανομή του φορτίου να αυξάνει πάρα πολύ τον χρόνο εκτέλεσης του workflow γιατί πιθανόν να σπαταλάται πολύς χρόνος στα μηνύματα που ανταλλάσσουν οι εξυπηρετητές.

Ένας τρόπος για να βρούμε την βέλτιστη λύση στο πρόβλημα, με τους περιορισμούς που θέσαμε στην προηγούμενη παράγραφο, δηλαδή, ότι μας ενδιαφέρει εξίσου η δίκαιη κατανομή φορτίου και ο χρόνος εκτέλεσης, είναι να εξετάσουμε όλες τις δυνατές λύσεις στο πρόβλημα και να επιλέξουμε την καλύτερη. Όμως, η κατασκευή όλων των δυνατών λύσεων είναι πολύ ακριβή και απαγορευτική για μεγάλο αριθμό από εξυπηρετητές και λειτουργίες *(βλέπετε ενότητα 4.1 Εξαντλητικός Αλγόριθμος)* και σε αυτήν την περίπτωση πρέπει να χρησιμοποιηθούν ευριστικοί αλγόριθμοι. Αυτοί οι ευριστικοί αλγόριθμοι πρέπει να εκμεταλλεύονται τις ιδιότητες που έχουν οι καλές λύσεις ώστε να δίνουν μία καλή λύση χωρίς να παράγουν όλες τις λύσεις. Οπότε, είναι σημαντικό να γνωρίζουμε τις ιδιότητες που έχουν οι καλές λύσεις ώστε οι ευριστικοί αλγόριθμοι να κατασκευάζουν τη λύση τους ακολουθώντας αυτές τις ιδιότητες.

Για να βρούμε τις ιδιότητες των καλών λύσεων εκτελέσαμε των εξαντλητικό αλγόριθμο σε απλά workflows με λίγες λειτουργίες και μελετήσαμε τα χαρακτηριστικά που είχαν οι καλύτερες λύσεις. Το συμπέρασμα που προκύπτει από τα πειράματα αυτά είναι ότι οι ιδιότητες που χαρακτηρίζουν τις καλές λύσεις στα πειράματα αυτά είναι κοινές και συνοψίζονται στις εξής προτάσεις:

1. *Αναλογία φορτίου-υπολογιστικής ισχύος.* Η πρώτη ιδιότητα που έχουν οι καλές λύσεις είναι ότι το φορτίου των λειτουργιών που αναλαμβάνει ένας εξυπηρετητής, δηλαδή το άθροισμα των κύκλων τους, είναι ανάλογο με την υπολογιστική του ισχύ. Δηλαδή, αν ένας εξυπηρετητής με υπολογιστική ισχύ 3 GHz εξυπηρετεί λειτουργίες οι οποίες απαιτούν συνολικά για την εκτέλεσή τους περίπου 3 εκατομμύρια κύκλους, τότε ένας εξυπηρετητής με υπολογιστική ισχύ 1 GHz εξυπηρετεί λειτουργίες οι οποίες απαιτούν συνολικά για την εκτέλεσή τους περίπου 1 εκατομμύριο κύκλους.
2. *Αποστολή μηνυμάτων μικρού μεγέθους από εξυπηρετητή σε εξυπηρετητή.* Η ιδιότητα αυτή υποδεικνύει ότι για να οδηγηθούμε σε μια καλή λύση πρέπει τα μηνύματα που στέλνονται από εξυπηρετητή σε εξυπηρετητή να είναι όσο το δυνατόν γίνεται μικρότερα. Με άλλα λόγια το κλάσμα μέγεθος μηνύματος προς ταχύτητα της γραμμής της οποίας στέλνεται το μήνυμα πρέπει να είναι όσο γίνεται μικρό.
3. *Αποστολή λίγων μηνυμάτων από εξυπηρετητή σε εξυπηρετητή.* Η ιδιότητα αυτή λέει ότι στις καλές λύσεις ο αριθμός τον μηνυμάτων που ανταλλάσσονται από εξυπηρετητή σε εξυπηρετητή είναι μικρός σε σχέση με τις υπόλοιπες λύσεις.

Η πρώτη ιδιότητα είναι προφανές ότι ισχύει γιατί θέλουμε καλή κατανομή του φορτίου ενώ οι υπόλοιπες δύο απορρέουν από την διαπίστωση ότι σε ένα workflow, κυρίως όταν η ταχύτητα των συνδέσεων μεταξύ των εξυπηρετητών είναι μικρή, ο περισσότερος χρόνος για την εκτέλεσή του δαπανάται στην μεταφορά των μηνυμάτων από εξυπηρετητή σε εξυπηρετητή.

3.6. Συμβολισμοί και Τύποι

Στην ενότητα αυτή δίνονται οι συμβολισμοί των παραμέτρων του προβλήματος (Πίνακας 3.6) και οι τύποι που ισχύουν για τις παραμέτρους του συστήματος και για τα χαρακτηριστικά αξιολόγησης.

Πίνακας 3.6 Συμβολισμοί – Τελεστές των Παραμέτρων του Προβλήματος.

Συμβολισμός	Είσοδος	Έξοδος
C(op)	Μία λειτουργία op∈O.	Οι κύκλοι ρολογιού που χρειάζεται η op για να εκτελεστεί.
P(s)	Ένας εξυπηρετητής s∈S.	Η υπολογιστική ισχύς που διαθέτει ο εξυπηρετητής s σε κύκλους/δευτερόλεπτο.
Server(op)	Μία λειτουργία op∈O.	Ο εξυπηρετητής όπου βρίσκεται τοποθετημένη η λειτουργία op.
$T_{comm}(op_i, op_j)$	Δύο λειτουργίες $op_i, op_j \in O$.	Ο χρόνος για την επικοινωνία της op_i με την op_j (να στείλει η op_i τα δεδομένα στην op_j). Πρέπει $(op_i, op_j) \in E$.
$T_{prop}(s_i, s_j)$	Δύο εξυπηρετητές $s_i, s_j \in S$.	Ο χρόνος διάδοσης μεταξύ των εξυπηρετητών s_i και s_j.
$Path(s_i, s_j)$	Δύο εξυπηρετητές $s_i, s_j \in S$.	Το μονοπάτι που ακολουθεί ένα μήνυμα που στέλνεται από τον εξυπηρετητή s_i στον εξυπηρετητή s_j. Δίνεται από κάποιον αλγόριθμο δρομολόγησης.
$T_{trans}(op_i, op_j)$	Δύο λειτουργίες $op_i, op_j \in O$.	Ο χρόνος μετάδοσης όταν επικοινωνεί η λειτουργία op_i με την λειτουργία op_j.
$T_{proc}(op)$	Μία λειτουργία op∈O.	Ο χρόνος που χρειάζεται ο εξυπηρετητής Server(op) για την επεξεργασία της λειτουργίας op.
$MsgSize(op_i, op_j)$	Δύο λειτουργίες $op_i, op_j \in O$.	Το μέγεθος του μηνύματος σε bytes που πρέπει να στείλει η λειτουργία op_i στην λειτουργία op_j. Πρέπει $(op_i, op_j) \in E$.
$Line\_Speed(s_i, s_j)$	Δύο εξυπηρετητές $s_i, s_j \in S$.	Η ταχύτητα της γραμμής (bps) που συνδέει τον εξυπηρετητή s_i με τον εξυπηρετητή s_j.
Load(s)	Ένας εξυπηρετητής s∈S.	Το συνολικό φορτίο εργασίας του εξυπηρετητή s. Είναι ο χρόνος που είναι απασχολημένος ο s με τις λειτουργίες για τις οποίες είναι υπεύθυνος.
$T_{execute}$	Μία αντιστοίχιση των λειτουργιών ενός W σε ένα N.	Ο χρόνος εκτέλεσης του W.
Time_Penalty	Μία αντιστοίχιση των λειτουργιών ενός W σε ένα N.	Η ποσότητα που μετράει πόσο καλή κατανομή φορτίου έχουμε στους εξυπηρετητές του N.

Τύποι

- $T_{proc}(op) = \frac{C(op)}{P(Server(op))}$

  Ο χρόνος επεξεργασίας μιας λειτουργίας υπολογίζεται διαιρώντας τους υπολογιστικούς κύκλους της λειτουργίας με την υπολογιστική ισχύ του εξυπηρετητή όπου εκτελείται η λειτουργία.

- $Load(s) = \sum_{j} T_{proc}(O_j)$, $O_j \rightarrow s \in$ **Mapping**, $O_j \in$ **O**.

  Το φορτίο ενός εξυπηρετητή ισούται με τον συνολικό χρόνο επεξεργασίας των λειτουργιών που εκτελούνται στον συγκεκριμένο εξυπηρετητή.

- $Path(s_i, s_j) = \{(s_a, s_b) \mid$ το μήνυμα που στέλνει ο s_i στον s_j στέλνεται από τον s_a στον s_b, $a,b \in S\}$

  $Path(s_i, s_j)$ είναι το σύνολο των ζευγών των εξυπηρετητών (αποστολέας-παραλήπτης) από τους οποίους περνάει ένα μήνυμα που στέλνεται από τον s_i στον s_j.

- $T_{trans}(op_i, op_j) = \sum_{a} \frac{MsgSize(opi,opj)}{Line\_Speed(s_a, s_b)}$, $(s_a,s_b) \in Path(Server(op_i), Server(op_j))$

  Ο χρόνος μετάδοσης ενός μηνύματος υπολογίζεται διαιρώντας το μέγεθος του μηνύματος με την ταχύτητα της γραμμής μέσω της οποίας γίνεται η αποστολή του μηνύματος. Επειδή ένα μήνυμα μπορεί να περνάει από πολλές γραμμές, ο παραπάνω τύπος αθροίζει όλους τους χρόνους μετάδοσης της κάθε γραμμής.

- $T_{comm}(op_i, op_j) = \sum_{a} T_{prop}(s_a, s_b) + T_{trans}(op_i, op_j)$, $(s_a,s_b) \in Path(Server(op_i), Server(op_j))$

  Ο χρόνος επικοινωνίας μεταξύ δύο λειτουργιών είναι ο χρόνος που χρειάζεται για να σταλεί το μήνυμα που ανταλλάσσουν οι λειτουργίες μέσω του δικτύου των εξυπηρετητών. Ισούται με το άθροισμα του χρόνου διάδοσης και του χρόνου μετάδοσης.

- $Time\_Penalty = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{|Load(s_i) - Load(s_j)|}{(1/2) \times N \times (N-1)}$ Εξ. 3.1

  Το Time_Penalty εκφράζει τη διαφορά του φορτίου (Load), μεταξύ δύο οποιονδήποτε εξυπηρετητών. Το Load ενός εξυπηρετητή είναι ο χρόνος που απασχολείται ο εξυπηρετητής με την επεξεργασία των λειτουργιών οι οποίες, είναι

τοποθετημένες στον συγκεκριμένο εξυπηρετητή. Υπολογίζεται ως εξής: για όλα τα ζεύγη των εξυπηρετητών υπολογίζουμε την διαφορά του φορτίου τους κατά απόλυτη τιμή και η μέση τιμή αυτών των διαφορών είναι το Time_Penalty.

- $T_{execute} = \sum_{j=1}^{M} T_{proc}(O_j) + T_{comm}^{(total)}$, Εξ. 3.2

Ο χρόνος εκτέλεσης του workflow ισούται με το άθροισμα του χρόνου επεξεργασίας όλων των λειτουργιών και του χρόνου επικοινωνίας των λειτουργιών ($T_{comm}^{(total)}$). Η μαθηματική φόρμουλα που υπολογίζει τον χρόνο επικοινωνίας δεν είναι συγκεκριμένη και εξαρτάται από την τοπολογία του workflow. Για παράδειγμα, για το workflow του Σχήματος 3.2, ο χρόνος εκτέλεσης υπολογίζεται ως εξής:

$$T_{execute} = T_{proc}(O_1) + T_{comm}(O_1, O_2) + T_{proc}(O_2) + \{T_{AND} \mid T_{OR} \mid T_{XOR}\} + T_{proc}(O_5) + T_{comm}(O_5, O_6) + T_{proc}(O_6),$$

όπου $T_{AND} = \max\{T_1, T_2\}$

$T_{OR} = \min\{T_1, T_2\}$

$T_{XOR} = f_1 \times T_1 + f_2 \times T_2$.

με $T_1 = T_{comm}(O_2, O_3) + T_{proc}(O_3) + T_{comm}(O_3, O_5)$ και

$T_2 = T_{comm}(O_2, O_4) + T_{proc}(O_4) + T_{comm}(O_4, O_5)$ οι χρόνοι απόκρισης των δύο κλάδων

και f_1, f_2 οι συχνότητες εκτέλεσης των δύο κλάδων αντίστοιχα.

ΚΕΦΑΛΑΙΟ 4. ΑΛΓΟΡΙΘΜΟΙ

4.1 Εξαντλητικός Αλγόριθμος
4.2 Αλγόριθμος Γραμμή–Γραμμή
4.3 Αλγόριθμοι Γραμμή–Δίαυλος
4.4 Αλγόριθμοι Τυχαίος Γράφος–Δίαυλος

Σε αυτό το κεφάλαιο περιγράφουμε αρχικά τον εξαντλητικό αλγόριθμο για την επίλυση του προβλήματος που ορίστηκε στο τρίτο κεφάλαιο. Στη συνέχεια περιγράφουμε τους αλγορίθμους που κατασκευάσαμε για τρεις διαστάσεις του προβλήματος, οι οποίες διαφοροποιούνται μεταξύ τους στις τοπολογίες των σύνθετων υπηρεσιών και του δικτύου. Για κάθε αλγόριθμο δίνεται και ο ψευδοκώδικάς του. Η επιλογή των διαστάσεων έγινε ώστε να μελετηθεί το πρόβλημα πρώτα στην πιο απλή του μορφή και έπειτα στις πιο πολύπλοκες μορφές.

4.1. Εξαντλητικός Αλγόριθμος

Για να βρούμε όλες τις λύσεις του προβλήματος που περιγράφτηκε στην ενότητα 3.2, χρησιμοποιούμε έναν εξαντλητικό αλγόριθμο. Ο αλγόριθμος δέχεται ως είσοδο ένα σύνολο από λειτουργίες και ένα σύνολο από εξυπηρετητές και επιστρέφει όλες τις δυνατές αντιστοιχίσεις. Για M λειτουργίες και N εξυπηρετητές όλες οι δυνατές λύσεις είναι N^M. Αυτή είναι και η πολυπλοκότητα του αλγορίθμου και επειδή είναι εκθετική ο αλγόριθμος δεν μπορεί να εφαρμοστεί για μεγάλο αριθμό από λειτουργίες και

εξυπηρετητές. Ενδεικτικά, ο αλγόριθμος είναι εφαρμόσιμος το πολύ για 10-11 λειτουργίες και 3-4 εξυπηρετητές. Αυτό είναι και το μοναδικό μειονέκτημα του εξαντλητικού αλγορίθμου. Αντιθέτως, το πλεονέκτημά του είναι ότι με τον εξαντλητικό αλγόριθμο μπορούμε να βρούμε τη βέλτιστη λύση για οποιοδήποτε σύνολο περιορισμών. Στη συνέχεια ακολουθεί ο ψευδοκώδικας του αλγορίθμου.

Εξαντλητικός Αλγόριθμος (Ψευδοκώδικας)

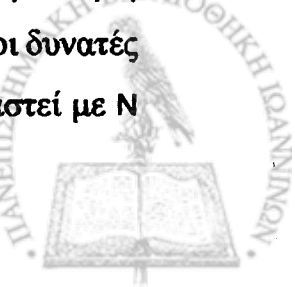
Είσοδος: Ένα σύνολο από λειτουργίες **O** = $\{O_1, O_2, ..., O_M\}$ και ένα σύνολο από εξυπηρετητές **S** = $\{S_1, S_2, ..., S_N\}$.

Έξοδος: Ένα σύνολο αντιστοιχίσεων **M** = $\{ M_j \{\{O\} \rightarrow \{S\}\} \mid j = 1, ..., N^M \}$.

Περιγραφή: ο αλγόριθμος παράγει το όλες τις δυνατές αντιστοιχίσεις που μπορεί να έχει το πρόβλημα. Ένας κανόνας $o \rightarrow s$ σημαίνει ότι η λειτουργία o τοποθετείται στον εξυπηρετητή s.

```
Begin
1   M = ∅, M' = ∅
2   o = O1
3   for each s∈S do
4       M' = M' ∪ {o→s}
5   for j = 2 to M do
6       for each m∈M' do
7           for each s∈S do
8               M = M ∪ {m ∪ {Oj→s}}
9           end for
10      end for
11      M' = M
12      if j<M   M = ∅
13  end for
14  return M
End
```

Ο εξαντλητικός αλγόριθμος αρχικά κατασκευάζει N διαφορετικούς κανόνες (γραμμές 1-4) που αντιστοιχούν στις αναθέσεις της O_1 στους N εξυπηρετητές. Αυτοί οι N κανόνες επεκτείνονται στα επόμενα βήματα του αλγορίθμου με την προσθήκη κανόνων που αφορούν τις υπόλοιπες λειτουργίες ώστε να προκύψουν όλες οι δυνατές αντιστοιχίσεις. Κάθε ένας από τους N αρχικούς κανόνες μπορεί να συνδυαστεί με N



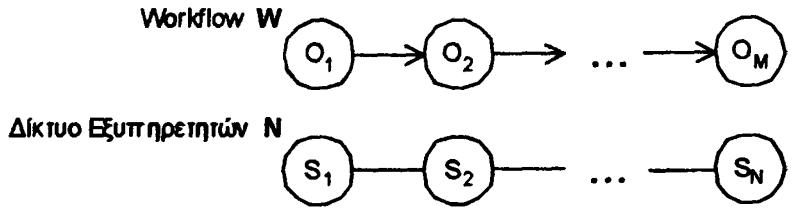
κανόνες που αφορούν την O_2, οι οποίοι εκφράζουν τις N διαφορετικές αναθέσεις της στους εξυπηρετητές. Έτσι, προκύπτουν N^2 σύνολα κανόνων τα οποία, επεκτείνονται με την προσθήκη κανόνων που αφορούν την O_3. Η O_3 μπορεί να ανατεθεί επίσης στους N εξυπηρετητές, επομένως, οι N αντίστοιχοι κανόνες συνδυάζονται με τα N^2 υποσύνολα κανόνων του προηγούμενου βήματος και προκύπτουν N^3 καινούρια υποσύνολα κανόνων. Η διαδικασία προσθήκης κανόνων συνεχίζεται έως ότου σε κάθε σύνολο κανόνων προστεθούν οι κανόνες που αφορούν όλες τις λειτουργίες. Έτσι τα N^M σύνολα κανόνων που προκύπτουν αποτελούν όλες τις δυνατές αντιστοιχίσεις των λειτουργιών στους εξυπηρετητές. Η διαδικασία προσθήκης κανόνων, που μόλις περιγράφτηκε, φαίνεται στις γραμμές 5-8 του ψευδοκώδικα. Ο αλγόριθμος παίρνει μία-μία τις λειτουργίες (γραμμή 5) και επεκτείνει κάθε σύνολο κανόνων του προηγούμενου βήματος (γραμμή 6), με N διαφορετικούς κανόνες (γραμμές 7-8) οι οποίοι αφορούν τις διαφορετικές αναθέσεις που μπορούν να γίνουν για τη συγκεκριμένη λειτουργία. Το σύνολο **M'** είναι βοηθητικό για την αποθήκευση των συνόλων κανόνων του προηγούμενου βήματος.

4.2. Αλγόριθμος Γραμμή–Γραμμή

Ο αλγόριθμος Γραμμή-Γραμμή είναι ένας ευριστικός αλγόριθμος για την εύρεση λύσης στο πρόβλημά μας όταν η τοπολογία τόσο του workflow όσο και του δικτύου εξυπηρετητών είναι γραμμή, όπως υποδηλώνει και το όνομά του. Ο αλγόριθμος δέχεται ως εισόδους ένα workflow και ένα δίκτυο εξυπηρετητών και επιστρέφει μία αντιστοίχιση. Για την κατασκευή της αντιστοίχισης ο αλγόριθμος στηρίζεται στις ιδιότητες των καλών λύσεων που περιγράφτηκαν στην ενότητα 3.5 και μπορούμε να τον χωρίσουμε σε δύο μέρη. Στο πρώτο μέρος, προσπαθεί να ικανοποιήσει συγχρόνως την πρώτη και τρίτη ιδιότητα, τοποθετώντας τις λειτουργίες στους εξυπηρετητές με τέτοιο τρόπο ώστε να υπάρχει δίκαιη κατανομή φορτίου και ο αριθμός των μηνυμάτων που ανταλλάσσονται να είναι ο ελάχιστος. Στο δεύτερο μέρος, ο αλγόριθμος προσπαθεί να ικανοποιήσει την δεύτερη ιδιότητα, μετακινώντας λειτουργίες σε διπλανούς εξυπηρετητές ώστε να μην στέλνονται μηνύματα μεγάλου μεγέθους σε γραμμές μικρής ταχύτητας. Η πολυπλοκότητα του αλγορίθμου είναι

γραμμική, O(M) είναι η πολυπλοκότητα του πρώτου μέρους και O(N) η πολυπλοκότητα του δεύτερου μέρους.

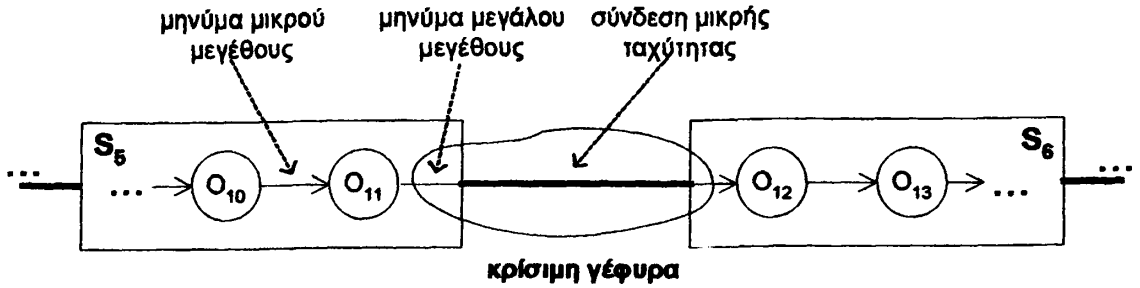
Στη συνέχεια αναλύεται η λειτουργία του αλγορίθμου λεπτομερέστερα. Οι είσοδοι του αλγορίθμου είναι ένα workflow W(O, E), όπου $O = (O_1, O_2, \ldots, O_M)$ και $E = \{(O_i, O_{i+1}) \mid \forall\ i = 1, \ldots, M-1\}$ και ένα δίκτυο εξυπηρετητών N(S, L), όπου $S = \{S_1, S_2, \ldots, S_N\}$ και $L=\{(S_i, S_{i+1}) \mid \forall i=1, \ldots, N-1\}$. Η τοπολογία του workflow και του δικτύου φαίνεται στο Σχήμα 4.1.



Σχήμα 4.1 Η Τοπολογία του Workflow και του Δικτύου Εξυπηρετητών όπου Χρησιμοποιείται ο Αλγόριθμος Γραμμή-Γραμμή.

Ο αλγόριθμος δουλεύει ως εξής: αρχικά, παίρνει μία-μία τις λειτουργίες του W και τις αναθέτει στους εξυπηρετητές, αρχίζοντας από τον αριστερότερο (S_1) και προχωρώντας προς τον δεξιότερο (S_N), ώστε η κατανομή του φορτίου να είναι δίκαιη και ο κάθε εξυπηρετητής να έχει τουλάχιστον μία λειτουργία. Με την ανάθεση αυτή κάθε εξυπηρετητής S_i στέλνει ακριβώς ένα μήνυμα στον δεξιότερό του S_{i+1}. Εδώ τελειώνει το πρώτο μέρος του αλγορίθμου. Στη συνέχεια, στο δεύτερο μέρος του αλγορίθμου, γίνεται η επιδιόρθωση των κακών γεφυρών. *Κακή γέφυρα* είναι μία σύνδεση του δικτύου η οποία έχει μικρή ταχύτητα ενώ το μήνυμα που περνάει διαμέσου αυτής είναι μεγάλο. Ο αλγόριθμος προσπαθεί να επιδιορθώσει τις κακές γέφυρες μετακινώντας λειτουργίες σε διπλανούς εξυπηρετητές με την προϋπόθεση ότι δεν θα μείνει κάποιος εξυπηρετητής χωρίς καμία λειτουργία. Ονομάζουμε *κρίσιμη γέφυρα* μία κακή γέφυρα που μπορεί να επιδιορθωθεί. Μία κρίσιμη γέφυρα έχει το χαρακτηριστικό ότι τουλάχιστον ένα διπλανό μήνυμα στο μήνυμα που στέλνεται από την κρίσιμη γέφυρα είναι μικρό και με τη μετακίνηση του μικρού μηνύματος στην κρίσιμη γέφυρα, η γέφυρα παύει να είναι κρίσιμη. Ένα παράδειγμα μιας κρίσιμης γέφυρας φαίνεται στο Σχήμα 4.2. Στην περίπτωση αυτή ο αλγόριθμος θα μετακινήσει την λειτουργία O_{11} από τον εξυπηρετητή S_5 στον εξυπηρετητή S_6, ή αλλιώς γίνεται

μία δεξιά μετακίνηση. Στην περίπτωση που το (O_{12}, O_{13}) ήταν μικρό μήνυμα, ή ήταν μικρότερο από το (O_{10}, O_{11}), τότε θα γινόταν αριστερή μετακίνηση και η O_{12} θα μεταφερόταν στον S_5.



Σχήμα 4.2 Παράδειγμα Κρίσιμης Γέφυρας.

Ένα θέμα που πρέπει να διευκρινιστεί είναι ο τρόπος που χαρακτηρίζονται τα μηνύματα σε μηνύματα μεγάλου και μικρού μεγέθους, όπως επίσης και οι συνδέσεις μικρής ταχύτητας. Ο τρόπος που γίνεται αυτό είναι με την ταξινόμηση των μηνυμάτων και των συνδέσεων σε δύο λίστες σύμφωνα με το μέγεθός τους και με την ταχύτητά τους αντίστοιχα κατά αύξουσα διάταξη. Έτσι, μικρού μεγέθους χαρακτηρίζονται τα μηνύματα που βρίσκονται στην κορυφή της αντίστοιχης λίστας, ενώ μεγάλου μεγέθους χαρακτηρίζονται τα μηνύματα που βρίσκονται στο τέλος της λίστας. Ομοίως, συνδέσεις μικρής ταχύτητας χαρακτηρίζονται οι συνδέσεις που βρίσκονται στην κορυφή της λίστας των συνδέσεων. Σημαντικό ρόλο παίζει το κατώφλι που καθορίζει ποια είναι η κορυφή και το τέλος των λιστών. Δηλαδή, το κατώφλι πρέπει να διαχωρίζει τα μικρά και τα μεγάλα μηνύματα καθώς και τις αργές από τις γρήγορες συνδέσεις. Για να γίνει αυτό πρέπει να γνωρίζουμε ποιο είναι το ποσοστό των μικρών και μεγάλων μηνυμάτων (αργών – γρήγορων συνδέσεων αντίστοιχα). Επειδή στην συγκεκριμένη εργασία αυτό το ποσοστό είναι το 25% χρησιμοποιούμε για κατώφλι το 20%.

Η φόρμουλα για τον υπολογισμό της κατανομής φορτίου στον αλγόριθμο Γραμμή-Γραμμή είναι η Εξ. 3.1, ενώ η φόρμουλα για τον υπολογισμό του χρόνου εκτέλεσης, σύμφωνα με την Εξ. 3.2 είναι η εξής:

$$T_{execute} = \sum_{i=1}^{M} T_{proc}(O_i) + \sum_{i=1}^{M-1} T_{comm}(O_i, O_{i+1}) \qquad \text{Εξ. 4.1}$$

Στη συνέχεια ακολουθεί ο ψευδοκώδικας του αλγορίθμου Γραμμή-Γραμμή. Η συνάρτηση Fix_Bad_Bridges υλοποιεί το δεύτερο μέρος του αλγορίθμου ενώ η συνάρτηση Is_Critical_Bridge είναι βοηθητική για τον έλεγχο αν μια γέφυρα είναι κρίσιμη.

Αλγόριθμος Γραμμή–Γραμμή (Ψευδοκώδικας)

Είσοδοι: Ένα workflow **W(O, E)** (με τοπολογία γραμμής), όπου $O = (O_1, O_2, ..., O_M)$ ένα σύνολο από λειτουργίες και $E = \{(O_i, O_{i+1}) \mid \forall\, i = 1,..., M-1\}$ το σύνολο των μεταβάσεων μεταξύ των λειτουργιών και ένα δίκτυο **N(S, L)** (σε τοπολογία γραμμής), όπου $S = \{S_1, S_2, ..., S_N\}$ ένα σύνολο από εξυπηρετητές και $L = \{(S_i, S_{i+1}) \mid \forall\, i = 1, ..., N-1\}$ οι συνδέσεις μεταξύ των εξυπηρετητών. Πρέπει **M>N**.

Έξοδος: Μία αντιστοίχιση **M** (Mapping) του **O** στο **S**.

Περιγραφή: Αρχικά, ο αλγόριθμος κατανέμει τις λειτουργίες έτσι ώστε το φορτίο του κάθε εξυπηρετητή να είναι ανάλογο με την υπολογιστική του ισχύ και κάθε εξυπηρετητής να στέλνει ακριβώς ένα μήνυμα στον επόμενό του στην γραμμή. Στη συνέχεια καλείται η συνάρτηση Fix_Bad_Bridges για τη μετακίνηση λειτουργιών έτσι ώστε να μην στέλνονται μεγάλα μηνύματα διαμέσου συνδέσεων με μικρή ταχύτητα.

Begin

1 Operations_List = $(O_1, O_2, ..., O_M)$

2 Servers_List = $(S_1, S_2, ..., S_N)$

3 Sum_Cycles = $\sum_{i=1}^{M} C(O_i)$ *// Οι συνολικοί κύκλοι που απαιτούν όλες οι λειτουργίες*

4 Sum_Capacity = $\sum_{j=1}^{N} P(S_j)$ *// Η συνολική υπολογιστική ισχύς των εξυπηρετητών*

5 M = ∅

6 s = Servers_List.pop

7 Ideal_Cycles = Sum_Cycles × P(s) / Sum_Capacity *// Ο ιδανικός (δίκαιος) φόρτος για*
// τον εξυπηρετητή s

8 Current_Cycles = 0 *// Οι κύκλοι που έχουν ανατεθεί στον s μέχρι στιγμής*

9 while Operations_List is not empty do

10 o = Operation_List.pop

11 if SizeOf(Operation_List) ≥ SizeOf(Servers_List) then

```
12              if Current_Cycles + C(o) < Ideal_Cycles + 0,2×Ideal_Cycles or
13               Current_Cycles = 0  or  s is S_N then
14                      Current_Cycles = Current_Cycles + C(o)
15              else
16                      s = Servers_List.pop
17                      Ideal_Cycles = Sum_Cycles × P(s) / Sum_Capacity
18                      Current_Cycles = C(o)
19              end if
20      M = M ∪ {o→s}
21      else
22              s = Servers_List.pop
23              M = M ∪ {o→s}
24              while Operation_List is not empty do
25                      o = Operation_List.pop
26                      s = Servers_List.pop
27                      M = M ∪ {o→s}
28              end while
29      end if
30  end while
31  Fix_Bad_Bridges (W, N, M);
32  return M
End
```

Στις πρώτες 8 γραμμές του αλγορίθμου Γραμμή–Γραμμή γίνονται οι απαραίτητες αρχικοποιήσεις. Οι λειτουργίες και οι εξυπηρετητές τοποθετούνται σε δύο λίστες, υπολογίζονται οι συνολικοί κύκλοι που απαιτούν οι λειτουργίες, η συνολική υπολογιστική ισχύς των εξυπηρετητών και το ιδανικό φορτίο που αντιστοιχεί σε κάθε εξυπηρετητή και η αντιστοίχιση **M** αρχικοποιείται στο κενό σύνολο. Στη συνέχεια (γραμμές 9-30) μία-μία λειτουργία του workflow (πρώτα η O_1 μετά η O_2 κ.ο.κ.), ανατίθεται στους εξυπηρετητές. Η σειρά των εξυπηρετητών όπου γίνεται η ανάθεση των λειτουργιών είναι αυτή που εμφανίζονται στην αντίστοιχη λίστα. Σε περίπτωση που το πλήθος των λειτουργιών που δεν έχουν ακόμη ανατεθεί είναι ίσο με το πλήθος των εξυπηρετητών που δεν έχουν αναλάβει καμία λειτουργία χρησιμοποιούνται οι γραμμές 21-29 όπου κάθε εξυπηρετητής αναλαμβάνει μία ακριβώς λειτουργία. Σε αντίθεση περίπτωση, χρησιμοποιούνται οι γραμμές 11-19. Σε αυτό το κομμάτι του κώδικα η λειτουργία που ανατίθεται στο συγκεκριμένο βήμα του αλγορίθμου, είτε

ανατίθεται στον εξυπηρετητή που έχει σειρά (γραμμές 12-14), είτε ανατίθεται στον αμέσως επόμενο εξυπηρετητή (γραμμές 16-18). Το πρώτο γίνεται αν ο εξυπηρετητής που έχει σειρά δεν καλύπτει το ιδανικό φορτίο που του αναλογεί, ή δεν το υπερβαίνει κατά 20% με την ανάθεση της λειτουργίας σε αυτόν (γραμμή 12), ή αν δεν έχει αναλάβει καμία λειτουργία, ή αν είναι ο τελευταίος εξυπηρετητής (γραμμή 13). Μετά το πρώτο μέρος του αλγορίθμου καλείται η συνάρτηση που κάνει την επιδιόρθωση των κακών γεφυρών (γραμμή 31).

Fix Bad Bridges Function

Είσοδος: Το workflow **W** και το δίκτυο **N** του αλγορίθμου Γραμμή-Γραμμή και μία αντιστοίχιση **M**.

Έξοδος: Η αντιστοίχιση **M** αλλαγμένη (αν γίνουν κάποιες αλλαγές).

Περιγραφή: Η συνάρτηση ελέγχει βήμα-βήμα τις συνδέσεις μεταξύ των εξυπηρετητών και αν είναι κρίσιμες γέφυρες τότε μετακινεί την κατάλληλη λειτουργία σε γειτονικό εξυπηρετητή ώστε το μικρό μήνυμα να στέλνεται διαμέσου της σύνδεσης της κρίσιμης γέφυρας. Απαραίτητη προϋπόθεση για την μετακίνηση της λειτουργίας είναι να μη μείνει κάποιος εξυπηρετητής χωρίς καμία λειτουργία. Η συνάρτηση για να βρει κρίσιμες γέφυρες, χρησιμοποιεί μια βοηθητική συνάρτηση, την **Is_Critical_Bridge** η οποία επιστέφει **True** ή **False** για το αν μια γέφυρα είναι κρίσιμη και στην περίπτωση που η γέφυρα είναι κρίσιμη επιστρέφει και την φορά που πρέπει να γίνει η μετακίνηση της λειτουργίας. Ο εξυπηρετητής S_1 θεωρείται ότι είναι αριστερά και ο S_N δεξιά. FirstOperationAt(s) είναι η λειτουργία που εκτελείται πρώτη στον εξυπηρετητή s, ενώ LastOperationAt(s) είναι η λειτουργία που εκτελείται τελευταία στον εξυπηρετητή s.

Begin

1 Sort the speeds of the lines of N at ongoing order in list L_1.
2 Sort the sizes of messages that are sending between servers (from last operation at server i to first operation at server i+1) at ongoing order in list L_2.
3 for i=1 to N-1 do
4 if Is_Critical_Bridge(S, M, i, L_1, L_2, shift_direction) = True then
5 if shift_direction = right then
6 M = M + {LastOperationAt(S_i)→S_{i+1}} – {LastOperationAt(S_i)→S_i}
7 else

```
8                    M = M + {FirstOperationAt(S_{i+1})→S_i} – {FirstOperationAt(S_{i+1})→S_{i+1}}
9                end if
10           end if
11   end for
End
```

Is Critical Bridge Function

Είσοδοι: Το workflow **W** και το δίκτυο **N** του αλγορίθμου Γραμμή-Γραμμή, μία αντιστοίχιση **M**, έναν ακέραιο "i" που αντιπροσωπεύει την γέφυρα πρέπει να ελεγχθεί και τις ταξινομημένες λίστες L_1 και L_2 με τις ταχύτητες των συνδέσεων και τα μεγέθη των μηνυμάτων που ανταλλάσσονται σε κάθε γέφυρα αντίστοιχα.

Έξοδος: Η συνάρτηση επιστρέφει True αν η γέφυρα μεταξύ του S_i και S_{i+1} είναι κρίσιμη, αλλιώς επιστρέφει False. Στην πρώτη περίπτωση ανατίθεται στην μεταβλητή shift_direction η φορά της μετακίνησης που πρέπει να γίνει.

Περιγραφή: Μία γέφυρα είναι κρίσιμη όταν ένα μεγάλο μήνυμα στέλνεται διαμέσου της συγκεκριμένης σύνδεσης και επιπλέον ένα μικρό μήνυμα βρίσκεται στη γειτονική ακμή. Αν το γειτονικό μικρό μήνυμα βρίσκεται στον S_{i+1} τότε η πρώτη λειτουργία του S_{i+1} πρέπει να μετακινηθεί αριστερά (στον S_i) ενώ αν βρίσκεται στον S_i τότε η τελευταία λειτουργία του S_i πρέπει να μετακινηθεί δεξιά (στον S_{i+1}). Πρώτη λειτουργία σε έναν εξυπηρετητή (FirstOperationAt) θεωρείται αυτή που εκτελείται πρώτη ενώ τελευταία (LastOperationAt) αυτή που εκτελείται τελευταία. Επίσης, SecondOperationAt(s) είναι η λειτουργία που εκτελείται δεύτερη στον εξυπηρετητή s και είναι αυτή που εκτελείται προτελευταία στον εξυπηρετητή s. Το TopA μιας λίστας είναι το N×A/100 στοιχείο της λίστας όπου N είναι το μέγεθος της λίστας. Το BottomA μιας λίστας είναι το N-N×A/100 όπου N είναι το μέγεθος της λίστας.

```
Begin
1   if Line_Speed(S_i, S_{i+1}) ≤ Top20 of L_1 and
       MsgSize(LastOperationAt(S_i), FirstOperationAt(S_{i+1})) ≥ Bottom20 of L_2 then
2      if MsgSize(PenultOperationAt(S_i), LastOperationAt(S_i)) ≤ Top20 of L_2 then
3            shift_direction = right
4            return True
5      end if
6      if   MsgSize(FirstOperationAt(S_{i+1}), SecondOperationAt(S_{i+1})) ≤ Top20 of L_2 then
7            shift_direction = left
```

```
8                return True
9        end if
10   end if
11   return False
End
```

4.2.1. Παραλλαγές αλγορίθμου Γραμμή–Γραμμή

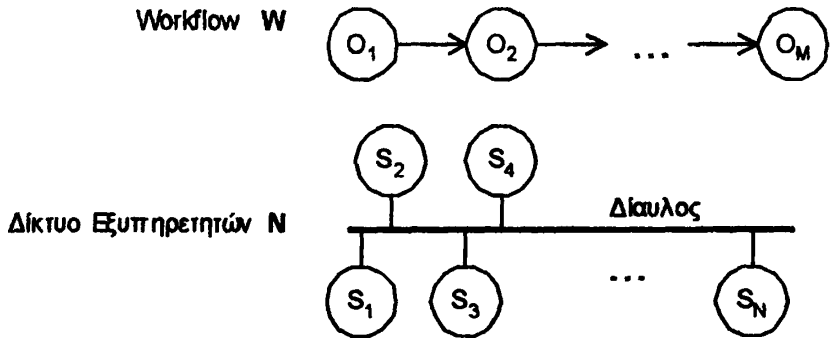
Το γεγονός ότι ο αλγόριθμος Γραμμή–Γραμμή χωρίζεται σε δύο διακριτά κομμάτια μας επιτρέπει να ορίσουμε δύο παραλλαγές του. Η πρώτη παραλλαγή είναι η εκτέλεση του πρώτου μέρους του αλγορίθμου χωρίς την επιδιόρθωση των κακών γεφυρών, ενώ η δεύτερη είναι η εκτέλεση του αλγορίθμου κάνοντας επιδιόρθωση των κακών γεφυρών (χρησιμοποιώντας τη συνάρτηση Fix_Bab_Bridges). Όπως περιγράφτηκε στην ενότητα 3.6. στον αλγόριθμο Γραμμή–Γραμμή, η ανάθεση των εξυπηρετητών γίνεται από τον αριστερότερο στον δεξιότερο. Η ανάθεση όμως μπορεί να γίνει και με τον αντίστροφο τρόπο, αρχίζοντας δηλαδή από τον δεξιότερο εξυπηρετητή και προχωρώντας προς τον αριστερότερο. Μια τρίτη λοιπόν παραλλαγή του αλγορίθμου είναι να εκτελείται το πρώτο μέρος του αλγορίθμου Γραμμή–Γραμμή με την ανάθεση των λειτουργιών και στις δύο κατευθύνσεις και να επιλέγεται η καλύτερη λύση. Αν στην παραπάνω εκδοχή κάνουμε επιπλέον και επιδιόρθωση των κακών γεφυρών προκύπτει και η τελευταία παραλλαγή. Συνοψίζοντας οι τέσσερις παραλλαγές του αλγορίθμου Γραμμή–Γραμμή είναι οι εξής:

- Μιας κατεύθυνσης Γραμμή–Γραμμή χωρίς επιδιόρθωση κακών γεφυρών (1d-NoFix)
- Μιας κατεύθυνσης Γραμμή–Γραμμή με επιδιόρθωση κακών γεφυρών (1d-WithFix)
- Διπλής κατεύθυνσης Γραμμή–Γραμμή χωρίς επιδιόρθωση κακών γεφυρών (2d-NoFix)
- Διπλής κατεύθυνσης Γραμμή–Γραμμή με επιδιόρθωση κακών γεφυρών (2d-WithFix)

4.3. Αλγόριθμοι Γραμμή–Δίαυλος

Όταν η τοπολογία του workflow του συστήματος είναι γραμμή και οι εξυπηρετητές του δικτύου συνδέονται μεταξύ τους μέσω ενός διαύλου, όπως φαίνεται στο Σχήμα 4.3, τότε χρησιμοποιούνται οι αλγόριθμοι που εντάσσονται στην κατηγορία Γραμμή–

Δίαυλος. Οι αλγόριθμοι βασίζονται κι αυτοί επίσης στις ιδιότητες που πρέπει να έχουν οι καλές λύσεις που περιγράφονται στην ενότητα 3.5. Για τον υπολογισμό της κατανομής του φορτίου και του χρόνου απόκρισης του workflow χρησιμοποιούνται και εδώ οι ίδιες φόρμουλες που χρησιμοποιούνται και στον αλγόριθμο Γραμμή–Γραμμή, δηλαδή οι εξισώσεις Εξ 3.1 και Εξ 4.1.



Σχήμα 4.3 Η Τοπολογία του Workflow και του Δικτύου Εξυπηρετητών όπου Χρησιμοποιούνται οι Αλγόριθμοι Γραμμή-Δίαυλος.

Ο πιο απλοϊκός από τους αλγορίθμους, ο Fair Load, προσπαθεί να κατανείμει τις λειτουργίες στους εξυπηρετητές ώστε οι κύκλοι που αναλαμβάνει κάθε εξυπηρετητής να είναι ανάλογοι με την υπολογιστική του ισχύ. Προσπαθεί δηλαδή να ικανοποιήσει την πρώτη από τις τρεις ιδιότητες της ενότητας 3.4 αδιαφορώντας για τις άλλες δύο. Από τον αλγόριθμο αυτό προκύπτουν άμεσα άλλοι δύο οι οποίοι κάνουν ότι ακριβώς και ο Fair Load με τη διαφορά ότι επιλέγουν με πιο έξυπνο τρόπο μία ανάθεση από ένα σύνολο αναθέσεων οι οποίες είναι ισοδύναμες στον Fair Load. Ο πρώτος από αυτούς ονομάζεται Fair Load–Tie Resolver$_1$ και χειρίζεται τις λειτουργίες που έχουν τον ίδιο αριθμό κύκλων. Ο δεύτερος ονομάζεται Fair Load–Tie Resolver$_2$ και χειρίζεται εκτός από τις λειτουργίες που έχουν τον ίδιο αριθμό κύκλων και τους εξυπηρετητές που μπορούν να εξυπηρετήσουν τον ίδιο αριθμό κύκλων. Δηλαδή, οι δύο τελευταίοι αλγόριθμοι επιλέγουν μία ανάθεση μεταξύ των αναθέσεων που είναι ίδιες για τον Fair Load. Η επιλογή αυτή βασίζεται στην δεύτερη ιδιότητα της ενότητας 3.4, ότι δηλαδή πρέπει να στέλνονται μηνύματα μικρού μεγέθους.

Ο αλγόριθμος Fair Load–Merge Messages' Ends δουλεύει όπως και Fair Load–Tie Resolver$_2$ με τη διαφορά ότι κάνει ένα επιπλέον έλεγχο κατά την ανάθεση λειτουργιών

στους εξυπηρετητές. Αν η ανάθεση μίας λειτουργίας αναγκάζει ένα μεγάλο μήνυμα να σταλεί μέσω του διαύλου τότε η ανάθεση αυτή ακυρώνεται και η λειτουργία ανατίθεται στον εξυπηρετητή εκείνο ώστε το μεγάλο μήνυμα να μην στέλνεται μέσω του διαύλου.

Τέλος, ο αλγόριθμος Heavy Operations–Large Messages αναθέτει λειτουργίες σε εξυπηρετητές παρόμοια με τον Fair Load με τη διαφορά ότι δεν χειρίζεται τις λειτουργίες μία-μία ξεχωριστά αλλά τις χειρίζεται σαν ομάδες από λειτουργίες. Στην ίδια ομάδα μπαίνουν λειτουργίες που ανταλλάσσουν μεγάλα μηνύματα μεταξύ τους. Ένα μήνυμα θεωρείται μεγάλο όταν ο χρόνος για να σταλεί μέσω του διαύλου είναι μεγαλύτερος από τον χρόνο επεξεργασίας των λειτουργιών της ομάδας με τους περισσότερους συνολικά κύκλους στον εξυπηρετητή με τους περισσότερους διαθέσιμους κύκλους τη δεδομένη χρονική στιγμή. Οι λειτουργίες μίας ομάδας πρέπει σε επόμενη ανάθεση να τοποθετηθούν στον ίδιο εξυπηρετητή.

Η πολυπλοκότητα του αλγορίθμου Fair Load είναι O(M×logM+N×logN+MN). Οι πρώτοι δύο όροι της πολυπλοκότητας προκύπτουν από την ταξινόμηση των λειτουργιών και των εξυπηρετητών αντίστοιχα, ενώ ο τρίτος όρος προκύπτει από την ανάθεση των λειτουργιών μία-μία. Το N στον τελευταίο όρο προκύπτει από την τοποθέτηση του εξυπηρετητή στον οποίο γίνεται η ανάθεση στην σωστή θέση στη λίστα των εξυπηρετητών. Για τους υπόλοιπους αλγορίθμους Γραμμή–Δίαυλος μπορούμε να δώσουμε μία πολυπλοκότητα που να τους καλύπτει όλους. Η πολυπλοκότητα που αντιπροσωπεύει το πάνω όριο της εκτέλεσης αυτών των αλγορίθμων είναι O(M×(M×logM+N×logN+MN)). Ο πρώτος παράγοντας προκύπτει από την ανάθεση όλων των λειτουργιών μία-μία. Πριν από την ανάθεση κάθε λειτουργίας γίνονται τα εξής: ταξινομούνται οι λειτουργίες (M×logM), ταξινομούνται οι εξυπηρετητές (N×logN) και υπολογίζεται το κέρδος των αναθέσεων των λειτουργιών που δεν έχουν ανατεθεί σε όλους τους εξυπηρετητές (MN). Έτσι προκύπτει ο δεύτερος παράγοντας της πολυπλοκότητας. Στον αλγόριθμο Heavy Operations–Large Messages, ο όρος MN γίνεται 1 γιατί γίνεται μόνο ένας απλός έλεγχος. Στη συνέχεια περιγράφονται πιο τυπικά οι αλγόριθμοι και δίνεται ο ψευδοκώδικάς τους.

Αλγόριθμος Fair Load

Είσοδοι: Ένα workflow **W(O, E)** (με τοπολογία γραμμής), όπου $O = (O_1, O_2, ..., O_M)$ ένα σύνολο από λειτουργίες και $E = \{(O_i, O_{i+1}) \mid \forall\ i = 1, ..., M-1\}$ το σύνολο των μεταβάσεων μεταξύ των λειτουργιών και ένα δίκτυο **N(S, L)** (σε τοπολογία διαύλου), όπου $S = \{S_1, S_2, ..., S_N\}$ ένα σύνολο από εξυπηρετητές και $L = \{(S_i, S_j) \mid \forall\ i, j = 1, ..., N, i \neq j\}$ οι συνδέσεις μεταξύ των εξυπηρετητών. Πρέπει **M>N**.

Έξοδος: μία αντιστοίχιση (Mapping) **M** του **O** στο **S**.

Περιγραφή: Ο αλγόριθμος αρχικά υπολογίζει του κύκλους που αναλογούν ακριβώς σε κάθε εξυπηρετητή για την βέλτιστη κατανομή του φορτίου και στη συνέχεια προσπαθεί να αναθέσει τις λειτουργίες στους εξυπηρετητές ώστε να προσεγγισθεί αυτή η βέλτιστη κατανομή. Αυτό γίνεται αναθέτοντας πρώτα τις πιο βαριές (με πολλούς κύκλους) λειτουργίες και στη συνέχεια τις πιο ελαφριές (με λίγους κύκλους). Η ανάθεση γίνεται με προτεραιότητα στους εξυπηρετητές που τη χρονική στιγμή της ανάθεσης χρειάζονται τους περισσότερους κύκλους για να συμπληρώσουν τους κύκλους που τους αντιστοιχούν στη βέλτιστη κατανομή. Ο τρόπος αυτός, με τον οποίο γίνεται η ανάθεση των λειτουργιών, βασίζεται σε έναν από τους ευρεστικούς αλγορίθμους *(Worst Fit)* που προτείνονται για το πρόβλημα του *bin packing*. Το πρόβλημα bin packing είναι ίδιο με το πρόβλημα που λύνει ο Fair Load, με τη διαφορά ότι σύμφωνα με το πρώτο, θα προσπαθούσαμε να ελαχιστοποιήσουμε τον αριθμό των εξυπηρετητών ενώ στο δεύτερο ο αριθμός των εξυπηρετητών είναι συγκεκριμένος. Ο αλγόριθμος χρησιμοποιεί δύο λίστες (Servers_List και Operations_List) με δείκτες προς τους εξυπηρετητές και τις λειτουργίες.

Begin

1. $\text{Sum\_Cycles} = \sum_{i=1}^{M} C(O_i)$ *// Οι συνολικοί κύκλοι που απαιτούν όλες οι λειτουργίες*
2. $\text{Sum\_Capacity} = \sum_{i=1}^{N} P(S_i)$ *// Η συνολική υπολογιστική ισχύς των εξυπηρετητών*
3. $\text{Ideal\_Cycles}(S_i) = \text{Sum\_Cycles} \times \frac{P(S_i)}{\text{Sum\_Capacity}},\ \forall\ i = 1, ..., N$
4. $\text{Servers\_List} = (s_1, s_2, ..., s_N) = (S_1, S_2, ..., S_N)$ *// Λίστα με δείκτες προς εξυπηρετητές*
5. $\text{Operations\_List} = (o_1, o_2, ..., o_M) = (O_1, O_2, ..., O_M)$ *// Λίστα με δείκτες προς λειτουργίες*
6. Sort Servers_List so that $\forall\ i = 1, ..., N-1$ $\text{Ideal\_Cycles}(s_i) \geq \text{Ideal\_Cycles}(s_{i+1})$
7. Sort Operations_List so that $\forall\ i = 1, ..., M-1$ $C(o_i) \geq C(o_{i+1})$

8. $M = \varnothing$
9. for i=1 to M do
10. $M = M \cup \{o_i \rightarrow s_1\}$
11. $Ideal\_Cycles(s_1) - = C(o_i)$
12. Move s_1 in Servers_List so that $\forall$ i = 1, ..., N-1: $Ideal\_Cycles(i) \geq Ideal\_Cycles(i+1)$
13. Continue with new Servers_List = $(s_1, s_2, \ldots, s_N)$
14. end for
15. return M

End

Ο αλγόριθμος Fair Load αρχικά υπολογίζει τους συνολικούς κύκλους των λειτουργιών και την συνολική υπολογιστική ισχύ των εξυπηρετητών (γραμμές 1-2). Έπειτα, για κάθε εξυπηρετητή, υπολογίζει τους κύκλους που του αναλογούν για την ιδανική κατανομή φορτίου (γραμμή 3). Στη συνέχεια, ταξινομεί σε δύο λίστες, τις λειτουργίες με βάση τους υπολογιστικούς κύκλους και τους εξυπηρετητές με βάση τους υπολογιστικούς κύκλους που τους αναλογούν (γραμμές 4-7). Αφού αρχικοποιήσει την αντιστοίχιση M στο κενό σύνολο κανόνων (γραμμή 8), παίρνει μία-μία τις λειτουργίες από την αντίστοιχη λίστα και τις αναθέτει στον εξυπηρετητή που βρίσκεται στην κορυφή της λίστας των εξυπηρετητών (γραμμή 10). Μετά από κάθε ανάθεση ο εξυπηρετητής μετακινείται στη λίστα των εξυπηρετητών ώστε η λίστα να παραμένει ταξινομημένη (γραμμές 11-14).

Το πρόβλημα του Fair Load είναι ότι δε λαμβάνει καθόλου υπόψη το χρόνο εκτέλεσης του workflow. Αυτό προσπαθούν να βελτιώσουν οι δύο επόμενοι αλγόριθμοι, Fair Load–Tie Resolver$_1$ και Fair Load–Tie Resolver$_2$, οι οποίοι επιλύουν ισοπαλίες που προκύπτουν στον Fair Load, στην επιλογή αναθέσεων, οι οποίες οδηγούν στην ίδια κατανομή φορτίου αλλά σε διαφορετικό χρόνο εκτέλεσης. Ο πρώτος επιλύει ισοπαλίες που προκύπτουν όταν πολλές λειτουργίες έχουν τον ίδιο αριθμό κύκλων, ενώ ο δεύτερος, επιλύει επιπλέον ισοπαλίες που προκύπτουν όταν πολλοί εξυπηρετητές μπορούν να αναλάβουν τους ίδιους υπολογιστικούς κύκλους.

Αλγόριθμος Fair Load – Tie Resolver$_1$

Είσοδοι: Ένα workflow W(O, E) (με τοπολογία γραμμής), όπου $O = (O_1, O_2, \ldots, O_M)$ ένα σύνολο από λειτουργίες και $E = \{(O_i, O_{i+1}) \mid \forall i = 1, \ldots, M-1\}$ το σύνολο των μεταβάσεων

μεταξύ των λειτουργιών και ένα δίκτυο N(S, L) (σε τοπολογία διαύλου), όπου $S = \{S_1, S_2, ..., S_N\}$ ένα σύνολο από εξυπηρετητές και $L = \{(S_i, S_j) \mid \forall\ i, j = 1, ..., N, i \neq j\}$ οι συνδέσεις μεταξύ των εξυπηρετητών. Πρέπει M>N.

Έξοδος: μία αντιστοίχιση (Mapping) M του O στο S.

Περιγραφή: Ο αλγόριθμος αυτός δουλεύει όπως ακριβώς και ο Fair Load με μία διαφορά στην επιλογή της λειτουργίας που πρόκειται να ανατεθεί. Και εδώ επιλέγονται πρώτα οι πιο βαριές λειτουργίες και μετά οι ελαφρότερες με τη διαφορά ότι γίνεται μία ακόμη επιπλέον επιλογή μεταξύ λειτουργιών με ίδιους κύκλους. Η επιλογή αυτή γίνεται με βάση το κέρδος που έχουμε κάνοντας κάθε επιλογή, χρησιμοποιώντας τη συνάρτηση Gain_Of_Put_Operation_At_Server. Για να δίνει η παραπάνω συνάρτηση ένα κέρδος σε κάθε κλήση της, η αντιστοίχιση M αρχικοποιείται σε μια τυχαία αντιστοίχιση. Σε αντίθετη περίπτωση, η συνάρτηση δεν θα έδινε κέρδος σε κλήσεις της στις πρώτες αναθέσεις γιατί δεν θα υπήρχαν μηνύματα στο δίαυλο (αφού δεν θα είχαν ανατεθεί ακόμη λειτουργίες) και η συνάρτηση δεν μπορεί να υπολογίσει κέρδος χωρίς μηνύματα. Ο αλγόριθμος χρησιμοποιεί δύο λίστες (Servers_List και Operations_List) με δείκτες προς τους εξυπηρετητές και τις λειτουργίες.

Begin

1. $Sum\_Cycles = \sum_{i=1}^{M} C(O_i)$ *// Οι συνολικοί κύκλοι που απαιτούν όλες οι λειτουργίες*
2. $Sum\_Capacity = \sum_{i=1}^{N} P(S_i)$ *// Η συνολική υπολογιστική ισχύς των εξυπηρετητών*
3. $Ideal\_Cycles(S_i) = Sum\_Cycles \times \frac{P(S_i)}{Sum\_Capacity}, \forall\ i = 1, ..., N$
4. $Servers\_List = (s_1, s_2, ..., s_N) = (S_1, S_2, ..., S_N)$ *// Λίστα με δείκτες προς εξυπηρετητές*
5. $Operations\_List = (o_1, o_2, ..., o_M) = (O_1, O_2, ..., O_M)$ *// Λίστα με δείκτες προς λειτουργίες*
6. Sort Servers_List so that $\forall\ i = 1, ..., N-1$ $Ideal\_Cycles(s_i) \geq Ideal\_Cycles(s_{i+1})$
7. Sort Operations_List so that $\forall\ i = 1, ..., M-1$ $C(o_i) \geq C(o_{i+1})$
8. Initialize M to a random Mapping
9. while Operations_List is not empty do *// μέχρι να ανατεθούν όλες οι λειτουργίες*
10. $gain_1 = Gain\_Of\_Put\_Operation\_At\_Server(o_1, s_1, M)$
11. i=2
12. while $C(o_1) = C(o_i)$ and $i \leq M$ do
13. $gain_2 = Gain\_Of\_Put\_Operation\_At\_Server(o_i, s_1, M)$
14. if $gain_2 > gain_1$

15. swap(o_1, o_i)
16. $gain_1 = gain_2$
17. end if
18. i++
19. end while
20. $M = M - \{o_1 \rightarrow Server(o_1)\}$ // ακύρωση της τυχαίας ανάθεσης της 1ης
21. $M = M \cup \{o_1 \rightarrow s_1\}$ // λειτουργίας της Operations_List και ανάθεσή της
22. Delete o_1 from Operations_List // στον εξυπηρ. που βρίσκεται 1ος στην Servers_List
23. $Ideal\_Cycles(s_1) - = C(o_1)$
24. Move s_1 in Servers_List so that $\forall$ i = 1, ..., N-1: $Ideal\_Cycles(i) \geq Ideal\_Cycles(i+1)$
25. Continue with new Servers_List = (s_1, s_2,..., s_N)
26. end while
27. return M

End

Οι πρώτες 7 γραμμές του Fair Load–Tie Resolver$_1$ είναι ίδιες με αυτές του Fair Load. Στη γραμμή 8 αρχικοποιεί την αντιστοίχιση **M** με μία τυχαία αντιστοίχιση. Στη συνέχεια διαφέρει από τον Fair Load στο ότι υπολογίζει το κέρδος από τις αναθέσεις λειτουργιών με ίδιο αριθμό κύκλων. Οι αναθέσεις αυτές οδηγούν σε ίδια κατανομή του φορτίου αλλά σε διαφορετικό χρόνο εκτέλεσης. Αφού υπολογίσει την ανάθεση με το μεγαλύτερο κέρδος και τοποθετήσει την λειτουργία, η οποία αντιστοιχεί σε αυτή την ανάθεση, στην κορυφή της λίστας των λειτουργιών (γραμμές 10-19), ο αλγόριθμος συνεχίζει όπως ο Fair Load. Δηλαδή, αναθέτει την λειτουργία που βρίσκεται στην κορυφή της λίστας των λειτουργιών στον εξυπηρετητή που βρίσκεται στην κορυφή της λίστας των εξυπηρετητών (γραμμές 21-25), αφού προηγουμένως ακυρώσει την ανάθεση της συγκεκριμένης λειτουργίας που έγινε κατά την αρχικοποίηση (γραμμή 20).

Συνάρτηση Gain_Of_Put_Operation_At_Server

Είσοδοι: Μία λειτουργία $O_i \in O$, έναν εξυπηρετητή $S_j \in S$ και μία αντιστοίχιση $M \subseteq O \times S$.

Έξοδος: Ένας πραγματικός αριθμός που εκφράζει το κέρδος από την ανάθεση της λειτουργίας O_i στον εξυπηρετητή S_j.

Περιγραφή: Η συνάρτηση θεωρεί ως κέρδος τα bytes που δεν θα σταλούν μέσω του διαύλου , με βάση την αντιστοίχιση **M**, αν γίνει η ανάθεση της O_i στον εξυπηρετητή S_j.

Δηλαδή στην περίπτωση που στον εξυπηρετητή S_j είναι τοποθετημένη η O_{i-1} τότε κερδίζουμε την αποστολή του μηνύματος (O_{i-1}, O_i) και παρομοίως, αν και η O_{i+1} είναι τοποθετημένη στον εξυπηρετητή S_j τότε κερδίζουμε την αποστολή του μηνύματος (O_i, O_{i+1}).

```
Begin
1. gain = 0
2. if Oi ∈ (O2, O3, ..., OM) and {Oi-1→Sj} ∈ M then
3.       gain += MsgSize(Oi-1, Oi)
4. end if
5. if Oi ∈ (O1, O2,..., OM-1) and {Oi+1→Sj} ∈ M then
6.       gain += MsgSize(Oi, Oi+1)
7. end if
8. return gain
End
```

Αλγόριθμος Fair Load – Tie Resolver$_2$

Είσοδοι: Ένα workflow W(O, E) (με τοπολογία γραμμής), όπου $O = (O_1, O_2, ..., O_M)$ ένα σύνολο από λειτουργίες και $E = \{(O_i, O_{i+1}) \mid \forall\ i = 1, ..., M-1\}$ το σύνολο των μεταβάσεων μεταξύ των λειτουργιών και ένα δίκτυο N(S, L) (σε τοπολογία διαύλου), όπου $S = \{S_1, S_2, ..., S_N\}$ ένα σύνολο από εξυπηρετητές και $L=\{(S_i, S_j) \mid \forall\ i, j = 1, ..., N, i \neq j\}$ οι συνδέσεις μεταξύ των εξυπηρετητών. Πρέπει M>N.

Έξοδος: μία αντιστοίχιση (Mapping) M του O στο S.

Περιγραφή: Ο αλγόριθμος δουλεύει όπως και ο Fair Load–Tie Resolver$_1$ με μια διαφορά στην επιλογή του εξυπηρετητή όπου πρόκειται να γίνει η ανάθεση. Ο εξυπηρετητής που επιλέγεται πρώτος, όπως και στους προηγούμενους αλγορίθμους, είναι αυτός που την δεδομένη χρονική στιγμή μπορεί να αναλάβει περισσότερους κύκλους. Ο συγκεκριμένος αλγόριθμος όμως κάνει μία επιπλέον επιλογή όταν υπάρχουν περισσότεροι του ενός τέτοιοι εξυπηρετητές. Όπως και στον Fair Load–Tie Resolver$_1$, η επιλογή αυτή γίνεται με βάση το κέρδος που έχουμε κάνοντας κάθε επιλογή, χρησιμοποιώντας τη συνάρτηση Gain_Of_Put_Operation_At_Server. Ο αλγόριθμος χρησιμοποιεί δύο λίστες (Servers_List και Operations_List) με δείκτες προς τους εξυπηρετητές και τις λειτουργίες.

```
Begin
```

1. Sum_Cycles = $\sum_{i=1}^{M} C(O_i)$ *// Οι συνολικοί κύκλοι που απαιτούν όλες οι λειτουργίες*
2. Sum_Capacity = $\sum_{i=1}^{N} P(S_i)$ *// Η συνολική υπολογιστική ισχύς των εξυπηρετητών*
3. Ideal_Cycles(S_i) = Sum_Cycles × $\frac{P(S_i)}{Sum\_Capacity}$, ∀ i = 1, ..., N
4. Servers_List = (s_1, s_2, ..., s_N) = (S_1, S_2, ..., S_N) *// Λίστα με δείκτες προς εξυπηρετητές*
5. Operations_List = (o_1, o_2, ..., o_M) = (O_1, O_2, ..., O_M) *// Λίστα με δείκτες προς λειτουργίες*
6. Not_Assigned_Operations = M
7. Initialize M to a random Mapping
8. while Not_Assigned_Operations ≠ 0 do
9. Sort Servers_List so that ∀ i = 1, ..., N-1 Ideal_Cycles(s_i) ≥ Ideal_Cycles(s_{i+1})
10. Sort Operations_List so that ∀ i = 1, ..., Not_Assigned_Operations-1 C(o_i) ≥ C(o_{i+1})
11. best_gain = 0;
12. for each operation o_i in Operations_List where C(o_i) = C(o_1) do
13. for each server s_j in Servers_List where Ideal_Cycles(s_j)=Ideal_Cycles(s_1) do
14. gain = Gain_Of_Put_Operation_At_Server(o_i, s_j, M)
15. if gain > best_gain then
16. best_gain = gain
17. bestO = o_i
18. bestS = s_j
19. end if
20. end for
21. end for
22. M = M – {bestO→Server(bestO)} // ακύρωση της τυχαίας ανάθεσης της bestO και
23. M = M ∪ {bestO→bestS} // ανάθεσή της στον εξυπηρετητή bestS
24. Not_Assigned_Operations – –
25. Ideal_Cycles(bestS) – = C(bestO)
26. Delete bestO from Operation_List
27. end while
28. return M

End

Ο ψευδοκώδικας του Fair Load–Tie Resolver$_2$ είναι ίδιος με αυτόν του Fair Load–Tie Resolver$_1$, με τη διαφορά ότι υπολογίζεται κέρδος για τις αναθέσεις λειτουργιών με ίσους υπολογιστικούς κύκλους, σε όλους τους εξυπηρετητές που μπορούν να αναλάβουν ίσο αριθμό υπολογιστικών κύκλων (γραμμές 12-21). Αφού βρεθεί η

ανάθεση (bestO→bestS) με το μεγαλύτερο κέρδος (best_gain), γίνεται η συγκεκριμένη ανάθεση, όπως και στους προηγούμενους αλγορίθμους (γραμμές 22-26).

Το πρόβλημα με τους προηγούμενους αλγορίθμους, είναι ότι μπορεί κάποιο μεγάλο μήνυμα να στέλνεται μέσω του διαύλου και να αυξάνεται έτσι ο χρόνος εκτέλεσης. Αυτό προσπαθεί να βελτιώσει ο Fair Load–Merge Messages' Ends, ο οποίος αποτρέπει την αποστολή μεγάλων μηνυμάτων μέσω του διαύλου.

Αλγόριθμος Fair Load – Merge Messages' Ends

Είσοδοι: Ένα workflow W(O, E) (με τοπολογία γραμμής), όπου $O = (O_1, O_2, \ldots, O_M)$ ένα σύνολο από λειτουργίες και $E = \{(O_i, O_{i+1}) \mid \forall\ i = 1, \ldots, M-1\}$ το σύνολο των μεταβάσεων μεταξύ των λειτουργιών και ένα δίκτυο N(S, L) (σε τοπολογία διαύλου), όπου $S = \{S_1, S_2, \ldots, S_N\}$ ένα σύνολο από εξυπηρετητές και $L = \{(S_i, S_j) \mid \forall\ i, j = 1, \ldots, N, i \neq j\}$ οι συνδέσεις μεταξύ των εξυπηρετητών. Πρέπει M>N.

Έξοδος: μία αντιστοίχιση (Mapping) M του O στο S.

Περιγραφή: Ο αλγόριθμος κάνει αναθέσεις λειτουργιών σε εξυπηρετητές όπως ακριβώς και ο Fair Load–Tie Resolver$_2$. Στην περίπτωση όμως που μία τέτοια ανάθεση παραβιάζει ένα περιορισμό, γίνεται μία διαφορετική ανάθεση. Ο περιορισμός αφορά τα μηνύματα που θα ανταλλαγούν μέσω του διαύλου αν γίνει η συγκεκριμένη ανάθεση ενώ αφορούν τη λειτουργία που ανατίθεται και τις γειτονικές της στο workflow. Όταν με την ανάθεση, κάποιο από τα μηνύματα που δέχεται ή στέλνει η λειτουργία που πρόκειται να ανατεθεί, είναι μεγάλο και στέλνεται μέσω του διαύλου, τότε υπάρχει περιορισμός και η ανάθεση δεν μπορεί να γίνει. Ο έλεγχος για περιορισμούς γίνεται με τη συνάρτηση There_Is_Constraints. Αν ο περιορισμός αφορά το μήνυμα που δέχεται η λειτουργία επιλύεται ως εξής: αναθέτουμε τη λειτουργία στον εξυπηρετητή όπου βρίσκεται η λειτουργία που στέλνει το μήνυμα. Αν ο περιορισμός αφορά το μήνυμα που στέλνει η λειτουργία επιλύεται ως εξής: αναθέτουμε τη λειτουργία στον εξυπηρετητή όπου βρίσκεται η λειτουργία που δέχεται το μήνυμα. Ο αλγόριθμος χρησιμοποιεί τρεις λίστες (Servers_List, Operations_List και Messages_List) με δείκτες προς τους εξυπηρετητές, τις λειτουργίες και τα μηνύματα.

Begin

1. $Sum\_Cycles = \sum_{i=1}^{M} C(O_i)$ *// Οι συνολικοί κύκλοι που απαιτούν όλες οι λειτουργίες*
2. $Sum\_Capacity = \sum_{i=1}^{N} P(S_i)$ *// Η συνολική υπολογιστική ισχύς των εξυπηρετητών*
3. $Ideal\_Cycles(S_i) = Sum\_Cycles \times \frac{P(S_i)}{Sum\_Capacity}, \forall\, i = 1, \ldots, N$
4. $Servers\_List = (s_1, s_2, \ldots, s_N) = (S_1, S_2, \ldots, S_N)$ *// Λίστα με δείκτες προς εξυπηρετητές*
5. $Operations\_List = (o_1, o_2, \ldots, o_M) = (O_1, O_2, \ldots, O_M)$ *// Λίστα με δείκτες προς λειτουργίες*
6. $Messages\_List = (m_1, m_2, \ldots, m_{M-1}) = ((O_1, O_2), (O_2, O_3), \ldots, (O_{M-1}, O_M))$
7. Sort Messages_List so that $\forall\, i = 1, \ldots, M\text{-}2$ $MsgSize(m_i) \geq MsgSize(m_{i+1})$
8. Not_Assigned_Operations = M
9. Initialize M to a random Mapping
10. while Not_Assigned_Operations ≠ 0 do
11. Sort Servers_List so that $\forall\, i = 1, \ldots, N\text{-}1$ $Ideal\_Cycles(s_i) \geq Ideal\_Cycles(s_{i+1})$
12. Sort Operations_List so that $\forall\, i = 1, \ldots,$ Not_Assigned_Operations-1 $C(o_i) \geq C(o_{i+1})$
13. best_gain = 0;
14. for each operation o_i in Operations_List where $C(o_i) = C(o_1)$ do
15. for each server s_j in Servers_List where $Ideal\_Cycles(s_j) = Ideal\_Cycles(s_1)$ do
16. gain = Gain_Of_Put_Operation_At_Server(o_i, s_j, M)
17. if gain > best_gain then
18. best_gain = gain
19. bestO = o_i
20. bestS = s_j
21. end if
22. end for
23. end for
24. M = M – {bestO→Server(bestO)} // ακύρωση της τυχαίας ανάθεσης της bestO
25. if There_Is_Constraints (bestO, bestS, M, $MsgSize(m_{(M-1)\times 0.1})$, constraints_flag) then
26. if constraints_flag = left_message then
27. M = M ∪ {bestO→ServerOf(LeftOperationOf(bestO)}
28. Ideal_Cycles(ServerOf(LeftOperationOf(bestO) – = C(bestO)
29. else *// constraints_flag = right_message*
30. M = M ∪ {bestO→ServerOf(RightOperationOf(bestO)}
31. Ideal_Cycles(ServerOf(RightOperationOf(bestO) – = C(bestO)
32. end if
33. else *// Ανάθεση ακριβώς όπως και στον Fair Load-Tie Resolver 2*
34. M = M ∪ {bestO→bestS}
35. Ideal_Cycles(bestS) – = C(bestO)

```
36.     end if
37.     Not_Assigned_Operations – –
38.     Delete bestO from Operation_List
39. end while
40. return M
End
```

Οι πρώτες 23 γραμμές του ψευδοκώδικα του Fair Load–Merge Messages' Ends είναι ίδιες με αυτές του Fair Load–Tie Resolver$_2$, με τη διαφορά ότι ο πρώτος χρησιμοποιεί και μία λίστα με τα μηνύματα ταξινομημένα ως προς το μέγεθός τους (γραμμές 6-7). Στην γραμμή 25 γίνεται έλεγχος για το αν η ανάθεση που πρόκειται να γίνει με βάση τον Fair Load–Tie Resolver$_2$ δεν παραβιάζει κάποιον περιορισμό. Αν δεν παραβιάζεται κάποιος περιορισμός, γίνεται κανονικά η ανάθεση στις γραμμές 33-36. Αλλιώς, η λειτουργία που πρόκειται να ανατεθεί, τοποθετείται στον εξυπηρετητή που βρίσκεται η λειτουργία του άλλου άκρου του μεγάλου μηνύματος, το οποίο παραβίασε τον περιορισμό (γραμμές 26-32). Οι γραμμές 27-28 εκτελούνται όταν η λειτουργία που πρόκειται να ανατεθεί δέχεται το μεγάλο μήνυμα, ενώ οι γραμμές 30-31 όταν η λειτουργία στέλνει το μεγάλο μήνυμα.

Συνάρτηση There_Is_Constraints

Είσοδοι: Μία λειτουργία $O_i \in O$, έναν εξυπηρετητή $S_j \in S$, έναν αριθμό big_message_size που ορίζει το μέγεθος ενός μεγάλου μηνύματος και ένα σύνολο κανόνων $M \subseteq O \times S$.

Έξοδος: Επιστρέφει True αν υπάρχει κάποιος περιορισμός στο να γίνει η ανάθεση της λειτουργίας O_i στον εξυπηρετητή S_j, διαφορετικά επιστρέφει False. Στην πρώτη περίπτωση επιστρέφει και ένα "flag" (constraints_flag) με τιμές left_message και right_message το οποίο επισημαίνει ποιο από τα δύο μηνύματα: (O_{i-1}, O_i) και (O_i, O_{i+1}) αντίστοιχα, ενεργοποιεί τον περιορισμό. Στην περίπτωση όπου και τα δύο μηνύματα ενεργούν τον περιορισμό τότε το constraints_flag παίρνει την τιμή του μηνύματος το οποίο παραβιάζει περισσότερο την συνθήκη του περιορισμού.

Περιγραφή: Στην περίπτωση που η $O_i=O_1$ ή $O_i=O_M$ τότε ελέγχεται αν υπάρχει περιορισμός μόνο σε ένα μήνυμα (είτε στο (O_1, O_2) στην πρώτη περίπτωση είτε στο

((O_{M-1}, O_M)) στη δεύτερη περίπτωση). Διαφορετικά ελέγχονται και τα δύο μηνύματα, αυτό που στέλνει και αυτό που δέχεται η O_i.

```
Begin
1.  if Oi = O1 then
2.      if MsgSize(O1, O2) ≥ big_message_size then
3.          constraints_flag = right_message
4.          return True
5.      end if
6.  else if Oi = OM then
7.      if MsgSize(OM-1, OM) ≥ big_message_size then
8.          constraints_flag = left_message
9.          return True
10.     end if
11. else if Oi ∈ (O2, O3,..., OM-1) then
12.     if MsgSize(Oi-1, Oi) ≥ big_message_size then
13.         constraints_flag = left_message
14.     end if
15.     if MsgSize(Oi, Oi+1) ≥ big_message_size then
16.         if constraints_flag = left_message and MsgSize(Oi-1,Oi) ≥ MsgSize(Oi,Oi+1)
17.             constraints_flag = left_message
18.             return True
19.         else
20.             constraints_flag = right_message
21.             return True
22.         end if
23.     end if
24. end if
25. return False
End
```

Στις πρώτες 5 γραμμές της συνάρτηση There_Is_Constraints ελέγχεται αν το μήνυμα που στέλνει η O_1 είναι μεγάλο, ενώ στις επόμενες 5 γραμμές ελέγχεται αν το μήνυμα που δέχεται η O_M είναι μεγάλο. Οι υπόλοιπες γραμμές του ψευδοκώδικα εκτελούνται όταν η λειτουργία που εξετάζεται δεν είναι μία από τις προηγούμενες δύο.

Οι αλγόριθμοι που παρουσιάσαμε μέχρι στιγμής για την περίπτωση Γραμμή–Δίαυλος προσπαθούν να κατασκευάσουν τη λύση, προσπαθώντας αρχικά να κατανείμουν

δίκαια το φορτίο και στη συνέχεια να κάνουν διορθωτικές κινήσεις για τη μείωση του χρόνου εκτέλεσης. Ο αλγόριθμος Heavy Operations–Large Messages προσπαθεί να κατασκευάσει τη λύση παίρνοντας συγχρόνως υπόψη τις δύο παραμέτρους.

Αλγόριθμος Heavy Operations – Large Messages

Είσοδοι: Ένα workflow W(O, E) (με τοπολογία γραμμής), όπου $O = (O_1, O_2, ..., O_M)$ ένα σύνολο από λειτουργίες και $E = \{(O_i, O_{i+1}) \mid \forall\ i = 1, ..., M-1\}$ το σύνολο των μεταβάσεων μεταξύ των λειτουργιών και ένα δίκτυο N(S, L) (σε τοπολογία διαύλου), όπου $S = \{S_1, S_2, ..., S_N\}$ ένα σύνολο από εξυπηρετητές και $L = \{(S_i, S_j) \mid \forall\ i, j = 1, ..., N, i \neq j\}$ οι συνδέσεις μεταξύ των εξυπηρετητών. Πρέπει M>N.

Έξοδος: μία αντιστοίχιση (Mapping) M του O στο S.

Περιγραφή: Ο αλγόριθμος αναθέτει τις λειτουργίες σε εξυπηρετητές παρόμοια με τον Fair Load με τη διαφορά ότι δεν χειρίζεται τις λειτουργίες μία-μία ξεχωριστά αλλά τις χειρίζεται σαν ομάδες από λειτουργίες χρησιμοποιώντας τη λίστα Group_Of_Oper_List. Στην ίδια ομάδα μπαίνουν λειτουργίες που ανταλλάσσουν μεγάλα μηνύματα μεταξύ τους. Ένα μήνυμα θεωρείται μεγάλο όταν ο χρόνος για να σταλεί μέσω του διαύλου είναι μεγαλύτερος από τον χρόνο επεξεργασίας των λειτουργιών της ομάδας με τους περισσότερους συνολικά κύκλους στον εξυπηρετητή με τους περισσότερους διαθέσιμους κύκλους τη δεδομένη χρονική στιγμή. Ο αλγόριθμος αρχικά θεωρεί κάθε λειτουργία ξεχωριστά μία ομάδα. Σε κάθε βήμα, είτε αναθέτει τις λειτουργίες της ομάδας με τους περισσότερους υπολογιστικούς κύκλους στον εξυπηρετητή που εκείνη τη στιγμή μπορεί να αναλάβει τους περισσότερους κύκλους, είτε ομαδοποιεί δύο ομάδες σε μία, είτε κάνει μία ανάθεση ώστε να αποτραπεί η αποστολή ενός μεγάλου μηνύματος μέσω του διαύλου.

Begin

1. $Sum\_Cycles = \sum_{i=1}^{M} C(O_i)$ *// Οι συνολικοί κύκλοι που απαιτούν όλες οι λειτουργίες*
2. $Sum\_Capacity = \sum_{i=1}^{N} P(S_i)$ *// Η συνολική υπολογιστική ισχύς των εξυπηρετητών*
3. $Ideal\_Cycles(S_i) = Sum\_Cycles \times \frac{P(S_i)}{Sum\_Capacity}, \forall\ i = 1, ..., N$
4. $Servers\_List = (s_1, s_2, ..., s_N) = (S_1, S_2, ..., S_N)$
5. $Group\_Of\_Oper\_List = (g_1, g_2, ..., g_M) = (O_1, O_2, ..., O_M)$

6. Messages_List = (m_1, m_2, ..., m_{M-1}) = ((O_1, O_2), (O_2, O_3), ..., (O_{M-1}, O_M))
7. Not_Assigned_Operations = M
8. while Not_Assigned_Operations ≠ 0 do
9. Sort Servers_List so that ∀ i=1...N-1 Ideal_Cycles(s_i) ≥ Ideal_Cycles(s_{i+1})
10. Sort Group_Of_Oper_List so that ∀ i=1...number of groups-1 C(g_i) ≥ C(g_{i+1})
11. Sort Messages_List so that
∀ i = 1, ..., size of Messages_List-1 MsgSize(m_i)≥MsgSize(m_{i+1})
12. if Tproc(g_1) at server s_1 > Time of sending m_1 via bus then
13. for each $o_i \in g_1$ do
14. M = M ∪ {$o_i \rightarrow s_1$}
15. Not_Assigned_Operations – –
16. Ideal_Cycles(s_1) – = C(o_i)
17. end for
18. Delete g_1 from Group_Of_Oper_List
19. else
20. if source(m_1) is not assigned and target(m_1) is assigned then
21. M = M ∪ {source(m_1)→ServerOf (target(m_1))}
22. Not_Assigned_Operations – –
23. Ideal_Cycles(ServerOf (target(m_1))) – = C(source(m_1))
24. Delete source(m_1) from its group in Group_Of_Oper_List
25. else if source(m_1) is assigned and target(m_1) is not assigned then
26. M = M ∪ {target(m_1)→ServerOf (source(m_1))}
27. Not_Assigned_Operations – –
28. Ideal_Cycles(ServerOf (source(m_1))) – = C(target(m_1))
29. Delete target(m_1) from its group in Group_Of_Oper_List
30. else // both source(m_1) and target(m_1) are not assigned
31. g_{new} = Merge (group(source(m_1)), group(target(m_1)))
32. Insert g_{new} in Group_Of_Oper_List
33. end if
34. end if
35. for i=1 to size of Messages_List do
36. if source(m_i) and target(m_i) are assigned
37. Delete m_i from Messages_Lists
38. end while
39. return M

End

Στις πρώτες 7 γραμμές του αλγορίθμου Heavy Operations–Large Messages υπολογίζεται το ιδανικό φορτίο για κάθε εξυπηρετητή και δημιουργούνται οι λίστες με τους εξυπηρετητές, τις ομάδες λειτουργιών και τα μηνύματα. Αρχικά, κάθε λειτουργία θεωρείται μία ομάδα. Ο αλγόριθμος, σε κάθε βήμα, ταξινομεί τις παραπάνω λίστες (γραμμές 9-11). Η λίστα των εξυπηρετητών ταξινομείται με βάση τους υπολογιστικούς κύκλους που πρέπει να αναλάβουν οι εξυπηρετητές, η λίστα των ομάδων λειτουργιών με βάση του συνολικούς υπολογιστικούς κύκλους των λειτουργιών της ομάδας και η λίστα των μηνυμάτων με βάση το μέγεθος των μηνυμάτων. Μετά τις ταξινομήσεις, ο αλγόριθμος, είτε αναθέτει τις λειτουργίες της ομάδας που βρίσκεται στην κορυφή της λίστας των λειτουργιών στον εξυπηρετητή που βρίσκεται στην κορυφή της λίστας των εξυπηρετητών (γραμμές 12-18), είτε κάνει μία ενέργεια που αποτρέπει την αποστολή ενός μεγάλου μηνύματος μέσω του διαύλου (γραμμές 19-34). Η πρώτη επιλογή γίνεται όταν ο χρόνος επεξεργασίας των λειτουργιών της ομάδας με τους περισσότερους κύκλους στον εξυπηρετητή που βρίσκεται στην κορυφή της λίστας των εξυπηρετητών είναι μεγαλύτερος από τον χρόνο που χρειάζεται για να σταλεί το μεγαλύτερο μήνυμα (της λίστας μηνυμάτων) μέσω του διαύλου (γραμμή 12). Όταν γίνεται η δεύτερη επιλογή ο αλγόριθμος κάνει τα εξής: αν μία από τις δύο λειτουργίες που ανταλλάσσουν το μήνυμα είναι τοποθετημένη σε κάποιον εξυπηρετητή, τότε και η άλλη λειτουργία τοποθετείται στον ίδιο εξυπηρετητή (γραμμές 20-29). Διαφορετικά, αν καμία από τις λειτουργίες που ανταλλάσσουν το μήνυμα δεν έχουν ανατεθεί σε κάποιον εξυπηρετητή, τότε οι ομάδες που ανήκουν οι δύο λειτουργίες, γίνονται μία ομάδα στην λίστα των ομάδων λειτουργιών (γραμμές 30-33). Τέλος, στις γραμμές 35-37, ελέγχεται αν κάποιο από τα μηνύματα που βρίσκονται στη λίστα των μηνυμάτων πρέπει να διαγραφεί από τη λίστα. Η διαγραφή ενός μηνύματος γίνεται αν οι δύο λειτουργίες που ανταλλάσσουν το μήνυμα έχουν ανατεθεί σε κάποιον εξυπηρετητή.

4.4. Αλγόριθμοι Τυχαίος Γράφος=Δίαυλος

Οι αλγόριθμοι που χρησιμοποιούνται σε αυτήν την περίπτωση είναι αυτοί που χρησιμοποιήθηκαν στην περίπτωση Γραμμή=Δίαυλος με μικρές τροποποιήσεις ώστε να λαμβάνουν υπόψη την τοπολογία του workflow. Μόνο ο αλγόριθμος Fair Load

χρησιμοποιείται χωρίς καμία τροποποίηση. Οι υπόλοιποι πρέπει να λαμβάνουν υπόψην τους ότι ένα μήνυμα ή μία λειτουργία μπορεί να εκτελείται λιγότερες φορές από αυτές που εκτελείται το workflow, λόγω των xor διακλαδώσεων. Έτσι οι κύκλοι κάθε λειτουργίας και το μέγεθος κάθε μηνύματος πρέπει να πολλαπλασιάζονται με ένα βάρος, το οποίο ισοδυναμεί με τη συχνότητα εκτέλεσης της λειτουργίας ή του μηνύματος αντίστοιχα. Επίσης, πρέπει να λαμβάνουν υπόψη ότι μία λειτουργία μπορεί να δέχεται περισσότερα του ενός μηνύματα και να στέλνει περισσότερα από ένα μηνύματα (σε αντίθεση με την περίπτωση Γραμμή–Δίαυλος).

ΚΕΦΑΛΑΙΟ 5. ΠΕΙΡΑΜΑΤΙΚΕΣ ΜΕΤΡΗΣΕΙΣ

5.1 Τιμές των Παραμέτρων στα Πειράματα
5.2 Πειράματα για τον Αλγόριθμο Γραμμή – Γραμμή
5.3 Πειράματα για τους Αλγορίθμους Γραμμή = Δίαυλος
5.4 Πειράματα για τους Αλγορίθμους Τυχαίος Γράφος = Δίαυλος

Στο συγκεκριμένο κεφάλαιο περιγράφονται τα πειράματα που έγιναν για τις τρεις διαφορετικές διαστάσεις του προβλήματος που περιγράφτηκαν στο 4° κεφάλαιο και για τις οποίες δόθηκαν οι αντίστοιχοι αλγόριθμοι. Στην αρχή του κεφαλαίου υπάρχει μία ενότητα που εξηγεί τις τιμές που ανατέθηκαν στις παραμέτρους του προβλήματος.

5.1. Τιμές των Παραμέτρων στα Πειράματα

Για μια σωστή εκτίμηση των πειραμάτων όπου μελετάται η επίδοση των αλγορίθμων, απαιτείται οι τιμές των παραμέτρων να είναι ρεαλιστικές και να αντιπροσωπεύουν τις τιμές που έχουν οι συγκεκριμένοι παράμετροι σε πραγματικά δίκτυα και υπηρεσίες διαδικτύου. Οι τιμές που επιλέχθηκαν για την υπολογιστική ισχύ των εξυπηρετητών είναι 1, 2 και 3 GHz, και για τις ταχύτητες των γραμμών 10, 100 και 1000 Mbps. Ο χρόνος διάδοσης μεταξύ δύο οποιονδήποτε εξυπηρετητών ορίζεται ίσος με 0.1 ms. Για τα μεγέθη των μηνυμάτων που ανταλλάσσουν οι λειτουργίες και για τους κύκλους που οι χρειάζονται οι λειτουργίες για να εκτελεστούν συμβουλευόμαστε τα

άρθρα [HGCL+05] και [NgCG04]. Στα συγκεκριμένα άρθρα γίνονται πειραματικές μετρήσεις σε διάφορες υλοποιήσεις του SOAP πρωτοκόλλου. Στο δεύτερο άρθρο περιγράφονται τρία είδη SOAP μηνυμάτων (*simple*, *medium* και *complex*) που θα μπορούσαν να είναι μηνύματα πραγματικών υπηρεσιών διαδικτύου καθώς και τα μεγέθη τους στις διαφορετικές SOAP υλοποιήσεις. Εμείς χρησιμοποιούμε τα μεγέθη των μηνυμάτων που προκύπτουν από την καλύτερη υλοποίηση (product A) και το καλύτερο στυλ κωδικοποίησης (Document/Literal). Πιο συγκεκριμένα, για το simple μήνυμα χρησιμοποιούμε το μέγεθος των 873 bytes (0.00666 Mbits), για το medium μήνυμα το μέγεθος των 7581 bytes (0.057838 Mbits), και για το complex το μέγεθος των 21392 bytes (0.163208 Mbits). Εκτός από τα μεγέθη των τριών μηνυμάτων, στο συγκεκριμένο άρθρο, υπολογίζεται και ο χρόνος που απαιτείται για την κλήση της αντίστοιχης υπηρεσίας διαδικτύου. Ο χρόνος αυτός περιλαμβάνει το χρόνο που σπαταλάται στο δίκτυο, το χρόνο που απαιτείται για το parsing (serialization, deserialization) των μηνυμάτων και το χρόνο που χρειάζεται ο εξυπηρετητής για την εκτέλεση της υπηρεσίας διαδικτύου. Ο χρόνος αυτός για τα τρία μηνύματα είναι 4, 10 και 20 ms. Υπολογίζεται ότι ο χρόνος που χρειάζεται ένα μήνυμα για το parsing στη μεριά του εξυπηρετητή είναι 37% περίπου αυτού του χρόνου. Έτσι προκύπτει ότι ο χρόνος για το parsing στη μεριά του εξυπηρετητή είναι 1.5 ms για το simple μήνυμα, 3.7 ms για το medium μήνυμα και 7.4 ms για complex μήνυμα. Μετατρέποντας το χρόνο σε κύκλους (χρησιμοποιήθηκε επεξεργαστής 1.6 GHz με 3669 Mbytes μνήμης), προκύπτει ότι οι κύκλοι που απαιτούνται για το parsing των τριών μηνυμάτων είναι περίπου 2.5, 6.3 και 12.7 εκατομμύρια κύκλοι για το simple, medium και complex μήνυμα αντίστοιχα. Λαμβάνοντας υπόψη τις παραπάνω τιμές ορίζουμε ελαφριές λειτουργίες (περίπου 5 εκατομμύρια κύκλους), μεσαίες λειτουργίες (περίπου 50 εκατομμύρια κύκλους) και βαριές λειτουργίες (περίπου 500 εκατομμύρια κύκλους).

Τα πειράματα χωρίζονται σε τρεις κατηγορίες ανάλογα με το ποιες παράμετροι του προβλήματος μεταβάλλονται. Στην κατηγορία A εξετάζεται η απόδοση των αλγορίθμων σε σχέση με την ταχύτητα των συνδέσεων και το μέγεθος των μηνυμάτων που ανταλλάσσονται, στην κατηγορία B εξετάζεται η απόδοση σε σχέση με την υπολογιστική ισχύ των εξυπηρετητών και των όγκο εργασίας των λειτουργιών

και στην κατηγορία Γ εξετάζεται η απόδοση μεταβάλλοντας όλες τις παραμέτρους τους προβλήματος.

Για τα πειράματα της κατηγορίας Α ισχύουν οι εξής παράμετροι:

- $C(O_i) = 20 \times 10^6$ κύκλοι, $\forall\ O_i \in \mathbf{O}$
- $P(S_i) = 2$ GHz, $\forall\ S_i \in \mathbf{S}$
- $Line\_Speed(S_i, S_{i+1}) = \left\{ \begin{array}{l} \text{10 Mbps με πιθανότητα 25\%} \\ \text{100 Mbps με πιθανότητα 50\%} \\ \text{1000 Mbps με πιθανότητα 25\%} \end{array} \right\}$, $\forall\ i \in [1, N-1]$
- $MsgSize(O_i, O_{i+1})$ = (από τον πίνακα 5.1), $\forall\ i \in [1, M-1]$. Για τα τρία διαφορετικά πειράματα της κατηγορίας Α (A_1, A_2, A_3) δοκιμάστηκαν τα μεγέθη μηνύματος που φαίνονται στον Πίνακα 5.1. Τα πειράματα A_1, A_2, A3 αντιστοιχούν σε μικρά, μεσαία και μεγάλα μηνύματα.

Πίνακας 5.1 Τα Μεγέθη Μηνυμάτων των Πειραμάτων της Κατηγορίας Α.

Πείραμα / Ποσοστό	A_1	A_2	A_3
25% μηνυμάτων	0.002 Mbits	0.02 Mbits	0.1 Mbits
50% μηνυμάτων	0.006 Mbits	0.06 Mbits	0.2 Mbits
25% μηνυμάτων	0.01 Mbits	0.1 Mbits	0.3 Mbits

Για τα πειράματα της κατηγορίας Β ισχύουν οι εξής παράμετροι:

- $MsgSize(O_i, O_{i+1}) = 0.06$ Mbits, $\forall\ i \in [1, M-1]$
- $Line\_Speed(S_i, S_{i+1}) = 100$ Mbps, $\forall\ i \in [1, N-1]$
- $P(S_i) = \left\{ \begin{array}{l} \text{1 GHz με πιθανότητα 25\%} \\ \text{2 GHz με πιθανότητα 50\%} \\ \text{3 GHz με πιθανότητα 25\%} \end{array} \right\}$, $\forall\ S_i \in \mathbf{S}$
- $C(O_i)$ = (από τον πίνακα 5.2), $\forall\ O_i \in \mathbf{O}$. Για τα τρία διαφορετικά πειράματα της κατηγορίας Β (B_1, B_2, B_3) δοκιμάστηκαν οι όγκοι εργασίας που φαίνονται στον Πίνακα 5.2. Τα τρία διαφορετικά πειράματα της κατηγορίας Β αντιστοιχούν σε ελαφριές, μεσαίες και βαριές λειτουργίες.

Πίνακας 5.2 Τα Μεγέθη Μηνυμάτων των Πειραμάτων της Κατηγορίας Β.

Πείραμα / Ποσοστό	B_1	B_2	B_3
25% λειτουργιών	3 εκ. κύκλοι	30 εκ. κύκλοι	300 εκ. κύκλοι
50% λειτουργιών	5 εκ. κύκλοι	50 εκ. κύκλοι	500 εκ. κύκλοι
25% λειτουργιών	7 εκ. κύκλοι	70 εκ. κύκλοι	700 εκ. κύκλοι

Για τα πειράματα της κατηγορίας Γ ισχύουν οι εξής παράμετροι:

- $MsgSize(O_i, O_{i+1}) = \left\{ \begin{array}{l} 0.006660 \text{ Mbits με πιθανότητα } 25\% \\ 0.057838 \text{ Mbits με πιθανότητα } 50\% \\ 0.163208 \text{ Mbits με πιθανότητα } 25\% \end{array} \right\}$, $\forall\, i \in [1, M-1]$

- $Line\_Speed(S_i, S_{i+1}) = \left\{ \begin{array}{l} 10 \text{ Mbps με πιθανότητα } 25\% \\ 100 \text{ Mbps με πιθανότητα } 50\% \\ 1000 \text{ Mbps με πιθανότητα } 25\% \end{array} \right\}$, $\forall\, i \in [1, N-1]$

- $C(O_i) = \left\{ \begin{array}{l} 10 \text{ εκ. κύκλοι με πιθανότητα } 25\% \\ 20 \text{ εκ. κύκλοι με πιθανότητα } 50\% \\ 30 \text{ εκ. κύκλοι με πιθανότητα } 25\% \end{array} \right\}$, $\forall\, O_i \in \mathbf{O}$

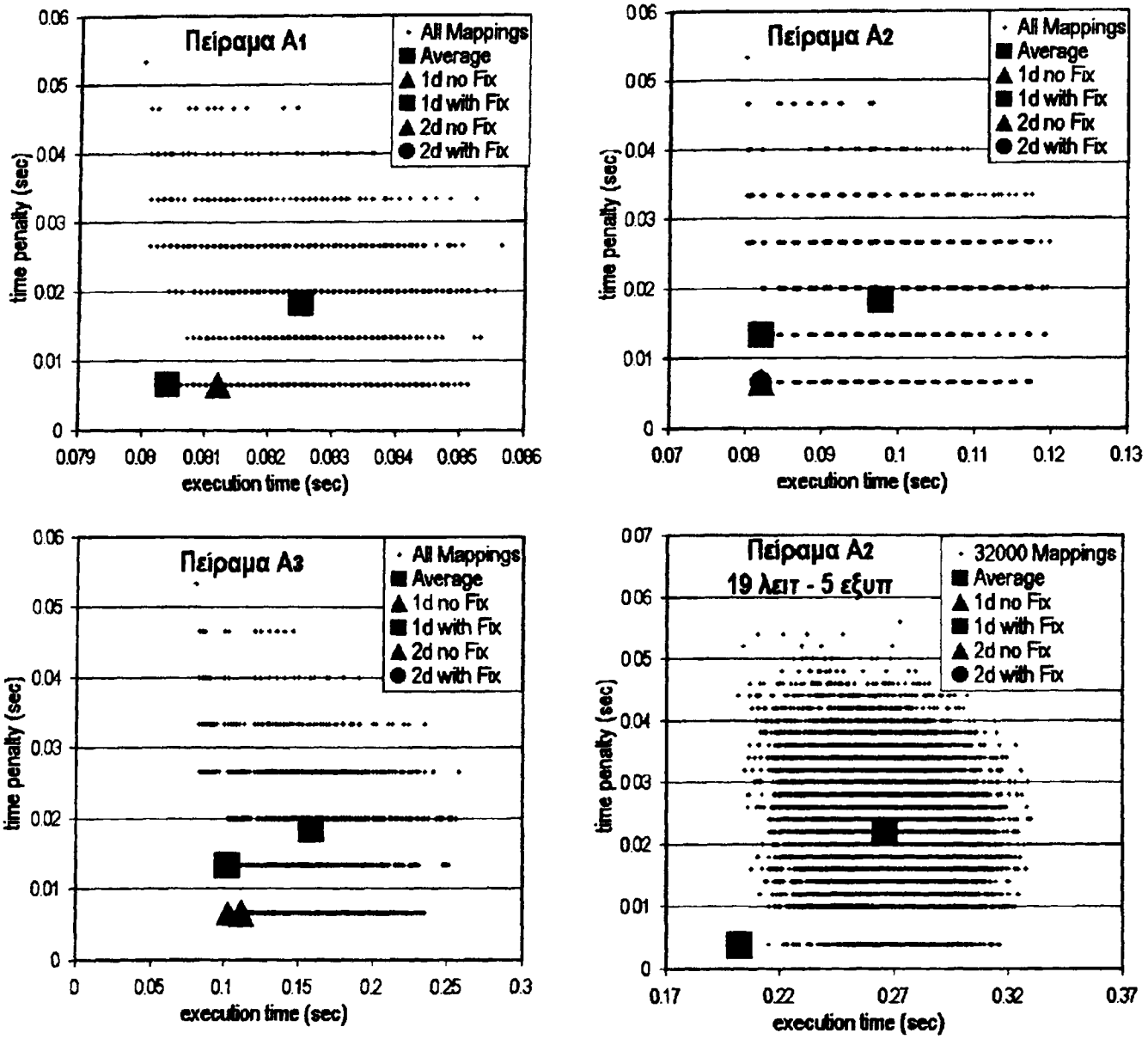
- $P(S_i) = \left\{ \begin{array}{l} 1 \text{ GHz με πιθανότητα } 25\% \\ 2 \text{ GHz με πιθανότητα } 50\% \\ 3 \text{ GHz με πιθανότητα } 25\% \end{array} \right\}$, $\forall\, S_i \in \mathbf{S}$

Στα πειράματα απεικονίζουμε εκτός των λύσεων των αλγορίθμων μας και τη μέση τιμή (Average) όλων των δυνατών λύσεων (ή του δείγματος σε περίπτωση δειγματοληψίας). Αυτό το κάνουμε για να ελέγξουμε αν οι αλγόριθμοί μας δίνουν καλύτερη λύση από μία λύση που θα παραγόταν τυχαία. Οι λύσεις που βρίσκονται πιο κάτω και πιο αριστερά από την μέση τιμή όλων των λύσεων είναι καλύτερες από

αυτή. Έτσι αν οι αλγόριθμοί μας δίνουν τέτοιες λύσεις κάνουν μία βελτίωση σε σχέση με την τυχαία ανάθεση λειτουργιών σε εξυπηρετητές.

5.2. Πειράματα για τον Αλγόριθμο Γραμμή-Γραμμή

Τα πειράματα που έγιναν για τον συγκεκριμένο αλγόριθμο αφορούν την απόδοση των τεσσάρων παραλλαγών του αλγορίθμου σε σύγκριση με τον εξαντλητικό αλγόριθμο. Οι τέσσερις παραλλαγές είναι οι εξής: μιας κατεύθυνσης Γραμμή – Γραμμή χωρίς, ή με εφαρμογή της επιδιόρθωσης κακών γεφυρών (1d no Fix και 1d with Fix αντίστοιχα) και διπλής κατεύθυνσης Γραμμή – Γραμμή χωρίς, ή με εφαρμογή της επιδιόρθωσης κακών γεφυρών (2d no Fix και 2d with Fix αντίστοιχα) Οι άξονες στα διαγράμματα των πειραμάτων είναι ο χρόνος εκτέλεσης (οριζόντιος άξονας) και το Time_Penalty (κάθετος άξονας). Στον οριζόντιο άξονα, όσο πιο κοντά είναι μία λύση στην αρχή των αξόνων τόσο πιο μικρό χρόνο εκτέλεσης έχει το workflow. Στον κάθετο άξονα, όσο πιο κοντά είναι μία λύση στην αρχή των αξόνων τόσο πιο δίκαιη είναι η κατανομή του φορτίου.

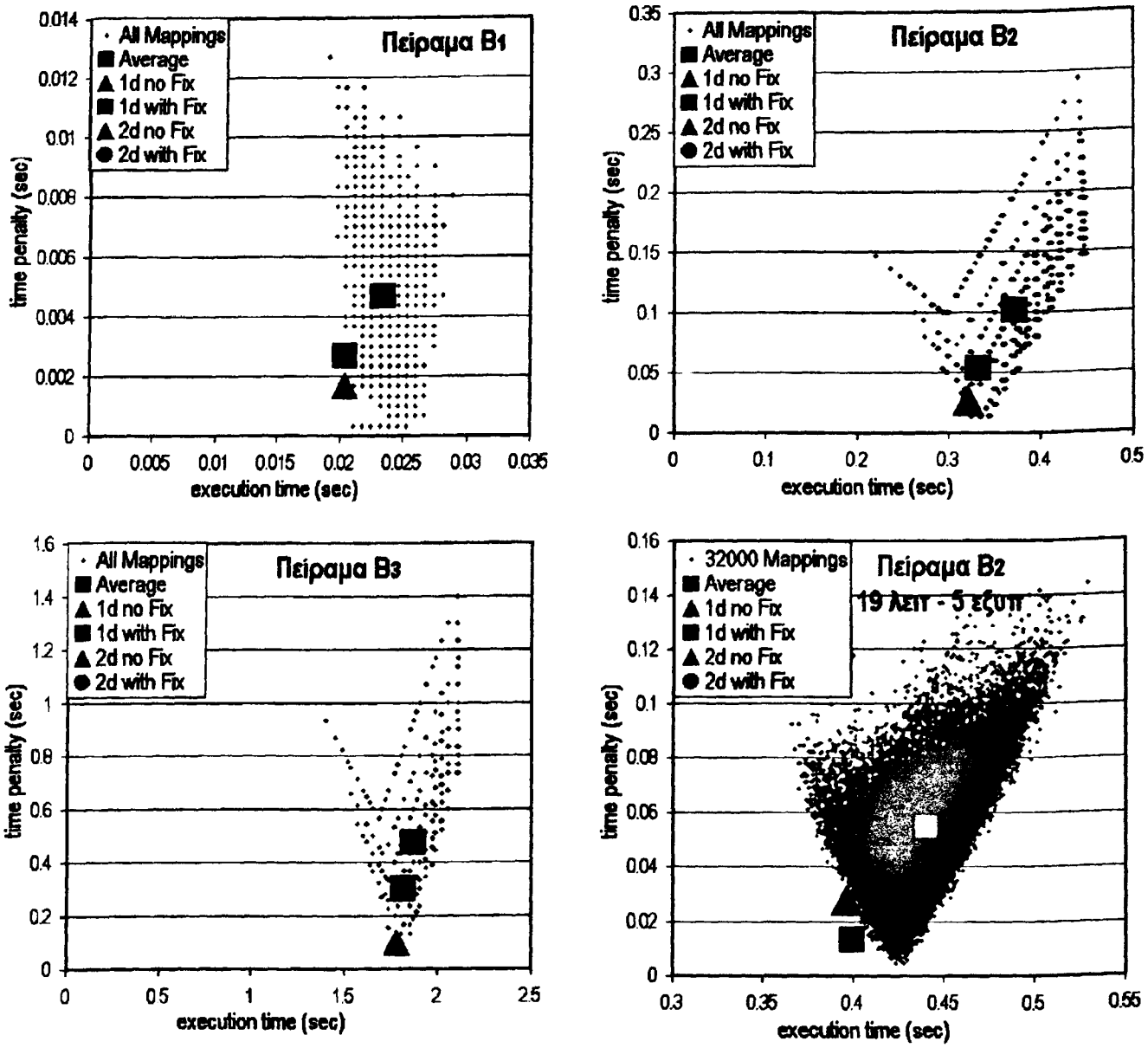


Σχήμα 5.1 Πειράματα Κατηγορίας Α για τον Αλγορίθμου Γραμμή–Γραμμή.

Στο Σχήμα 5.1 φαίνονται τρία πειράματα (A_1, A_2, A_3) της κατηγορίας Α για 8 λειτουργίες και 3 εξυπηρετητές και ένα πείραμα A_2 με 19 λειτουργίες και 5 εξυπηρετητές. Σε κάθε πείραμα απεικονίζονται όλες οι δυνατές λύσεις (εκτός του τελευταίου που έγινε δειγματοληψία) που προκύπτουν από τον εξαντλητικό αλγόριθμο (All Mappings), η μέση τιμή όλων των δυνατών λύσεων (Average), όπως επίσης και οι λύσεις που δίνουν οι τέσσερις παραλλαγές του αλγορίθμου Γραμμή–Γραμμή. Από τα πειράματα διαπιστώνεται ότι όλοι οι αλγόριθμοι δίνουν καλή λύση, ενώ φαίνεται ότι η επιδιόρθωση των κρίσιμων γεφυρών (που συμβολίζεται ως Fix στα διαγράμματα) μπορεί να χαλάσει λίγο την κατανομή του φορτίου στους εξυπηρετητές

και επιπλέον δεν βελτιώνει πάντα το χρόνο εκτέλεσης. Έτσι, κοιτώντας τα διαγράμματα, καλές λύσεις στο πρόβλημά μας, είναι αυτές που είναι κοντά στην αρχή των αξόνων γιατί θέλουμε λύσεις με μικρό χρόνο εκτέλεσης αλλά και με δίκαιη κατανομή του φορτίου. Σε αυτό το σημείο, πρέπει να σημειωθεί ότι οι κλίμακες των δύο αξόνων διαφέρουν πολύ από πείραμα σε πείραμα. Αυτό σημαίνει, για παράδειγμα στον οριζόντιο άξονα, ότι σε ένα διάγραμμα μπορεί να είναι πιο σημαντική η διαφορά δύο διαφορετικών λύσεων, όσον αφορά των χρόνο εκτέλεσης, από ότι σε ένα άλλο διάγραμμα (π.χ., στα πειράματα A_1 και A_3 ο οριζόντιος άξονας είναι πιο σημαντικός στο πείραμα A_3).

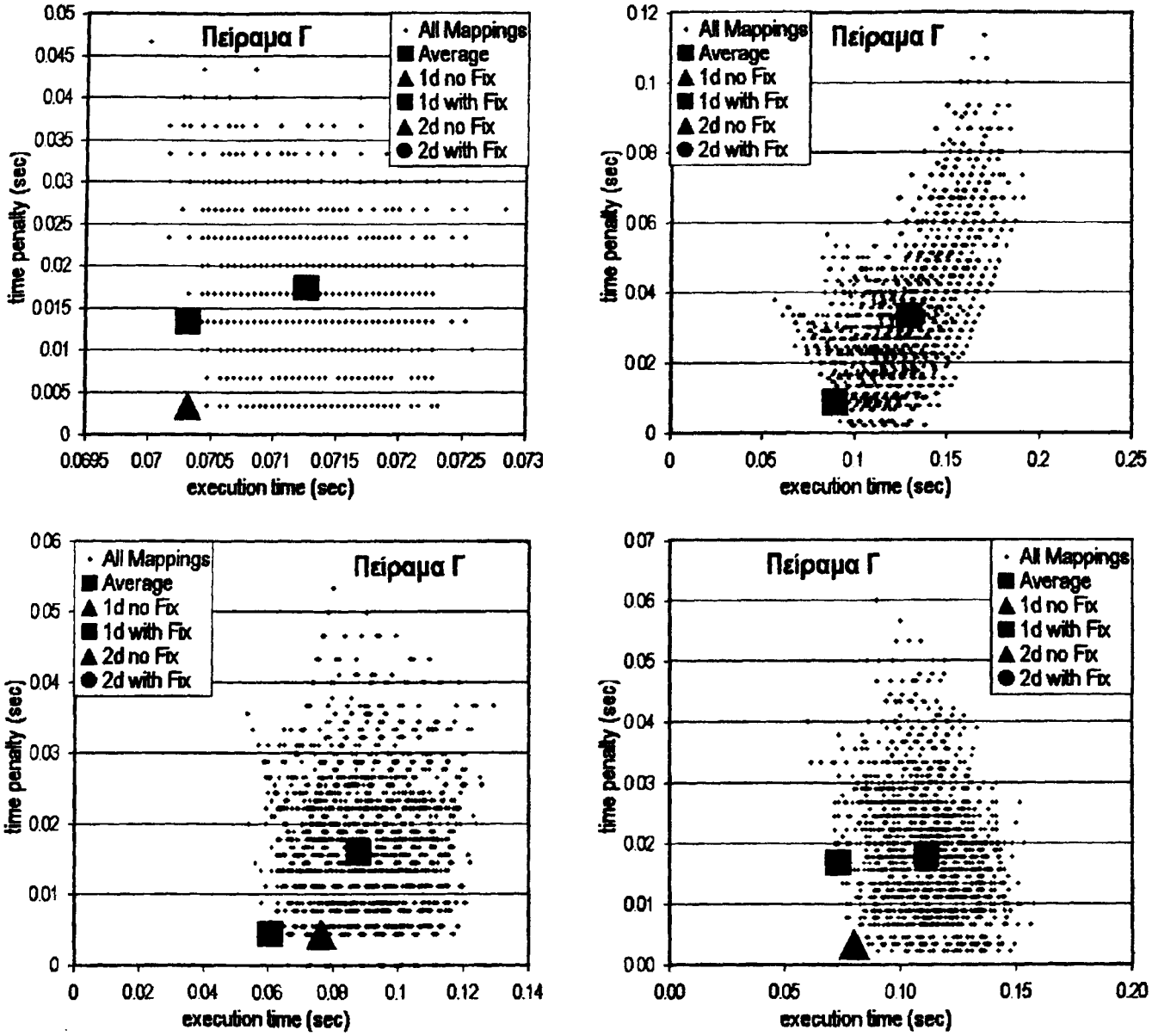
Στο Σχήμα 5.2 φαίνονται τρία πειράματα (B_1, B_2, B_3) της κατηγορίας B για 8 λειτουργίες και 3 εξυπηρετητές και ένα πείραμα B_2 για 19 λειτουργίες και 5 εξυπηρετητές. Σε κάθε πείραμα απεικονίζονται όλες οι δυνατές λύσεις (εκτός του τελευταίου που έγινε δειγματοληψία) που προκύπτουν από τον εξαντλητικό αλγόριθμο (All Mappings), η μέση τιμή όλων των δυνατών λύσεων (Average), όπως επίσης και οι λύσεις που δίνουν οι τέσσερις παραλλαγές του αλγορίθμου Γραμμή-Γραμμή. Παρατηρώντας τα πειράματα διαπιστώνουμε ότι η επιδιόρθωση των κρίσιμων γεφυρών χειροτερεύει την κατανομή του φορτίου, ενώ μπορεί να χειροτερεύει και το χρόνο εκτέλεσης. Συμπεραίνουμε ότι η επιδιόρθωση των κρίσιμων γεφυρών δεν πρέπει να γίνεται στην κατηγορία B γιατί όλα τα μηνύματα είναι ίσα και σε κάθε βήμα οι γέφυρες θεωρούνται εσφαλμένα κρίσιμες.



Σχήμα 5.2 Πειράματα Κατηγορίας Β για τον Αλγόριθμο Γραμμή–Γραμμή.

Στο Σχήμα 5.3 φαίνονται τέσσερα πειράματα της κατηγορίας Γ για 8 λειτουργίες και 3 εξυπηρετητές. Σε ένα από αυτά όλες οι λύσεις των παραλλαγών συμπίπτουν. Στα δύο η επιδιόρθωση των κρίσιμων γεφυρών χειροτερεύει την κατανομή φορτίου χωρίς να καλυτερεύει πολύ το χρόνο εκτέλεσης ενώ σε ένα από αυτά η επιδιόρθωση των κρίσιμων γεφυρών καλυτερεύει το χρόνο εκτέλεσης χωρίς να χειροτερεύει καθόλου

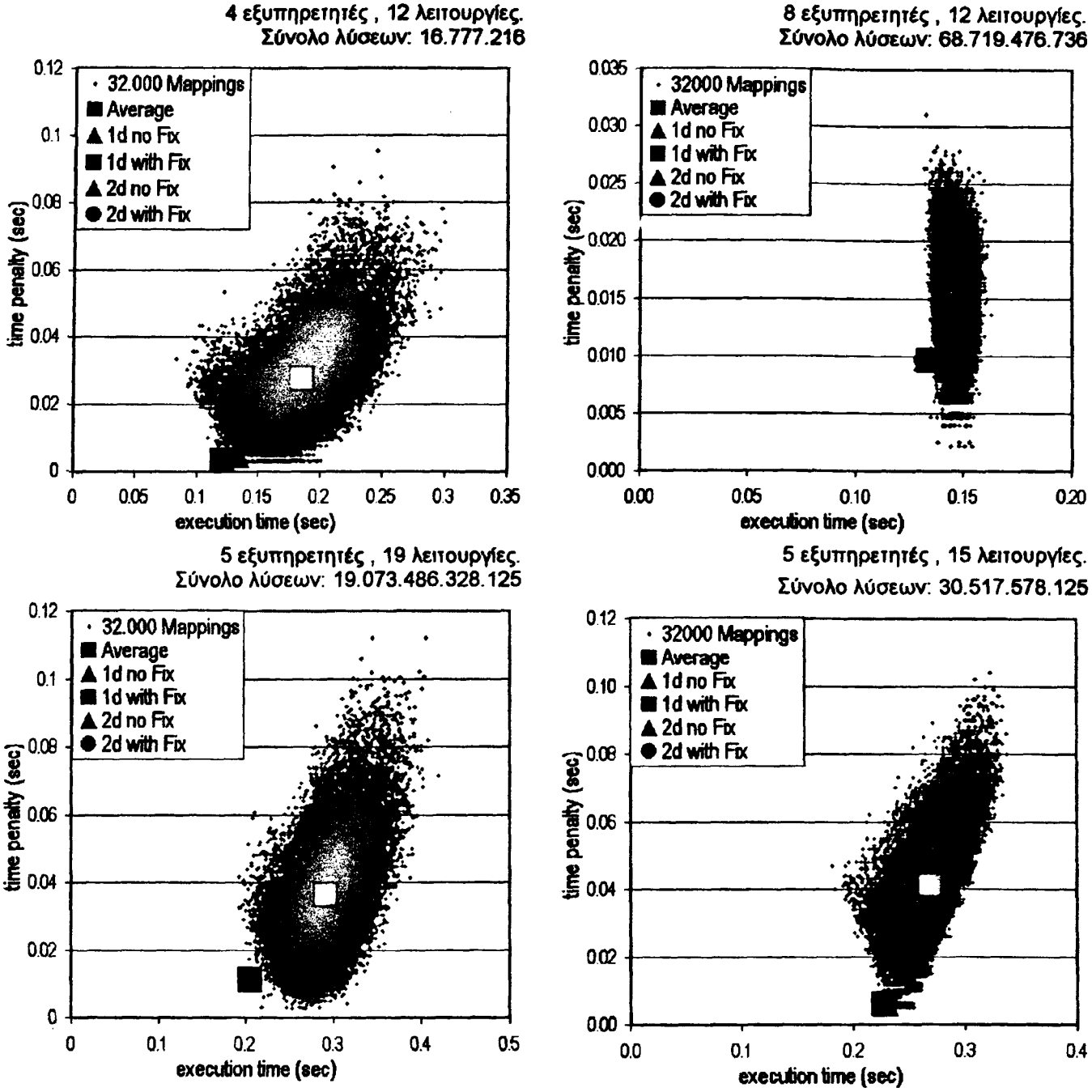
την κατανομή του φορτίου. Ως συμπέρασμα μπορούμε να πούμε ότι χρησιμοποιώντας και τις τέσσερις παραλλαγές του αλγορίθμου Γραμμή–Γραμμή μπορούμε να λάβουμε μία λύση πολύ κοντά σε αυτή που θεωρούμε βέλτιστη.



Σχήμα 5.3 Πειράματα Κατηγορίας Γ για τον Αλγόριθμο Γραμμή–Γραμμή.

Στο Σχήμα 5.4 απεικονίζονται τέσσερα πειράματα κατηγορίας Γ για μεγαλύτερα όμως συστήματα, με περισσότερους εξυπηρετητές και περισσότερες λειτουργίες. Στα πειράματα αυτά επειδή το πλήθος των συνολικών λύσεων είναι πολύ μεγάλο, δεν τις παράγουμε όλες αλλά γίνεται τυχαία δειγματοληψία 32.000 λύσεων. Όπως και σε μικρότερα συστήματα έτσι κι εδώ φαίνεται ότι οι παραλλαγές των αλγορίθμων δεν

έχουν σημαντικές διαφορές μεταξύ τους και η λύση που δίνουν είναι ικανοποιητική, όχι μόνο ως προς την κατανομή του φορτίου αλλά και ως προς το χρόνο εκτέλεσης.

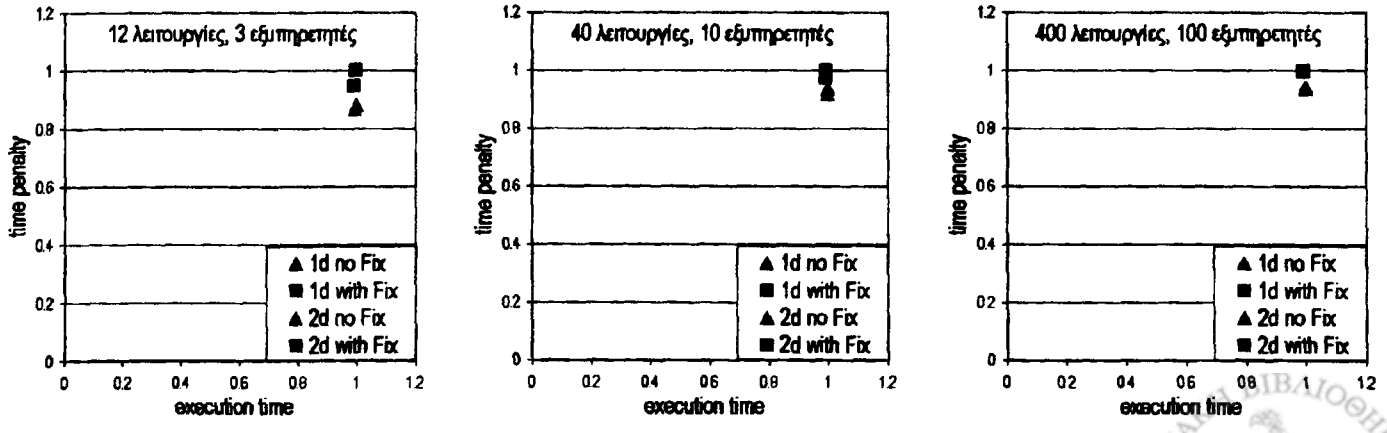


Σχήμα 5.4 Επίδοση του Αλγορίθμου Γραμμή–Γραμμή και των Παραλλαγών του σε Μεγαλύτερα Συστήματα.

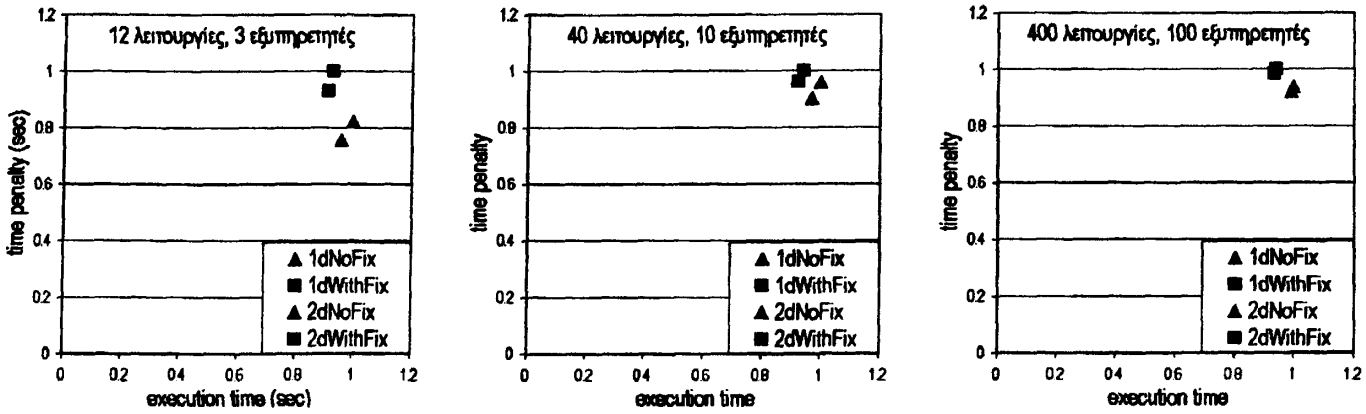
Στη συνέχεια ακολουθεί σύγκριση των παραλλαγών του αλγορίθμου Γραμμή–Γραμμή σε 100 συνολικά πειράματα. Για τη σύγκριση έγιναν τρεις διαφορετικές αναθέσεις όσον αφορά τον αριθμό των εξυπηρετητών και των λειτουργιών. Και στις τρεις αναθέσεις ο παράγοντας ομαδοποίησης ισούται με τέσσερα (K = 4), με τον αριθμό τον

εξυπηρετητών να είναι για τις τρεις αναθέσεις 3, 10 και 100. Τα αποτελέσματα της σύγκρισης φαίνονται στα Σχήματα 5.5 και 5.6 όπου οι τιμές στους άξονες είναι κανονικοποιημένες. Η διαφορά στα πειράματα των δύο αυτών σχημάτων είναι ότι στο Σχήμα 5.5 χρησιμοποιούνται μεγάλες ταχύτητες συνδέσεων (25% των συνδέσεων 10 Mbps, 50% των συνδέσεων 100 Mbps και 25% των συνδέσεων 1000 Mbps) ενώ στο Σχήμα 5.6 μικρότερες (25% των συνδέσεων 1 Mbps, 50% των συνδέσεων 10 Mbps και 25% των συνδέσεων 100 Mbps).

Το πρώτο συμπέρασμα που προκύπτει είναι ότι ο αλγόριθμος 2d είναι καλύτερος από τον 1d. Το δεύτερο συμπέρασμα είναι ότι το "Fix" των κακών γεφυρών χειροτερεύει περισσότερο την κατανομή του φορτίου (Time_Penalty) από ότι καλυτερεύει το χρόνο εκτέλεσης ($T_{execute}$) κυρίως όταν οι ταχύτητες συνδέσεων μεταξύ των εξυπηρετητών είναι μεγάλες. Ένα τρίτο συμπέρασμα είναι ότι οι μεταβολές στις τιμές του $T_{execute}$ και του Time_Penalty, όταν γίνεται επιδιόρθωση των κρίσιμων γεφυρών, είναι μεγαλύτερες όταν το μέγεθος του συστήματος είναι μικρό (μικρός αριθμός εξυπηρετητών και λειτουργιών). Ένα γενικό συμπέρασμα είναι ότι η επιδιόρθωση των κρίσιμων γεφυρών πρέπει να γίνεται όταν οι ταχύτητες των συνδέσεων μεταξύ των εξυπηρετητών είναι μικρές, ή εξίσου, όταν τα μεγέθη των μηνυμάτων που ανταλλάσσονται μεταξύ των εξυπηρετητών είναι πολύ μεγάλα.



Σχήμα 5.5 Σύγκριση των Παραλλαγών του Αλγορίθμου Γραμμή–Γραμμή για Συνδέσεις με Μεγάλες Ταχύτητες.

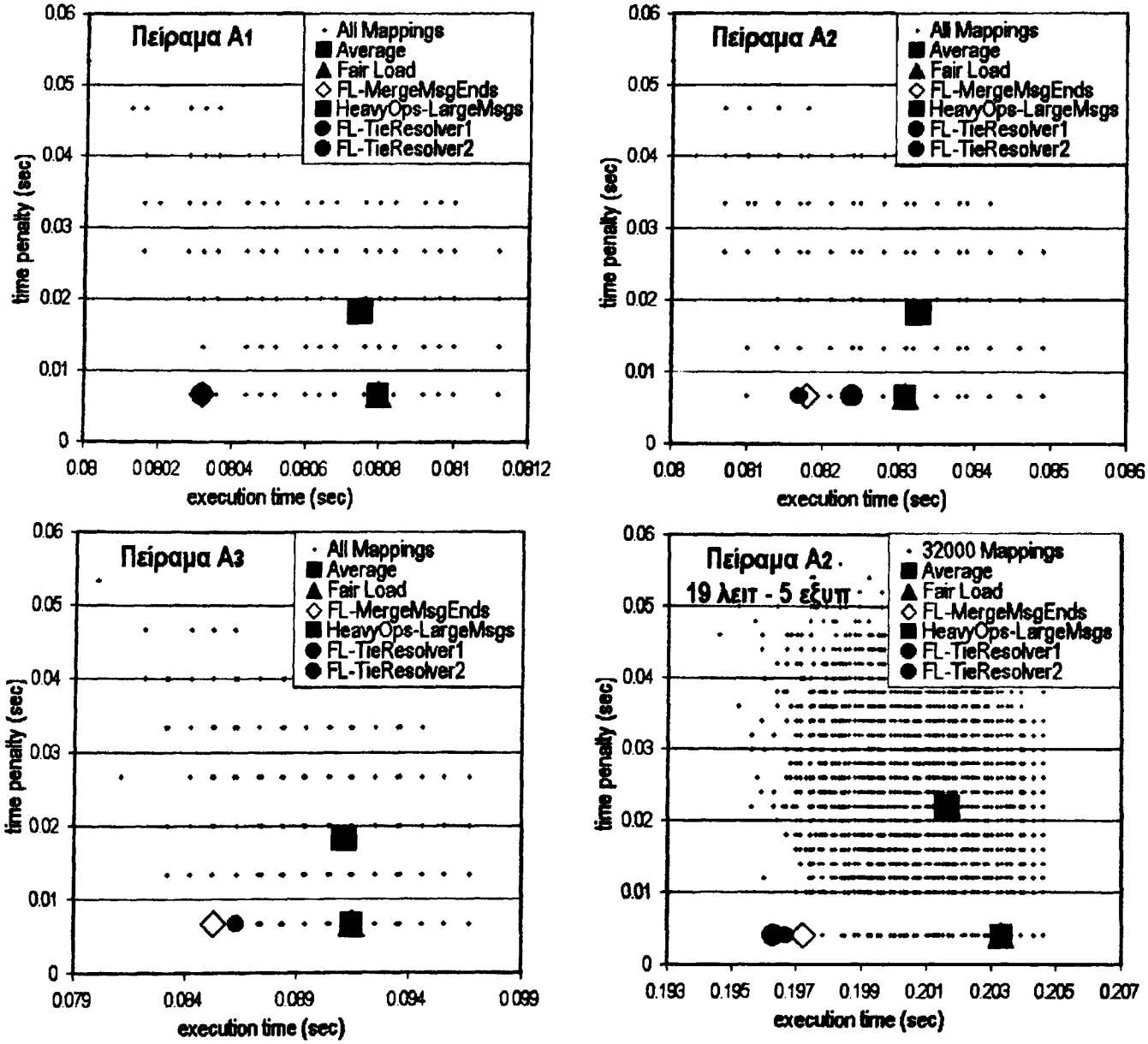


Σχήμα 5.6 Σύγκριση των Παραλλαγών του Αλγορίθμου Γραμμή–Γραμμή για Συνδέσεις με Μικρές Ταχύτητες.

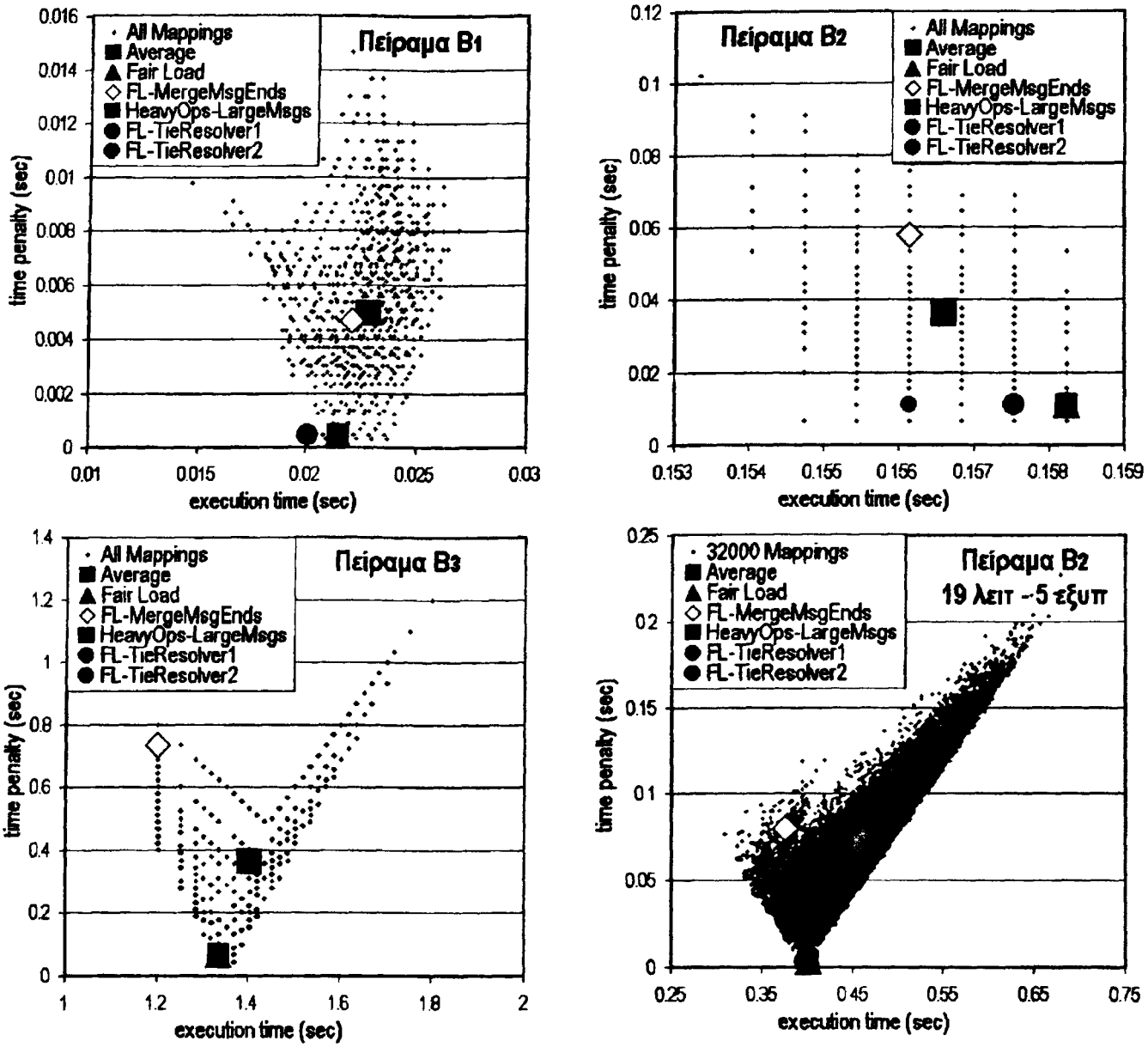
5.3. Πειράματα για τους Αλγορίθμους Γραμμή–Δίαυλος

Όπως και στον αλγόριθμο Γραμμή–Γραμμή, έτσι και στους αλγορίθμους Γραμμή–Δίαυλος γίνονται πειράματα των τριών κατηγοριών και συγκρίνονται οι λύσεις που δίνουν οι διάφοροι αλγόριθμοι για την τοπολογία Γραμμή–Δίαυλος με τις λύσεις του εξαντλητικού αλγορίθμου. Στο Σχήμα 5.7 απεικονίζονται πειράματα της κατηγορίας Α, στο Σχήμα 5.8 πειράματα της κατηγορίας Β και στο Σχήμα 5.9 πειράματα της κατηγορίας Γ. Σε αυτά τα πειράματα χρησιμοποιούνται 3 εξυπηρετητές και 8 λειτουργίες εκτός από δύο πειράματα στα Σχήματα 5.7 και 5.8 όπου χρησιμοποιούνται 5 εξυπηρετητές και 19 λειτουργίες.

Στα πειράματα της κατηγορίας Α βλέπουμε ότι όλοι οι αλγόριθμοι δίνουν λύσεις με την ίδια κατανομή φορτίου. Ξεχωρίζουν οι αλγόριθμοι Fair Load–Merge Messages' Ends και Fair Load–Tie Resolver$_2$ που βελτιώνουν πολύ το χρόνο εκτέλεσης του Fair Load.



Σχήμα 5.7 Πειράματα Κατηγορίας Α για τους Αλγορίθμους Γραμμή–Δίαυλος.

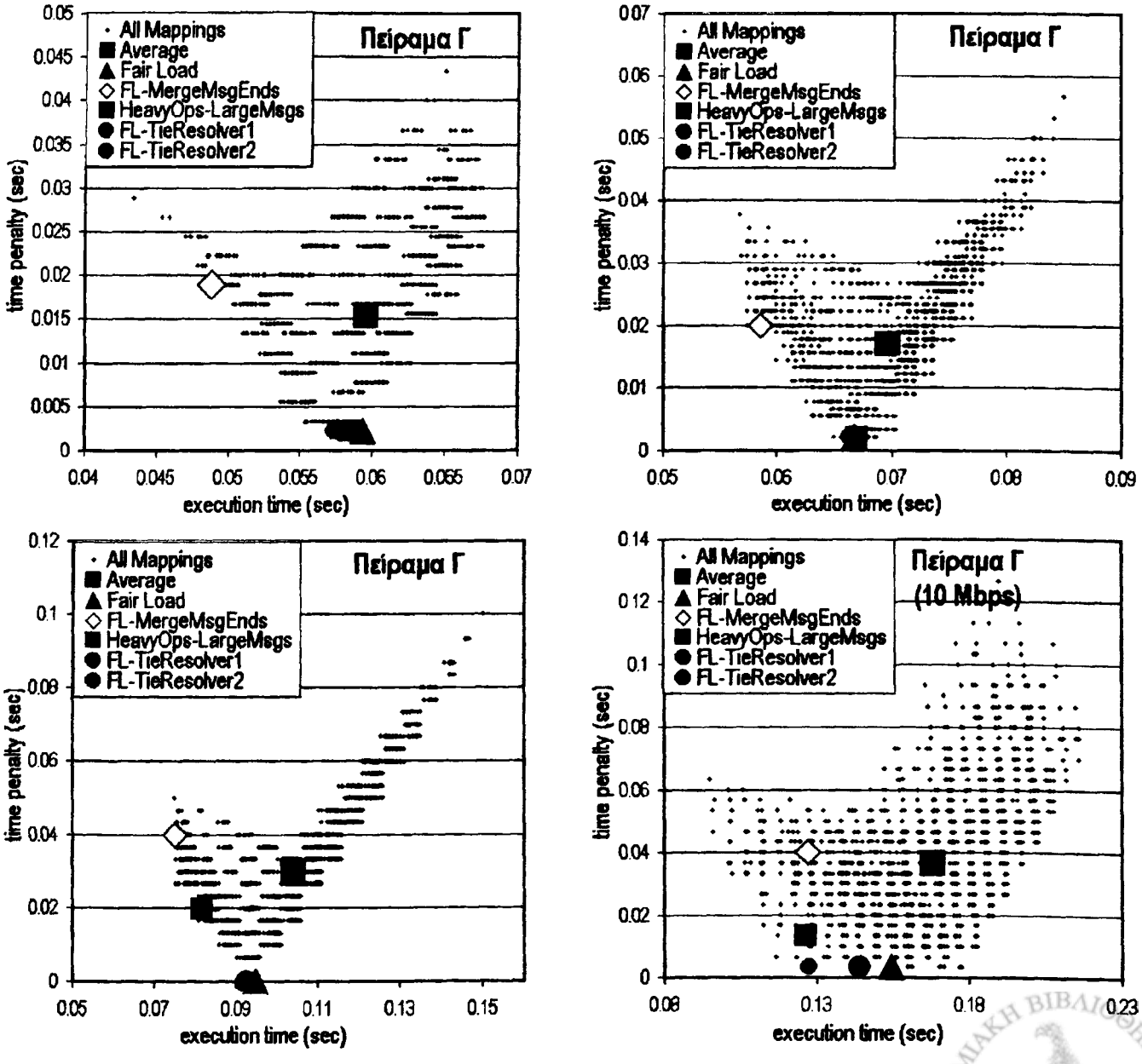


Σχήμα 5.8 Πειράματα Κατηγορίας Β για τους Αλγορίθμους Γραμμή–Δίαυλος.

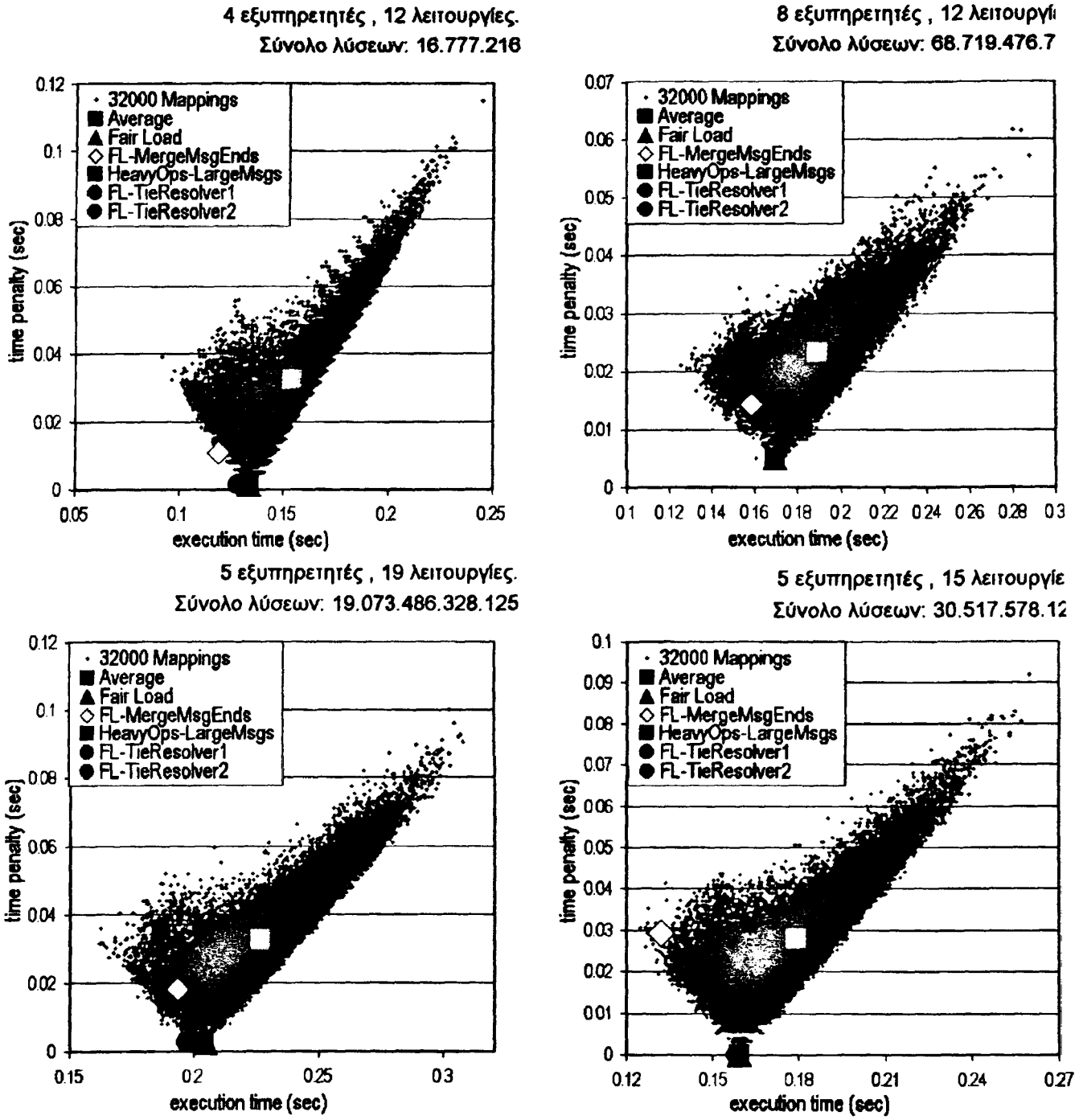
Στα πειράματα της κατηγορίας Β παρατηρούμε επίσης πολύ καλά αποτελέσματα των αλγορίθμων με τους Tie Resolver αλγορίθμους να βελτιώνουν το χρόνο εκτέλεσης του Fair Load όταν υπάρχει περιθώριο βελτίωσης. Ο αλγόριθμος Fair Load–Merge Messages' Ends δεν πρέπει να λαμβάνεται υπόψην για αυτήν την κατηγορία πειραμάτων γιατί όλα τα μηνύματα είναι ίσου μεγέθους και οι βελτιώσεις που κάνει μετακινώντας μηνύματα δεν έχουν κάποια βάση.

Στα πειράματα της κατηγορίας Γ, όπως είπαμε, μεταβάλουμε όλες τις παραμέτρους του συστήματος και είναι ίσως τα πιο αντιπροσωπευτικά του πραγματικού κόσμου.

Κι εδώ παρατηρούμε τις καλές επιδόσεις των αλγορίθμων μας ενώ διαπιστώνουμε τη σταθερότητα του **Fair Load–Tie Resolver$_2$** και την διαφοροποίηση του **Fair Load–Merge Messages' Ends** από τους υπόλοιπους ο οποίος βελτιώνει στις περισσότερες περιπτώσεις αρκετά το χρόνο εκτέλεσης με κόστος όμως την χειροτέρευση της κατανομής φορτίου. Στο κάτω δεξιά πείραμα του Σχήματος 5.9 όπου χρησιμοποιούμε δίαυλο ταχύτητας 10 Mbps (σε αντίθεση με τα υπόλοιπα τρία όπου χρησιμοποιούμε ταχύτητα 100 Mbps), βλέπουμε τη διαφορά στην κλίμακα του άξονα του χρόνου εκτέλεσης και διαπιστώνουμε ότι οι **Tie Resolver** αλγόριθμοι είναι περισσότερο απαραίτητοι όταν δεν υπάρχουν γρήγορες συνδέσεις.



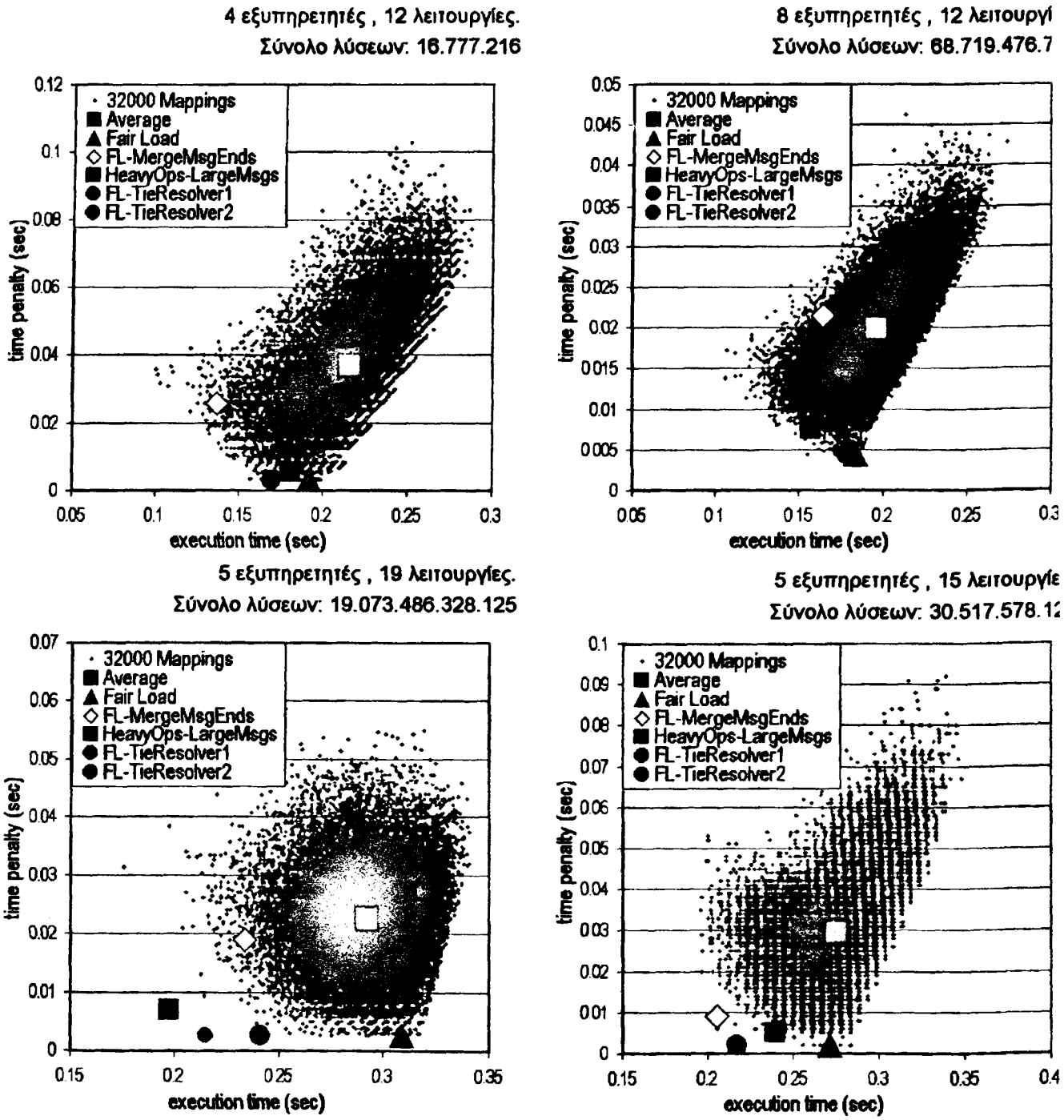
Σχήμα 5.9 Πειράματα Κατηγορίας Γ για τους Αλγορίθμους Γραμμή–Δίαυλος.



Σχήμα 5.10 Επίδοση του Αλγορίθμων Γραμμή–Δίαυλος σε Μεγαλύτερα Συστήματα με Δίαυλο Ταχύτητας 100 Mbps.

Στα Σχήματα 5.10 και 5.11 απεικονίζονται πειράματα της κατηγορίας Γ για μεγαλύτερα συστήματα με τη διαφορά ότι στα πειράματα του πρώτου σχήματος ο δίαυλος επικοινωνίας μεταξύ των εξυπηρετητών είναι 100 Mbps ενώ στα πειράματα του δεύτερου σχήματος ο δίαυλος επικοινωνίας μεταξύ των εξυπηρετητών είναι 10 Mbps. Στο Σχήμα 5.10, παρατηρούμε ότι ο αλγόριθμος που διαφοροποιεί τη λύση του από τους υπόλοιπους είναι ο Fair Load–Merge Messages' Ends που βελτιώνει το χρόνο

εκτέλεσης, χειροτερεύοντας παράλληλα την κατανομή του φορτίου. Όλοι οι υπόλοιποι δίνουν λύσεις με μικρές αποκλίσεις μεταξύ τους. Στο Σχήμα 5.11 βλέπουμε ότι οι διαφοροποιήσεις των αλγορίθμων είναι σημαντικότερες επειδή έχουμε μικρότερη ταχύτητα διαύλου και πρέπει να λαμβάνονται υπόψην όλοι οι αλγόριθμοι.



Σχήμα 5.11 Επίδοση του Αλγορίθμων Γραμμή–Δίαυλος σε Μεγαλύτερα Συστήματα με Δίαυλο Ταχύτητας 10 Mbps.

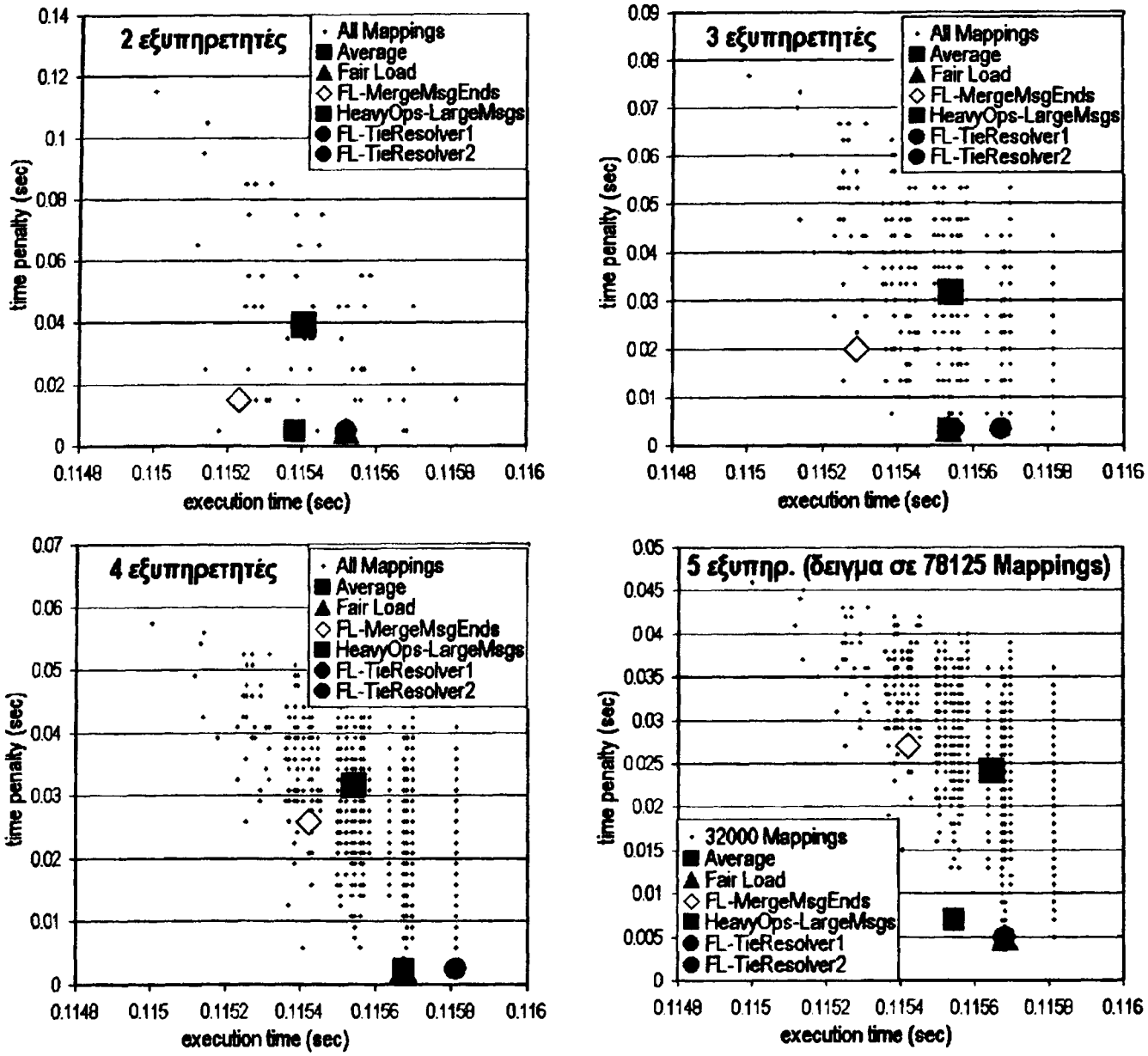
Επίσης οι αλγόριθμοι Γραμμή–Δίαυλος εφαρμόστηκαν και στο workflow του Σχήματος 3.4, που περιγράφει την επικοινωνία ενός ασθενή με το γιατρό του. Τα μεγέθη των μηνυμάτων καθώς και οι κύκλοι των λειτουργιών του workflow, που χρησιμοποιήθηκαν φαίνονται στους Πίνακες 5.3 και 5.4 αντίστοιχα. Στο Σχήμα 5.12 φαίνονται τα αποτελέσματα των πειραμάτων για τις περιπτώσεις που το workflow εκτελείται σε 2, 3, 4 και 5 εξυπηρετητές. Η υπολογιστική ισχύς των εξυπηρετητών θεωρήθηκε 1 GHz και η ταχύτητα της σύνδεσης του διαύλου 100 Mbps. Στα πειράματα με 2, 3 και 4 εξυπηρετητές, παράγουμε όλες τις λύσεις ενώ στο πείραμα με τους 5 εξυπηρετητές, παράγουμε ένα τυχαίο δείγμα 32000 λύσεων στο σύνολο των 78125 λύσεων. Τα αποτελέσματα είναι παρόμοια με αυτά που είδαμε μέχρι στιγμής στους αλγορίθμους Γραμμή–Δίαυλος. Στα πειράματα με 3 και 4 εξυπηρετητές παρατηρούμε ότι οι Tie Resolver αλγόριθμοι μπορεί να χειροτερεύουν το χρόνο εκτέλεσης του Fair Load και αυτό συμβαίνει γιατί μια απόφαση σε πρώιμο βήμα του αλγορίθμου που οδηγεί σε τοπικά βέλτιστη λύση μπορεί να επηρεάσει την τελική λύση.

Πίνακας 5.3 Τα Μεγέθη των Μηνυμάτων του Workflow του Σχήματος 3.4.

Μήνυμα	m_1	m_5	m_6	m_8	m_9	m_{10}
Μέγεθος (bytes)	500	200	1000	500	200	400

Πίνακας 5.4 Οι Κύκλοι των Λειτουργιών του Workflow του Σχήματος 3.4.

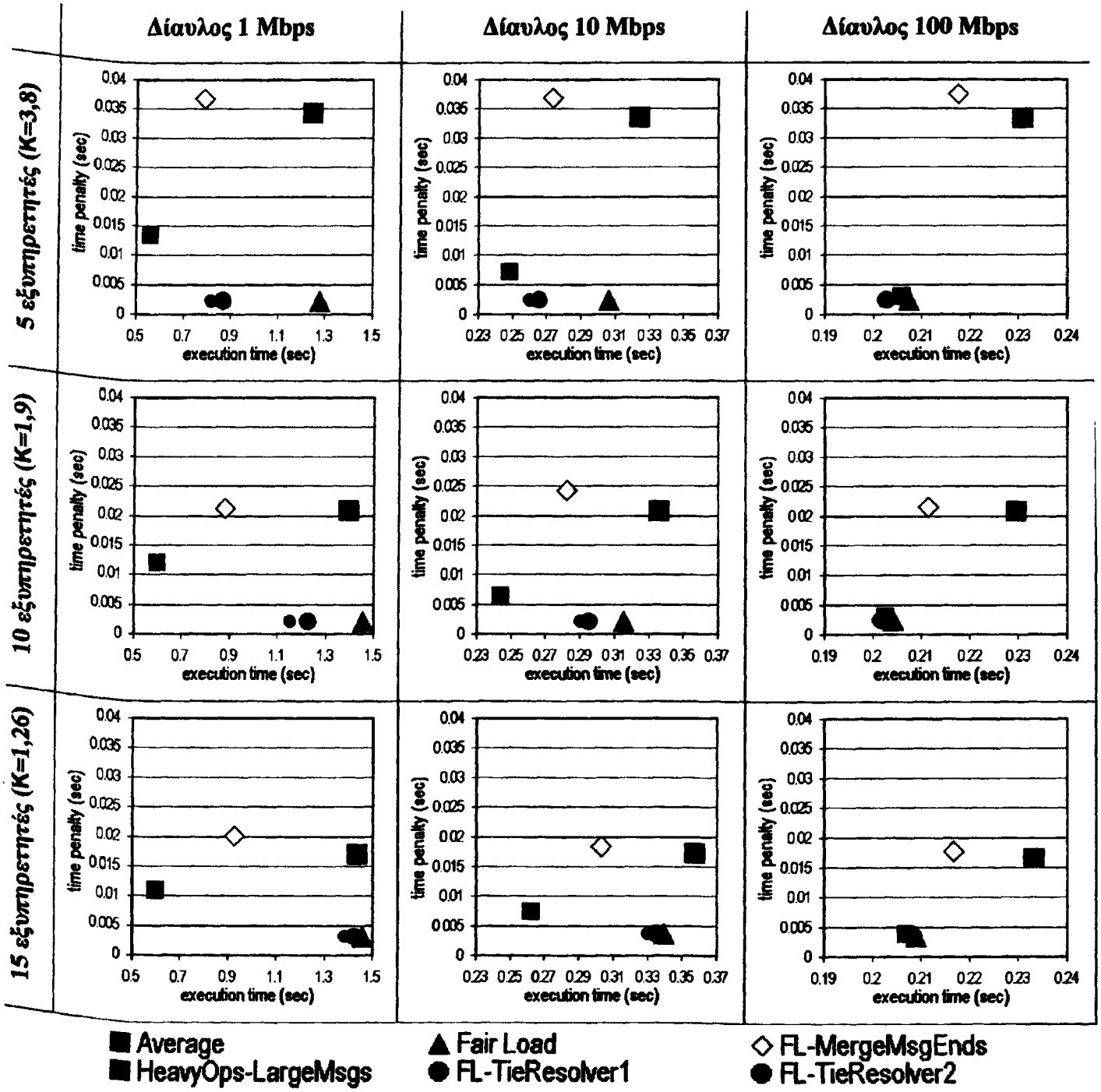
Λειτουργία / Λειτουργίες	1	2, 3, 4, 5, 6	7	8, 9, 10	11	12	13
Κύκλοι ($\times 10^6$)	5	20	30	15	20	15	10



Σχήμα 5.12 Επίδοση του Αλγορίθμων Γραμμή–Δίαυλος στο Workflow του Σχήματος 3.3.

Στο Σχήμα 5.13 κάνουμε μία σύγκριση των αλγορίθμων Γραμμή – Δίαυλος παίρνοντας τη μέση τιμή των λύσεων των αλγορίθμων σε 50 πειράματα κατηγορίας Γ. Στα πειράματα μελετούμε τρεις διαφορετικές ταχύτητες διαύλου, 1 Mbps, 10 Mbps και 100 Mbps, και τρία διαφορετικά δίκτυα εξυπηρετητών, με 5, 10 και 15 εξυπηρετητές. Τα παραπάνω συνδυάζονται με εννέα διαφορετικούς τρόπους, όσα είναι και τα κελιά στο Σχήμα 5.13. Το κάθε ένα από αυτά τα κελιά απεικονίζει τα αποτελέσματα από 50 πειράματα με τυχαία workflows 19 λειτουργιών. Σε όλα τα πειράματα γίνεται δειγματοληψία των λύσεων από όπου προκύπτει και το Average των λύσεων.

Το πρώτο συμπέρασμα που προκύπτει ειναι ότι ο αλγόριθμος Heavy Operations–Large Messages είναι ο πιο σταθερός σε σχέση με τους υπόλοιπους στη μεταβολή του παράγοντα ομαδοποίησης K. Βλέπουμε ότι οι Tie Resolver δεν βελτιώνουν πολύ τον χρόνο εκτέλεσης του Fair Load όσο μικραίνει ο παράγοντας ομαδοποίησης.



Σχήμα 5.13 Σύγκριση Αλγορίθμων Γραμμή–Δίαυλος για Workflow με 19 Λειτουργίες, Κατηγορίας Γ.

Ένα δεύτερο συμπέρασμα προκύπτει για τον Fair Load–Merge Messages' Ends, ο οποίος καλυτερεύει τον χρόνο εκτέλεσης του Fair Load–Tie Resolver$_2$ (στον οποίο βασίζεται) για μικρές ταχύτητες διαύλου ενώ αντίθετα χειροτερεύει το χρόνο εκτέλεσης για μεγάλες ταχύτητες διαύλου. Επίσης, οι λύσεις του σε όλες τις περιπτώσεις υστερούν από αυτές του Heavy Operations–Large Messages και στους δύο άξονες. Ένα τρίτο συμπέρασμα που προκύπτει είναι ότι για μεγάλες ταχύτητες διαύλου όλοι οι αλγόριθμοι εκτός του Fair Load–Merge Messages' Ends δίνουν το ίδιο ικανοποιητικές λύσεις και οι βελτιώσεις που κάνουν στο χρόνο εκτέλεσης είναι μικρές σε σχέση με τις περιπτώσεις όπου χρησιμοποιείται μικρότερη ταχύτητα διαύλου. Ως γενικό συμπέρασμα από το Σχήμα 5.13 μπορούμε να πούμε ότι ο αλγόριθμος Fair Load–Tie Resolver$_2$ είναι ο πιο αξιόλογος όταν θέλουμε κυρίως δίκαιη κατανομή φορτίου, ενώ ο αλγόριθμος Heavy Operations–Large Messages είναι ο πιο αξιόλογος όταν θέλουμε μικρό χρόνο εκτέλεσης χάνοντας λίγο από τη δίκαιη κατανομή φορτίου.

5.4. Πειράματα για τους Αλγορίθμους Τυχαίος Γράφος–Δίαυλος

Στην ενότητα αυτή περιγράφονται πειράματα που έγιναν για την περίπτωση που η τοπολογία του workflow είναι τυχαίος γράφος και η τοπολογία του δικτύου είναι δίαυλος. Αρχικά το workflow που χρησιμοποιούμε στα πειράματα είναι αυτό του παραδείγματος του 3^{ου} κεφαλαίου (Σχήμα 3.3), που περιγράφει την επικοινωνία ενός ασθενή με τον γιατρό του. Τα μεγέθη των μηνυμάτων καθώς και οι κύκλοι των λειτουργιών του workflow, που χρησιμοποιήθηκαν φαίνονται στους Πίνακες 5.5 και 5.6 αντίστοιχα. Στον Πίνακα 5.5 δεν υπάρχουν τα μηνύματα που στέλνονται προς τις conditional λειτουργίες του workflow, δηλαδή τις λειτουργίες 5, 9, 14 και 15. Στις λειτουργίες 5 και 9 θεωρούμε ότι στέλνονται και τα δύο μηνύματα που προωθούν αυτές στη συνέχεια στους επιμέρους κλάδους. Δηλαδή, στη λειτουργία 5 στέλνονται από την λειτουργία 4 τα μηνύματα m_4 και m_5 (400 bytes), ενώ στη λειτουργία 9 στέλνονται από την λειτουργία 8 τα μηνύματα m_7 και m_8 (1000 bytes). Τα μηνύματα που στέλνονται στις λειτουργίες 14 και 15 είναι μηνύματα επιβεβαίωσης και θεωρούμε το μέγεθός τους ίσο με 60 bytes, περίπου το ελάχιστο μέγεθος πακέτου που μπορεί να σταλεί μέσω TCP. Στο Σχήμα 5.14 φαίνονται τα αποτελέσματα των

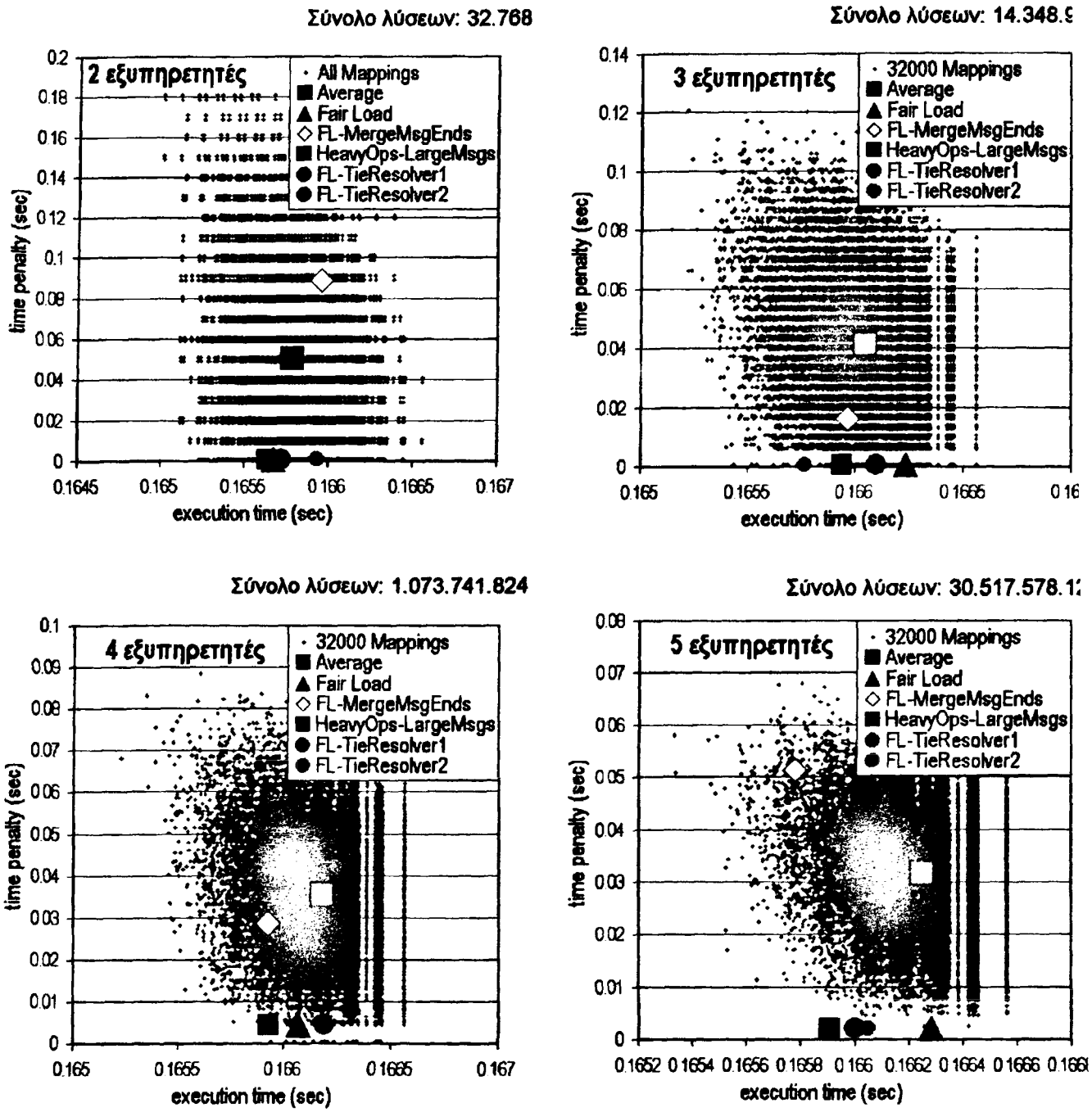
πειραμάτων για τις περιπτώσεις που το workflow εκτελείται σε 2, 3, 4 και 5 εξυπηρετητές. Για τις περιπτώσεις των 3, 4 και 5 εξυπηρετητών έγινε δειγματοληψία και δεν παράχθηκαν όλες οι λύσεις επειδή είναι πάρα πολλές. Η υπολογιστική ισχύς των εξυπηρετητών θεωρήθηκε 1 GHz και η ταχύτητα της σύνδεσης του διαύλου 100 Mbps. Από το Σχήμα 5.14 δεν προκύπτουν πολλά συμπεράσματα. Το μόνο που μπορούμε να διακρίνουμε είναι ότι για μικρό αριθμό εξυπηρετητών (2-3 εξυπηρετητές) οι αλγόριθμοι δεν δίνουν πολύ καλές λύσεις.

Πίνακας 5.5 Τα Μεγέθη των Μηνυμάτων του Workflow του Σχήματος 3.3.

Μήνυμα	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}
Μέγεθος (bytes)	500	300	100	200	200	1000	500	500	200	400

Πίνακας 5.6 Οι Κύκλοι των Λειτουργιών του Workflow του Σχήματος 3.3.

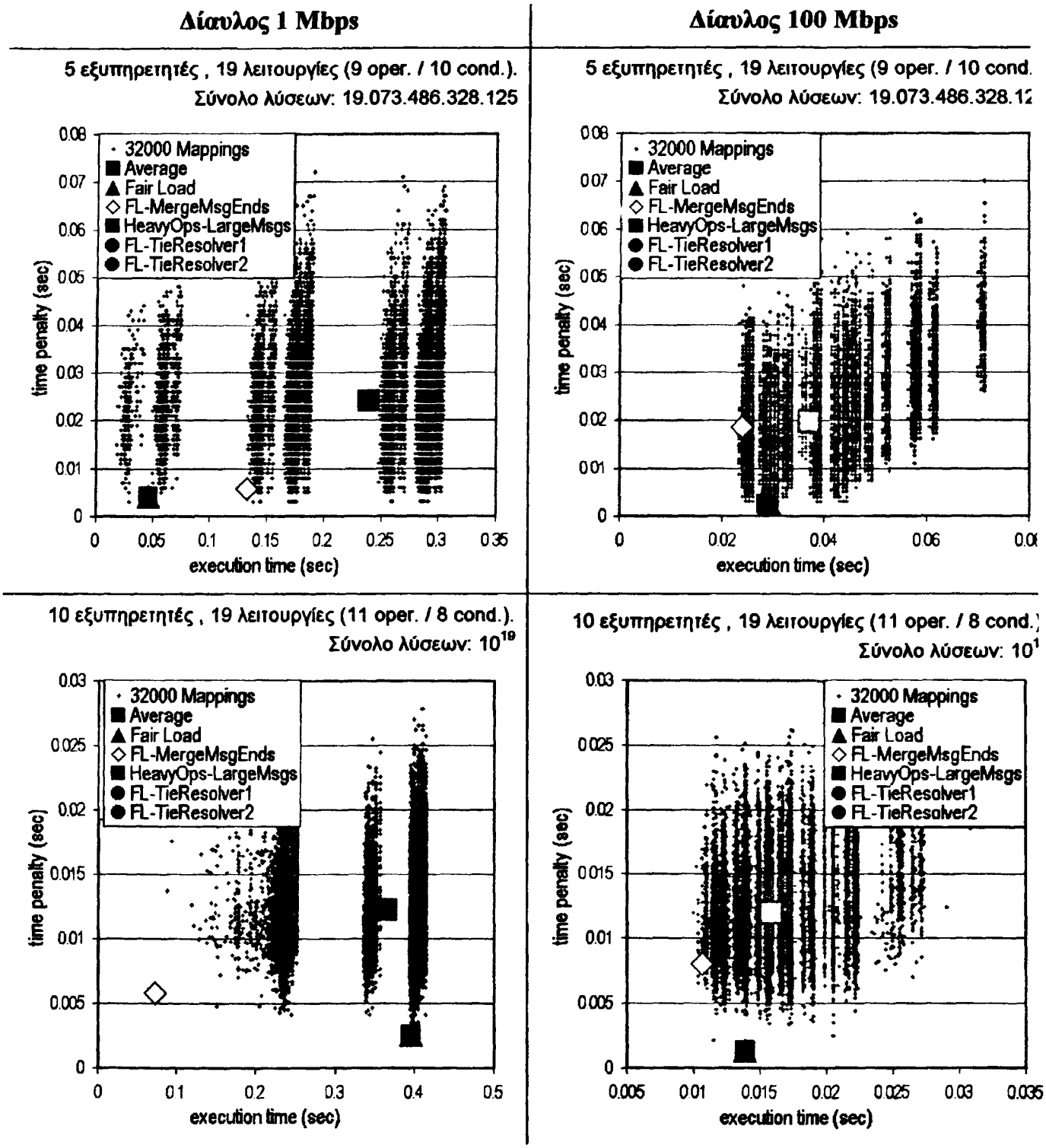
Λειτουργία	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Κύκλοι ($\times 10^6$)	5	30	10	20	0.001	1	30	25	0.001	15	20	15	10	0.001	0.001



Σχήμα 5.14 Επίδοση του Αλγορίθμων Τυχαίος Γράφος–Δίαυλος στο Workflow του Σχήματος 3.2.

Στο Σχήμα 5.15 βλέπουμε τέσσερα διαφορετικά πειράματα σε bushy workflows με 19 λειτουργίες. Διαφέρουν στον αριθμό των εξυπηρετητών και στην ταχύτητα του διαύλου. Φαίνεται ότι ο αλγόριθμος που διαφοροποιείται από τους πέντε είναι ο Fair Load–Merge Messages' Ends, ο οποίος μπορεί να βελτιώσει πολύ τη λύση που δίνουν οι υπόλοιποι, ειδικά όταν η ταχύτητα του διαύλου είναι μικρή (π.χ., στο κάτω αριστερά πείραμα). Επίσης, ο συγκεκριμένος αλγόριθμος μπορεί να δώσει χειρότερες

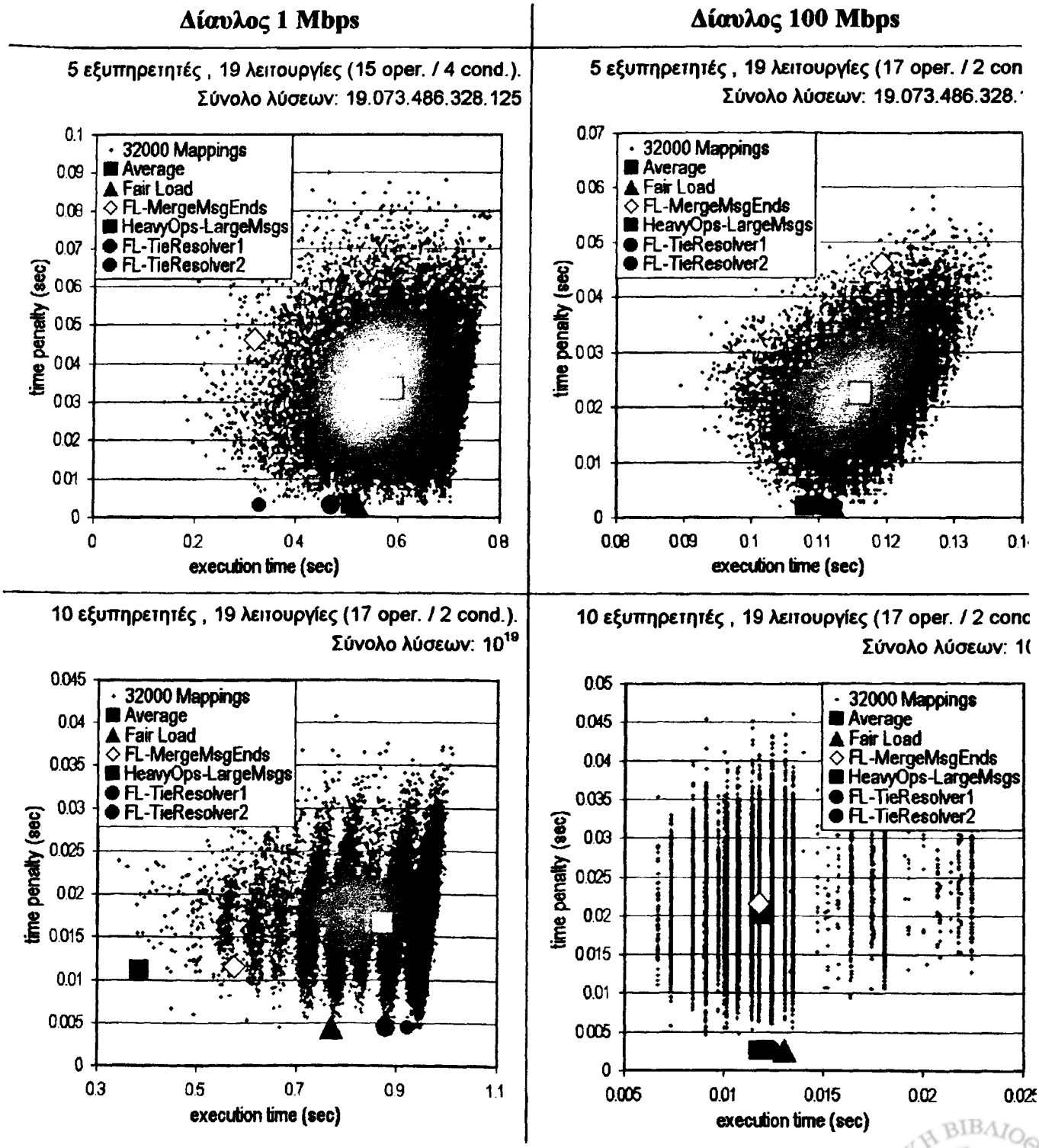
λύσεις από τους υπόλοιπους (π.χ., στο πάνω αριστερά πείραμα). Γενικότερα, με τη χρήση των αλγορίθμων που προτείνουμε μπορούμε να πάρουμε ικανοποιητικές λύσεις σε bushy workflows.



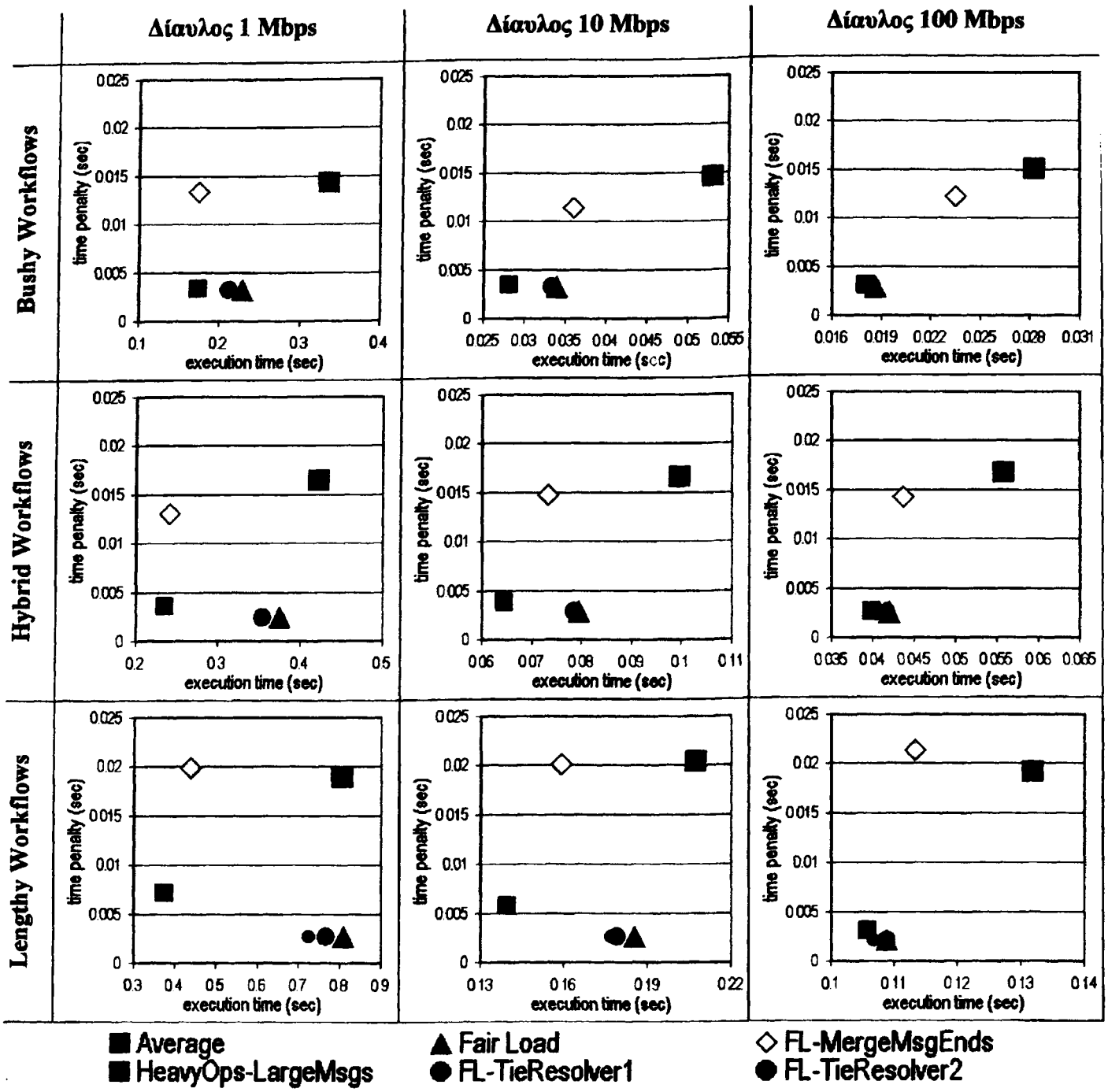
Σχήμα 5.15 Επίδοση του Αλγορίθμων Τυχαίος Γράφος–Δίαυλος σε Bushy Workflows.

Στο Σχήμα 5.16 κάνουμε τα ίδια πειράματα που περιγράψαμε για το Σχήμα 5.15 για τα bushy workflows, με τη διαφορά σε αυτά τα πειράματα έχουμε lengthy workflows.

Στα πειράματα με δίαυλο ταχύτητας 100 Mbps οι λύσεις που δίνουν οι αλγόριθμοι δεν είναι πολύ ικανοποιητικές, ενώ παρατηρούμε ότι ο Fair Load–Merge Messages' Ends έχει χειρότερη λύση από το μέσο όρο των λύσεων. Στα πειράματα με δίαυλο 1 Mbps παίρνουμε καλύτερες λύσεις. Κι εδώ βλέπουμε ότι οι Tie Resolver μπορεί να είναι χειρότεροι από τον Fair Load (στον οποίο βασίζονται).



Σχήμα 5.16 Επίδοση του Αλγορίθμων Τυχαίος Γράφος–Δίαυλος σε Lengthy Workflows.



Σχήμα 5.17 Σύγκριση Αλγορίθμων Τυχαίος Γράφος–Δίαυλος για Workflows με 19 Λειτουργίες και 10 Εξυπηρετητές, Κατηγορίας Γ.

Τέλος για τη σύγκριση των αλγορίθμων Τυχαίος Γράφος–Δίαυλος παρουσιάζουμε πειράματα κατηγορίας Γ, στο Σχήμα 5.17, που κάναμε σε πολλά workflows και πήραμε τη μέση τιμή των λύσεων των αλγορίθμων. Στα πειράματα μελετάμε τους τρεις τύπους workflow, bushy, hybrid και lengthy, και τρεις διαφορετικές ταχύτητες διαύλου, 1Mbps, 10 Mbps και 100 Mbps. Τα παραπάνω συνδυάζονται με εννέα διαφορετικούς τρόπους, όσα είναι και τα κελιά στο Σχήμα 5.13. Το κάθε ένα από

αυτά τα κελιά απεικονίζει τα αποτελέσματα από 50 πειράματα με τυχαία workflows 19 λειτουργιών και 10 εξυπηρετητών. Σε όλα τα πειράματα γίνεται δειγματοληψία των λύσεων από όπου προκύπτει και το Average των λύσεων.

Το πρώτο πράγμα που παρατηρεί κανείς κοιτάζοντας το σχήμα είναι ότι ο Heavy Operations–Large Messages δίνει πολύ καλές λύσεις, με κατανομή φορτίου πολύ κοντά σε αυτή του Fair Load αλλά με καλύτερο χρόνο εκτέλεσης, και ειδικά στα πιο θαμνώδη workflow. Για τον Fair Load–Merge Messages' Ends ισχύει ότι και στην περίπτωση Γραμμή–Δίαυλος. Βελτιώνει δηλαδή τον Fair Load–Tie Resolver$_2$ μόνο όταν η ταχύτητα του διαύλου είναι 1 Mbps. Όμως παραμένει χειρότερος από τον Heavy Operations–Large Messages. Ένα άλλο συμπέρασμα που συμπίπτει επίσης με αυτό της περίπτωσης Γραμμή–Δίαυλος, είναι ότι σε συστήματα με δίαυλο ταχύτητας 100 Mbps οι τέσσερις αλγόριθμοι (όλοι εκτός του Fair Load–Merge Messages' Ends) δίνουν μια ικανοποιητική λύση και δεν διαφέρουν σημαντικά μεταξύ τους. Αντίθετα, για μικρή ταχύτητα διαύλου (1 Mbps) οι διαφορές των αλγορίθμων είναι σημαντικές και πρέπει να λαμβάνονται υπόψην. Ο πιο αξιόλογος αλγόριθμος στα πειράματα αυτά είναι ο Heavy Operations–Large Messages και ακολουθεί ο Fair Load–Tie Resolver$_2$.

5.5. Ανακεφαλαίωση

Συνοπτικά, τα συμπεράσματα των πειραμάτων είναι τα εξής:

- Στην τοπολογία Γραμμή–Γραμμή: στην περίπτωση αυτή ο καταλληλότερος αλγόριθμος είναι ο διπλής κατεύθυνσης Γραμμή–Γραμμή. Η επιδιόρθωση των κακών γεφυρών πρέπει να γίνεται όταν το κλάσμα ταχύτητα συνδέσεων προς μέγεθος μηνυμάτων είναι πολύ μικρό.
- Στην τοπολογία Γραμμή–Δίαυλος: όταν ο χρήστης επιθυμεί δίκαιη κατανομή φορτίου, καταλληλότερος αλγόριθμος είναι ο Fair Load–Tie Resolver$_2$, ενώ όταν επιθυμεί μικρό χρόνο εκτέλεσης οι καταλληλότεροι αλγόριθμοι είναι οι Fair Load–Merge Messages' Ends και Heavy Operations–Large Messages, με τον πρώτο να μην ενδείκνυται για περιπτώσεις όπου όλα τα μηνύματα είναι σχεδόν ίδιου μεγέθους (Κατηγορία Β) και όταν η ταχύτητα του διαύλου είναι μεγάλη.

- Στην τοπολογία Γραμμή–Τυχαίος Γράφος: ισχύουν τα ίδια με την προηγούμενη τοπολογία, με το επιπλέον σχόλιο ότι για θαμνώδη workflows ο **Fair Load–Merge Messages' Ends** είναι προτιμότερος από τον **Heavy Operations–Large Messages** όταν ο χρήστης επιθυμεί μικρό χρόνο εκτέλεσης και η ταχύτητα του διαύλου είναι μικρή.

ΚΕΦΑΛΑΙΟ 6. ΕΠΙΛΟΓΟΣ

6.1 Ανακεφαλαίωση
6.2 Μελλοντικές Επεκτάσεις

6.1. Ανακεφαλαίωση

Το πρόβλημα με το οποίο ασχοληθήκαμε είναι, δοθέντος μιας ροής εργασίας υπηρεσιών διαδικτύου και ενός δικτύου εξυπηρετητών, πώς πρέπει να κατανεμηθούν οι λειτουργίες της ροής εργασίας στους εξυπηρετητές ώστε να ελαχιστοποιείται μία συνάρτηση κόστους και να ικανοποιούνται οι περιορισμοί του χρήστη. Εμείς, χρησιμοποιήσαμε δύο συναρτήσεις κόστους που αφορούν το χρόνο εκτέλεσης της ροής εργασίας και την κατανομή του φορτίου στους εξυπηρετητές. Επικεντρωθήκαμε στην περίπτωση που ο χρήστης θέλει να ελαχιστοποιήσει και τις δύο συναρτήσεις κόστους, δηλαδή, να ελαχιστοποιήσει το χρόνο εκτέλεσης της ροής εργασίας και να κατανείμει όσο πιο δίκαια γίνεται το φορτίο στους εξυπηρετητές, να απασχολούνται δηλαδή, όλοι οι εξυπηρετητές τον ίδιο χρόνο. Μελετήσαμε ξεχωριστά διαφορετικές τοπολογίες της ροής εργασίας και του δικτύου εξυπηρετητών και προτείνουμε αλγορίθμους για την εύρεση λύσης σε κάθε περίπτωση. Είδαμε ότι η εύρεση της βέλτιστης λύσης στο πρόβλημα, δεν είναι πάντα εφικτή γιατί έχει εκθετική πολυπλοκότητα. Αντιθέτως, οι αλγόριθμοί μας έχουν πολυωνιμική πολυπλοκότητα. Η συνεισφορά της εργασίας είναι η διαστασιοποίηση του σύνθετου προβλήματος που αναφέραμε, ο ορισμός ενός απλού μοντέλου για την περιγραφή του, οι αλγόριθμοι

που προτείναμε για την επίλυση του, καθώς και τα πειράματα που έδειξαν την επίδοση των αλγορίθμων για διαφορετικές αναθέσεις στις παραμέτρους του συστήματος. Το μοντέλο που προτείναμε βασίζεται στις εξής παραμέτρους: υπολογιστική ισχύς των εξυπηρετητών, υπολογιστική ισχύς που χρειάζονται οι λειτουργίες για να εκτελεστούν, μηνύματα που στέλνονται μεταξύ των λειτουργιών και ταχύτητες των συνδέσεων μεταξύ των εξυπηρετητών. Οι αλγόριθμοί μας έδωσαν ικανοποιητικές λύσεις, ενώ από τα πειράματα είδαμε ποιος είναι ο καταλληλότερος σε κάθε περίπτωση.

6.2. Μελλοντικές Επεκτάσεις

Η συγκεκριμένη εργασία, ασχολήθηκε με την κατανομή ενός workflow σε ένα δίκτυο εξυπηρετητών. Ενδιαφέρον αποτελεί η περίπτωση που θέλουμε να κατανείμουμε πολλά workflows σε ένα δίκτυο εξυπηρετητών. Σε αυτήν την περίπτωση πρέπει να εξετασθεί πώς μπορούν να χρησιμοποιηθούν οι αλγόριθμοι που προτείναμε και να ελεγχθεί τι επιδόσεις έχουν. Επίσης, η εργασία αυτή μπορεί να συνεχισθεί με την τροποποίηση του μοντέλου που προτείναμε ώστε να λαμβάνονται υπόψη και άλλες παράμετροι του συστήματος. Για παράδειγμα, ο χρόνος απόκρισης είναι ένα μέτρο αξιολόγησης που θα μπορούσε να μελετηθεί αν στο μοντέλο υπήρχαν πολλές αιτήσεις για το workflow οπότε σχηματιζόταν ουρές προτεραιότητας.

ΑΝΑΦΟΡΕΣ

[ACKM04] G. Alonso, F. Casati, H. Kuno, V. Machiraju. Web Services Concepts, Architecture and Applications. Sprinter, 2004.

[CoBF05] I. Constantinescu, W. Binder, B. Faltings. Optimally Distributing Interactions Between Composed Semantic Web Services. ESWC 2005: 32-46.

[CSMA+04] J. Cardoso, A. Sheth, J. Miller, J. Arnold, K. Kochut. Quality of Service for Workflows and Web Service Processes. Journal of Web Semantics, 1(3): 281-308. 2004.

[DHM+04] X. Dong, A. Y. Halevy, J. Madhavan, E. Nemes, J. Zhang: Simlarity Search for Web Services. VLDB 2004: 372-383 2004.

[DoZV05] C. Doulkeridis, V. Zafeiris, M. Vazergiannis. The Role of Caching and Context-Awareness in P2P Service Discovery. MDM, Ayia Napa, Cyprus. 2005.

[ESAA04] F. Emekçi, O. D. Sahin, D. Agrawal, A. El Abbadi: A Peer-to-Peer Framework for Web Service Discovery with Ranking. ICWS 2004: 192-199.

[HGSL+05] M. Head, M. Govindaraju, A. Slominski, P. Liu, N. Abu-Ghazaleh, R. van Engelen, K. Chiu, M. Lewis. A Benchmark Suite for SOAP-based Communication in Grid Web Services. SC|05, Seatle WA. November 2005.

[GiWW02] M. Gillmann, G. Weikum, W. Wonner. Workflow Management with Service Quality Guarantees. ACM SIGMOD. Madison, Wisconsin, USA. 2002.

[Kreg01] H. Kreger (IBM Software Group). Web Sevices Conceptual Architecture (WSCA 1.0). May 2001. Διαθέσιμο στη σελίδα: http:www-900.ibm.com/developerWorks/webservices/ws-wsca/part1/index.shtml

[LeWY93] A. Leff, J.L. Wolf, P.S. Yu. Replication algorithms in a remote caching architecture. IEEE Transactions on Parallel and Distributed Systems, 4(11), 1185-1204, Nov. 1993.

[LiZh05] J. Liu, H. Zhuge. A Semantic-Link-Based Infrastracture for Web Service Discovery in P2P Networks. WWW2005, May 10-14, 2005, China, Japan.

[LTZS05] N. Laoutaris, O. Telelis, V. Zissimopoulos, I. Stavrakakis. Distributed Selfish Replication. IEEE Transactions on Parallel and Distributed Systems, 2005.

[NgCG04] A. Ng, S. Chen, P. Greenfield. An Evaluation of Contemporary Commercial SOAP Implementations. AWSA2004. Melbourne, Australia, 2004.

[OMG06] Object Management Group. Business Process Management Initiative. Διαθέσιμο στη σελίδα http://bpmn.org/Samples/Choreography/DoctorVisit/VisitIndex.htm. Ιούνιος 2006.

[SaZh04] R. Salellariou, H. Zhao. A Low-Cost Rescheduling Policy for Efficient Mapping of Workflows on Grid Systems. Scientific Programming, 12(4), 253-262, December 2004.

[ScPa04] C. Schmidt, M. Parashar. A Peer-to-Peer Approach to Web Service Discovery. World Wide Web Journal, 7(2), June 2004.

[SMKK+01] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. SIGCOMM. 2001.

[SWMM05] U. Srivastava, J. Widom, K. Munagala, R. Motwani. Query Optimization over Web Services. October 2005. Διαθέσιμο στη σελίδα: http://dbpubs.stanford.edu:8090/pub/2005-30.

[ZBD+03] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnam, Q. Sheng. Quality Driven Web Services Composition. WWW2003, May 20-24, 2003, Budapest, Hungary.

ΣΥΝΤΟΜΟ ΒΙΟΓΡΑΦΙΚΟ

Ο Κωνσταντίνος Σταμκόπουλος γεννήθηκε το 1982 και μεγάλωσε στο Πλατανόρρευμα Κοζάνης. Το έτος 2000 εισήχθη στο τμήμα Πληροφορικής του Πανεπιστημίου Ιωαννίνων από όπου αποφοίτησε το έτος 2004, ενώ ξεκίνησε τις μεταπτυχιακές του σπουδές στο ίδιο τμήμα.