Ατομική Διπλωματική Εργασία

ΜΕΛΕΤΗ ΕΞΙΣΟΡΡΟΠΗΣΗΣ ΦΟΡΤΙΟΥ ΒΑΣΙΣΜΕΝΗ ΣΕ ΑΛΓΟΡΙΘΜΟΥΣ ΜΕΤΑΝΑΣΤΕΥΣΗΣ ΠΛΗΘΥΣΜΩΝ

Χρυσή Σέα

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2013

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μελέτη εξισορρόπησης φορτίου βασισμένη σε αλγόριθμους μετανάστευσης πληθυσμών

Χρυσή Σέα

Επιβλέπουσα Καθηγήτρια
Άννα Φιλίππου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Ευχαριστίες

Τα χρόνια των σπουδών μου ήταν για μένα ένα όμορφο ταξίδι, με πολλές εμπειρίες, αρκετές προκλήσεις και έναν μαγικό βυθό γνώσεων. Η πορεία δύσκολη, μαχητική αλλά με μια γλυκιά κούραση και μιαν ικανοποίηση ότι έδωσα τον καλύτερο μου εαυτό, ότι ξεπέρασα τον εαυτό μου.

Στο ταξίδι αυτό αισθάνομαι ευλογημένη που είχα δίπλα μου αγαπημένους μου ανθρώπους που η συμβολή τους ήταν πολύτιμη και καθοριστική ώστε το ταξίδι μου να φτάσει στην Ιθάκη. Επί ευκαιρίας της ολοκλήρωσης της Ατομικής Διπλωματικής μου Εργασίας, που αποτελεί το τελευταίο στάδιο της ολοκλήρωσης του πτυχίου μου, θα ήθελα να εκφράσω από καρδιάς τις θερμές ευχαριστίες μου.

Πρωτίστως ευχαριστώ το Θεό, που με αξίωσε να φτάσω στο τέρμα του ταξιδιού μου με στήριζε σε κάθε μου βήμα και σε κάθε δυσκολία μου έδινε τη δύναμη να συνεχίσω. Ευχαριστώ τον πνευματικό μου πατέρα, π. Γεώργιο Σεργίδη, που με στήριζε με τις συμβουλές και τις προσευχές του. Ευχαριστώ τους γονείς μου που συνέβαλαν με την υπομονή και το ενδιαφέρον τους και τα αγαπημένα μου αδέλφια που με μεγάλη προθυμία έβαλαν το λιθαράκι τους στην παρούσα εργασία. Ευχαριστώ τους φίλους μου, για την πλήρη κατανόηση τους όταν σε περιόδους «κρίσης» επιβαλλόταν να περιοριστώ με πλήρη αφοσίωση στις σπουδές μου στερώντας τους την παρέα μου. Τέλος, αισθάνομαι την ανάγκη να ευχαριστήσω την επιβλέπουσα καθηγήτρια μου, Δρα Άννα Φιλίππου, για καθοδήγηση σε κάθε στάδιο της εργασίας μου.

Κλείνοντας, θα ήθελα σε όλους τους αναγνώστες να υπενθυμίσω την φράση του Ισοκράτη: «Τῆς παιδείας αἱ μέν ῥίζαι πικραί οἱ δε καρποί γλυκεῖς». Ο λόγος του είναι για μένα η επιβεβαίωση ότι το ταξίδι αξίζει!

Περίληψη

Στην παρούσα διπλωματική εργασία μελετούμε το πρόβλημα της εξισορρόπησης φόρτου εργασίας των χρηστών ενός δικτύου σε ένα μοντέλο το οποίο ξεχωρίζει από το γεγονός ότι θεωρούμε πως οι κόμβοι του δικτύου έχουν κάποια χωρητικότητα. Θεωρούμε επαναληπτικό αλγόριθμο που καταλήγει σε κατάσταση ισορροπίας εμπνευσμένος από τη διαδικασία που ακολουθούν τα πουλιά προς μετανάστευση που ονομάζω Αλγόριθμο 2.

Ο αλγόριθμος αυτός συγκρίνεται με προσομοίωση του αλγορίθμου των Petra Berenbrink, Tom Friedetzky, Leslie Ann Goldberg, Paul W. Goldberg, Zengjian Hu, Russell A. Martin, όπως περιγράφεται στο άρθρο [2], λαμβάνοντας υπ όψιν και την έννοια της χωρητικότητας την οποία μελετούμε στο δικό μας πρόβλημα και ονομάζω Αλγόριθμο 1. Στόχος είναι να μελετηθεί κατά πόσο το σύστημα φτάνει γρηγορότερα σε ισορροπία με παραλλαγές των αλγορίθμων, θεωρώντας ισορροπία την κατάσταση κατά την οποία κανένας χρήστης δεν επιθυμεί να αλλάξει κόμβο εξυπηρέτησης. Παρατηρήσαμε ότι ο αλγόριθμος βασισμένος στον αλγόριθμο μετανάστευσης πληθυσμών φτάνει πολύ γρηγορότερα σε κατάσταση ισορροπίας.

Στη συνέχεια, ο Αλγόριθμος που είχε τα καλύτερα αποτελέσματα, Αλγόριθμος 2, μελετήθηκε πειραματικά αλλά και θεωρητικά. Συγκεκριμένα, υλοποιήσαμε τον αλγόριθμο με διαφορετικά χαρακτηριστικά, χωρητικότητα κόμβων, φορτίο χρηστών, τοπολογία και μέγεθος δικτύου καταλήγοντας σε συμπεράσματα ως προς το χρόνο σύγκλισης σε δίκτυα με διαφορετικά χαρακτηριστικά. Σε γενικές γραμμές, συμπεραίνουμε ότι σε υπερφορτωμένα δίκτυα οι χρήστες πρέπει να μεταναστεύουν με μεγάλη πιθανότητα και δίκτυα με τοπολογία αστέρι δίνουν ταχύτερους χρόνους σύγκλισης Επιπλέον, μια ομάδα από κεντρικούς κόμβους στην οποία ο χρήστης θα έχει διπλάσια πιθανότητα αρχικής επιλογής για εξυπηρέτηση, μπορεί να φέρει καλύτερα αποτελέσματα με μικρές πιθανότητες μετανάστευσης των χρηστών.

Θεωρητικά ο Αλγόριθμος μελετήθηκε χρησιμοποιώντας Θεωρία Παιγνίων [8,9] και επιζητώντας κατά πόσο το μοντέλο διαθέτει καταστάσεις ισορροπίας. Η μελέτη έδειξε ότι το πρόβλημά μας μπορεί να θεωρηθεί Δυνητικό Παιχνίδι όταν οι χρήστες φέρουν το ίδιο φορτίο και αυτό ισούται με ένα (1).

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Κίνητρο

Το πρόβλημα εξισορρόπησης φορτίου είναι ένα θέμα που απασχολεί την επιστήμη των Υπολογιστών και εντείνεται με την ανάπτυξη της. Βελτιστοποιώντας συστήματα και δίκτυα, προστίθεται επιπλέον φόρτος εργασίας ο οποίος πρέπει να κατανέμεται εξίσου σε όλους τους εξυπηρετητές για να μπορούν τα οφέλη της επέκτασης ενός δικτύου να είναι εποικοδομητικά.

Ιδιαίτερα δε με την ανάπτυξη του παραλληλισμού η πρόκληση γίνεται ακόμη μεγαλύτερη. Προβλήματα που απαιτούν μεγάλους, πολλούς και περίπλοκους υπολογισμούς με εκατομμύρια δεδομένα σε χώρους όπως η Επιστήμη της Βιολογίας, της Γενετικής, και της Αστροφυσικής βρίσκουν την λύση τους με παραλληλισμό που για να επιτευχθεί χρειάζεται ίση κατανομή φόρτου εργασίας μεταξύ των επεξεργαστών. Έτσι, η αναζήτηση σε μεγάλες βάσεις δεδομένων, ο σχεδιασμός μεγάλων οργανικών μορίων που βρίσκει εφαρμογές στο σχεδιασμό φαρμάκων και στη βιοχημεία και η εξομοίωση μεγάλων συστημάτων, που βρίσκει εφαρμογές στην κίνηση ουράνιων σωμάτων, στη μελέτη ωκεάνιων ρευμάτων στην πρόγνωση καιρού και στην αεροδυναμική, είναι μόνο μερικά από τα προβλήματα που επιλύνονται με

παραλληλισμό και ταχύτερα με σωστή κατανομή φόρτου εργασίας μεταξύ των επεξεργαστών.

1.2 Στόχοι Εργασίας

Στην παρούσα εργασία, επιχειρείται η θεωρητική μελέτη της εξισορρόπησης φορτίου στους κόμβους εξυπηρέτησης, με στόχο δημιουργία Αλγορίθμων μέσω των οποίων να επέρχεται γρηγορότερα η ισορροπία στο σύστημα, να έχουμε δηλαδή όσο το δυνατό ταχύτερο χρόνο σύγκλισης. Για τη δημιουργία Αλγορίθμων, μελετούμε Αλγόριθμους Μετανάστευσης Πληθυσμών [6] και χρησιμοποιούμε στοιχεία της μεταναστευτικής διαδικασίας που ακολουθούν, με στόχο να εξακριβώσουμε εάν αυτοί μπορούν να χρησιμοποιηθούν για Εξισορρόπηση Φόρτου Εργασίας στα δικά μας δίκτυα. Τέλος, θέλουμε να μελετήσουμε πειραματικά τον Αλγόριθμο που επιφέρει τα καλύτερα αποτελέσματα και λαμβάνοντας υπ όψιν την διαδικασία που παρουσιάζεται στο πρόβλημα «Balls and Bins» [7] να βρούμε τις κατάλληλες προδιαγραφές που πρέπει να έχουμε σε κάθε δίκτυο ώστε να έχουμε ταχύτερους χρόνους σύγκλισης.

1.3 Μεθοδολογία και Αποτελέσματα

Στην πρώτη φάση της εργασίας, έγινε θεωρητική μελέτη. Στην ανασκόπηση βιβλιογραφίας μελετήθηκαν αλγόριθμοι που προτάθηκαν μέχρι στιγμής για εξισορρόπηση φόρτου εργασίας σε δίκτυα με διαφορετικά χαρακτηριστικά. Στην ανασκόπηση αυτή εστιάσαμε την προσοχή μας στη μελέτη του αλγορίθμου που χρησιμοποιεί μια ομάδα πληθυσμών πουλιών τα wood thrush για τις μεταναστεύσεις της. [6] Η συμπεριφορά των πουλιών όταν μετανάστευαν σε άλλες περιοχές παρουσίαζε ενδιαφέρον όσον αφορά τον αλγόριθμο που χρησιμοποιούσαν και τις παραμέτρους που λάμβαναν υπ όψιν τους κατά την μετανάστευση. Ιδιαίτερα δε η πιθανότητα μετανάστευσης είχε ιδιαίτερο ενδιαφέρον, αφού παρατηρήσαμε ότι τα πουλιά με οικογένεια μετανάστευαν με πολύ μικρότερη πιθανότητα από τα άλλα ακόμη κι αν η περιοχή που ήταν εγκατεστημένα δεν τους ικανοποιούσε.

Στη δεύτερη φάση της εργασίας, έγινε πειραματική μελέτη. Δημιουργήσαμε το δικό μας αλγόριθμο εξυπηρέτησης φορτίου βασισμένο στον Αλγόριθμο Μετανάστευσης Πληθυσμών [6] και προσαρμοσμένο στις ανάγκες του δικτύου που θέλαμε να μελετήσουμε. Ο αλγόριθμος στη συνέχεια συγκρίθηκε με παραλλαγή ενός

Αλγορίθμου που πρότειναν στο άρθρο τους [2] οι Petra Berenbrink, Tom Friedetzky, Leslie Ann Goldberg, Paul W. Goldberg, Zengjian Hu, Russell A. Martin,, και ήταν πιο κοντά στο μοντέλο που μελετούμε εμείς.

Από τις προσομοιώσεις που έγιναν επιβεβαιώσαμε ότι η δική μας υλοποίηση ήταν καλύτερη για τα συστήματα που θέλαμε να μελετήσουμε, αφού για εξισορρόπηση φόρτου εργασίας χρειαζόταν λιγότερες επαναλήψεις. Προχωρήσαμε με περεταίρω μελέτη του ιδίου του αλγορίθμου σε άλλα δίκτυα με περισσότερες διαφοροποιήσεις σε παραμέτρους τόσο των χρηστών που περίμεναν να εξυπηρετηθούν, όπως φόρτος εργασίας, όσο και των κόμβων εξυπηρέτησης, όπως η χωρητικότητα.

Για την πειραματική μελέτη, δημιουργήσαμε διάφορα σενάρια εστιάζοντας σε παραμέτρους χρηστών και κόμβων που θεωρήσαμε ότι θα παρουσίαζαν μεγαλύτερο ενδιαφέρον στη σύγκριση, όπως πιθανότητα μετανάστευσης ή όχι, τοπολογία δικτύου, φόρτος εργασίας χρηστών και χωρητικότητα κόμβων εξυπηρέτησης. Σε κάθε πειραματική διαδικασία μελετούσαμε μια από τις παραμέτρους, ή συνδυασμό δύο από αυτές και καταγράφοντας τα αποτελέσματα και τις παρατηρήσεις μας. Οι προσομοιώσεις, μας οδήγησαν σε συμπεράσματα για τα στοιχεία που πρέπει να λαμβάνονται υπ όψιν για την σωστή κατανομή φόρτου εργασίας σε διάφορα είδη δικτύων και πως με τις διαφοροποιήσεις τους θα επέρχεται ταχύτερα εξισορρόπηση φορτίου στο σύστημα.

Η τοπολογία αστέρι αποδείχθηκε καλύτερη από την τοπολογία τετράγωνο, ενώ περισσότεροι κόμβοι με μεγαλύτερη χωρητικότητα στο δίκτυο επέφεραν ταχύτερους χρόνους σύγκλισης συγκριτικά με περισσότερους κόμβους μικρότερης χωρητικότητας. Επιπλέον, το σύστημα έφτανε γρηγορότερα σε ισορροπία με μεγαλύτερη πιθανότητα μετανάστευσης των χρηστών, ενώ μια ομάδα κεντρικών κόμβων με διπλάσια πιθανότητα επιλογής ως αρχικούς κόμβους εξυπηρέτησης από τους χρήστες, επέτρεπε ταχύτερο χρόνο σύγκλισης στις περιπτώσεις που είχαμε μικρότερη πιθανότητα μετανάστευσης και η ισοπιθανοτική επιλογή υστερούσε.

Τέλος, θεωρήσαμε το πρόβλημά μας ως Παιχνίδι και προσπαθήσαμε μέσω της Θεωρίας Παιγνίων [8] να αποδείξουμε ότι είναι ένα Δυνητικό Παιχνίδι. Αποδείξαμε ότι είναι ένα Δυνητικό Παιχνίδι με κατάλληλη Συνάρτηση Χρησιμότητας και Δυνητική Συνάρτηση, για την ειδική περίπτωση όπου όλοι οι χρήστες έχουν ίδιο φόρτο εργασίας και αυτός ισούται με ένα (1). Αυτό εισηγείται ότι στη διαδικασία που

ακολουθείται για να επέλθει ισορροπία στο σύστημα, μπορούμε να έχουμε Ισορροπία Nash, κατάσταση κατά την οποία κανένα χρήστη δε συμφέρει να αλλάξει τον τρόπο λήψης απόφασης προς μετανάστευση, όταν ο φόρτος εργασίας των χρηστών ισούται με ένα. Σε αντίθετη περίπτωση, δε μπορούμε να εγγυηθούμε Ισορροπία.

1.4 Δομή Διπλωματικής Εργασίας

Στο Κεφάλαιο 2 που ακολουθεί, γίνεται Εισαγωγή στο Πρόβλημα Εξισορρόπησης Φόρτου Εργασίας, παρουσιάζονται Τεχνικοί Αλγόριθμοι που έχουν προταθεί καθώς επίσης και οι Θεωρητικές Προσεγγίσεις που μελετήθηκαν [1,2,5]. Εκτενής αναφορά γίνεται για τον Αλγόριθμο Μετανάστευσης Πληθυσμών [6] και ακολουθεί σύγκριση ανάμεσα σε αυτών και τους υπόλοιπους που παρουσιάστηκαν.

Ακολούθως, στο Κεφάλαιο 3 παρουσιάζονται οι Αλγόριθμοι Εξισορρόπησης Φόρτου Εργασίας που προτάθηκαν και υλοποιήθηκαν [3,4] οι οποίοι επίσης συγκρίνονται με τον Αλγόριθμο Μετανάστευσης Πληθυσμών [6].

Στο Κεφάλαιο 4 παρουσιάζεται το πρόβλημα της Εξισορρόπησης Φόρτου Εργασίας και οι Αλγόριθμοι που δημιουργήσαμε για την μελέτη του. Ο πρώτος, που ονομάζουμε Αλγόριθμο 1 αποτελεί παραλλαγή ενός προτεινόμενου όπως παρουσιάζεται στο άρθρο [2], ενώ ο δεύτερος που ονομάζουμε Αλγόριθμο 2, είναι εμπνευσμένος από τον Αλγόριθμο Μετανάστευσης Πληθυσμών [6].

Στο Κεφάλαιο 5 παρουσιάζεται η Σύγκριση Αλγορίθμων 1 και 2, εστιάζοντας στους χρόνους σύγκλισης έως ότου επέλθει ισορροπία στο δίκτυο και στο ίδιο Κεφάλαιο γίνεται και η Πειραματική Αξιολόγηση του Αλγόριθμου 2 που είναι δική μας υλοποίηση.

Στο Κεφάλαιο 6 παρουσιάζονται σε συντομία τα βασικά χαρακτηριστικά της Θεωρίας Παιγνίων [8,9] και στη συνέχεια αποδεικνύεται ότι το πρόβλημά μας μπορεί να εκφραστεί ως ένα Δυνητικό Παιχνίδι.

Τέλος στο Κεφάλαιο 7 παραθέτουμε τα Συμπεράσματα τόσο από τη θεωρητική όσο και από την Πειραματική Μελέτη του προβλήματος του Φόρτου Εργασίας κλείνοντας με μελλοντική έρευνα που προτείνουμε να γίνει γύρω από το Θέμα.

Κεφάλαιο 2

Εξισορρόπηση Φόρτου Εργασίας και Θεωρητικές Προσεγγίσεις

2.1 Εισαγωγή στο Πρόβλημα Εξισορρόπησης Φόρτου Εργασίας

Τα τελευταία χρόνια, κυρίως τις τελευταίες δύο δεκαετίες, παρατηρείται ραγδαία αύξηση των χρηστών των πληροφοριακών συστημάτων οι οποίοι συγχρόνως αυξάνουν και τις απαιτήσεις τους σε υπολογιστική ισχύ. Ως εκ τούτου, έχουμε κατακόρυφη εξόρυξη δεδομένων και πληροφοριών στα συστήματα που διαχειρίζονται οι χρήστες και κατ επέκταση το πρόβλημα της ορθής διαχείρισης του τεράστιου αυτού όγκου ψηφιακής πληροφορίας .

Μετά την εξάντληση ης αναβάθμισης της ταχύτητας των επεξεργαστών ώστε να εκτελούν τις διεργασίες ταχύτερα, στο χώρο εισήχθηκε ένας νέος όρος, η παράλληλη επεξεργασία. Παρόλο που η παράλληλη επεξεργασία δεν ικανοποίησε τις προσδοκίες των κατασκευαστών σε επίδοση και υπερέβη κατά πολύ το προβλεπόμενο κόστος, εντούτοις η μελέτη της συνεχίστηκε, λόγω της μεγάλης

ανάγκης επεξεργασίας τεράστιου όγκου δεδομένων πολλαπλών χρηστών ταυτόχρονα, η οποία αυξάνεται με γεωμετρική πρόοδο.

Στόχος της παράλληλης επεξεργασίας είναι η μέγιστη απόδοση του συστήματος, που σημαίνει βέλτιστος χρόνος απόκρισης με το ελάχιστο κόστος, πράγμα που απαιτεί κατάλληλη εξισορρόπηση του φόρτου εργασίας μεταξύ των πόρων του συστήματος.

Η εξισορρόπηση του φόρτου εργασίας αποτελεί και το μεγαλύτερο πρόβλημα σε κατανεμημένα περιβάλλοντα όπως τα δίκτυα που θέλουμε να δημιουργήσουμε. Το πρόβλημα της εξισορρόπησης φόρτου εργασίας αφορά την προσπάθεια του δικτύου να φτάσει σε κατάσταση κατά την οποία να ισχύει για κάθε κόμβο ότι ο συνολικός φόρτος εργασίας των χρηστών που εξυπηρετεί δε ξεπερνά το μέγιστο που μπορεί να αντέξει, τη χωρητικότητά του.

Στην παρούσα διπλωματική εργασία θα ασχοληθούμε με τη μελέτη τεχνικών – αλγόριθμων εξισορρόπησης φορτίου και θα επικεντρωθούμε στην εξεύρεση κατάλληλου αλγόριθμου που θα εξισορροπεί το φόρτο εργασίας στην κάθε εφαρμογή.

2.2 Μελέτη Τεχνικών και Αλγορίθμων

Για να καταστεί ένας αλγόριθμος εξισορρόπησης φορτίου αποδοτικός προϋποθέτει να έχει δύο βασικά χαρακτηριστικά, επιγραμματική λειτουργικότητα και προσαρμοστικότητα. Το πρώτο εγγυάται ανάθεση του αιτήματος που καταφθάνει στην καταλληλότερη μηχανή, αυτή που μπορεί να εξυπηρετήσει το αίτημα κρατώντας συγχρόνως το φόρτο του συστήματος σε μια ισορροπία, χωρίς να έχει λεπτομερή επίγνωση του φόρτου εργασίας ολόκληρου του δικτύου, ενώ το δεύτερο γρήγορη προσαρμογή στις συνεχόμενες αλλαγές που επέρχονται στο σύστημα από τη διαφοροποίηση του φόρτου εργασίας σε κάθε χρονική στιγμή, λόγω της αλλαγής του πλήθους των αιτημάτων που καταφθάνουν κάθε φορά.

Οι κατηγορίες που μπορούμε να κατατάξουμε τους αλγόριθμους αναλόγως της πληροφορίας που χρησιμοποιούν και τους μηχανισμούς συμπεριφοράς τους είναι οι Στατικοί Αλγόριθμοι, Static Algorithms, οι Δυναμικοί Αλγόριθμοι, Dynamic Algorithms και οι Προσαρμοστικοί Αλγόριθμοι,

2.2.1 Στατικοί Αλγόριθμοι

Οι Στατικοί Αλγόριθμοι, συμπεριφέρονται πάντοτε με τον ίδιο τρόπο, χρησιμοποιούν την ίδια λογική κατά την απόφαση ανάθεσης αιτήματος προς εξυπηρέτηση σε μία μηχανή. Επιπλέον θεωρούν ότι γνωρίζουν όλες τις πληροφορίες για τις εργασίες, τους κόμβους, το επικοινωνιακό δίκτυο. Τέτοιοι αλγόριθμοι είναι οι Τυχαίας Ανάθεσης, Random, Κυκλικής Ανάθεσης, Round Robin, και Κυκλικής Ανάθεσης με Βάρη, Weighted Round Robin ή Radio .

Ο Αλγόριθμος Τυχαίας Ανάθεσης, επιλέγει τυχαία σε ποια από τις μηχανές που διαθέτει θα αναθέσει το αίτημα που καταφθάνει χωρίς να λαμβάνει υπ όψιν του οποιεσδήποτε άλλες παραμέτρους. Αυτή η μέθοδος αν και δεν είναι ιδιαίτερα αποδοτική, έχει πολύ μικρό χρόνο απόκρισης. Χρησιμοποιείται σε συστήματα που η εξισορρόπηση φορτίου δεν τα απασχολεί.

Ο Αλγόριθμος Κυκλικής Ανάθεσης, επιλέγει να αναθέσει το αίτημα που καταφθάνει στην τρέχουσα διαθέσιμη μηχανή. Οι μηχανές μπαίνουν σε τυχαία σειρά και το αίτημα ανατίθεται στη μηχανή που βρίσκεται στην κεφαλή της σειράς και αυτή μεταφέρεται στην ουρά. Ο Αλγόριθμος Κυκλικής Ανάθεσης λειτουργεί αρκετά αποδοτικά στα περισσότερα συστήματα αλλά παρουσιάζονται προβλήματα σε περίπτωση που οι μηχανές του συστήματος δεν έχουν τις ίδιες επιδόσεις σε επεξεργασία, συνδέσεις και μνήμη, διότι σε μία τέτοια περίπτωση ένα αίτημα με μεγαλύτερες απαιτήσεις θα ήταν καλύτερο να ανατεθεί σε μία μηχανή με καλύτερες επιδόσεις παρά στην «τυχαία» που περιμένει στη σειρά.

Ο Αλγόριθμος Κυκλικής Ανάθεσης με Βάρη λειτουργεί όπως ο Αλγόριθμος Κυκλικής Ανάθεσης με τη διαφορά ότι κάθε μηχανή έχει μία παράμετρο που αντιπροσωπεύει το βάρος της και κατ επέκταση και την επίδοσή της. Σε μηχανές με μεγαλύτερο βάρος ο αλγόριθμος αναθέτει περισσότερες αιτήσεις προς εξυπηρέτηση. Ο Αλγόριθμος Κυκλικής Ανάθεσης με Βάρη, λύνει το πρόβλημα της διαφορετικής επίδοσης κάθε μηχανής του συστήματος. Παρ όλα αυτά τα βάρη που ανατίθενται παραμένουν τα ίδια καθ όλη τη διάρκεια ζωής του συστήματος χωρίς να γίνεται έλεγχος εάν εξακολουθούν να ισχύουν. Εάν δηλαδή οι μηχανές εξακολουθούν να έχουν την ίδια επίδοση.

2.2.2 Δυναμικοί Αλγόριθμοι

Οι Δυναμικοί Αλγόριθμοι, συμπεριφέρονται αναλόγως της κατάστασης του συστήματος. Λαμβάνουν, για παράδειγμα, υπ όψιν τους το πλήθος των αιτημάτων που καταφθάνουν και το φόρτο εργασίας κάθε μηχανής την τρέχουσα χρονική στιγμή στην απόφασή για ανάθεση του αιτήματος στην κατάλληλη μηχανή.

Το κυριότερο μειονέκτημά τους είναι η επιβάρυνση από την ανάλυση της κατάστασης για την απόφαση της καταλληλότερης μηχανής. Επιπλέον σχετικά με την ανάθεση φόρτου εργασίας χρησιμοποιούν την τρέχουσα πληροφορία για τις αποφάσεις τους. Παραδείγματα τέτοιων αλγόριθμων αποτελούν οι Δυναμικός Κυκλικής Ανάθεσης, Dynamic Round Robin ή Dynamic Radio, ο Αλγόριθμος Γρηγορότερης Ανάθεσης, Fastest, ο Αλγόριθμος Λιγότερων Διασυνδέσεων Least Connections, ο Αλγόριθμος Παρατήρησης, Observed, και ο Αλγόριθμος Πρόβλεψης, Predictive.

2.2.2.1 Δυναμικός Αλγόριθμος Κυκλικής Ανάθεσης

Ο Δυναμικός Αλγόριθμος Κυκλικής Ανάθεσης λειτουργεί όπως και ο στατικός Κυκλικής Ανάθεσης με Βάρη, με τη διαφορά ότι τα βάρη περιοδικά διαφοροποιούνται αναλόγως αποδοτικότητας της μηχανής, η οποία μπορεί να μεταφράζεται ως αριθμός αιτημάτων που εξυπηρετεί τη δεδομένη χρονική στιγμή, ως χρόνος απόκρισης, κτλ. Οι μηχανές δηλαδή επανεξετάζονται για να αποφασιστεί πόσο είναι το βάρος τους πριν ξαναμπούν στην ουρά. Ο Δυναμικός Αλγόριθμος Κυκλικής Ανάθεσης έρχεται να λύσει το πρόβλημα της στατικότητας του Αλγόριθμου Κυκλικής Ανάθεσης αλλά προσθέτει το κόστος του συνεχές υπολογισμού των βαρών των μηχανών που στις πλείστες περιπτώσεις είναι μεγάλο και αυξάνεται εκθετικά με την προσθήκη έξτρα μηχανών.

2.2.2.2 Αλγόριθμος Γρηγορότερης Ανάθεσης

Ο αλγόριθμος Γρηγορότερης Ανάθεσης επιλέγει να αναθέσει το αίτημα που καταφθάνει στη μηχανή με τον καλύτερο χρόνο απόκρισης, response time. Ο αλγόριθμος είναι κατάλληλος σε περιπτώσεις όπου οι διακομιστές, servers του συστήματος είναι διάσπαρτοι σε πολλά σημεία απομακρυσμένα μεταξύ τους αλλά μπορεί να δημιουργήσει προβλήματα συμφόρησης καθώς ο χρόνος απόκρισης της

κάθε μηχανής μεταβάλλεται συνεχώς και σε πολύ τακτά χρονικά διαστήματα. Αυτό σημαίνει ότι η μηχανή με το μικρότερο χρόνο απόκρισης που επιλέξαμε να εξυπηρετήσει το αίτημα που μόλις έχει καταφθάσει, δε σημαίνει ότι θα εξακολουθεί να διατηρεί τον ίδιο χαμηλό χρόνο απόκρισης καθ όλη τη διάρκεια που την εξυπηρετεί. Να εξακολουθεί δηλαδή να είναι η καταλληλότερη για την επεξεργασία του αιτήματος.

2.2.2.3 Αλγόριθμος Λιγότερων Διασυνδέσεων

Ο αλγόριθμος Λιγότερων Διασυνδέσεων αναθέτει το αίτημα που καταφθάνει στη μηχανή που εξυπηρετεί τα λιγότερα αιτήματα τη δεδομένη χρονική στιγμή. Είναι κατάλληλος σε περιπτώσεις όπου όλες οι μηχανές του συστήματος έχουν τις ίδιες δυνατότητες αλλά κρίνεται ακατάλληλος στις περιπτώσεις όπου τα αιτήματα που καταφθάνουν έχουν διαφορετικό χρόνο επεξεργασίας έως ότου πάρουμε το τελικό αποτέλεσμα.

2.2.2.4 Αλγόριθμος Παρατήρησης

Ο αλγόριθμος Παρατήρησης αποτελεί συνδυασμό των αλγορίθμων Γρηγορότερης Ανάθεσης και Λιγότερων Διασυνδέσεων Καταλληλότερη μηχανή προς εξυπηρέτηση του αιτήματος που καταφθάνει είναι η μηχανή με τα λιγότερα τρέχοντα αιτήματα και το μικρότερο χρόνο απόκρισης. Ο αλγόριθμος ενώ δημιουργήθηκε για να συνδυάσει τα προτερήματα και των δύο αλγόριθμων πολλές φορές ενεργεί με χειρότερα και από τους δύο, συνδυάζοντας τα αρνητικά τους. Στις αδυναμίες του αλγόριθμου μπορεί να προστεθεί και ο χρόνος που χρειάζεται για να υπολογίσει την καταλληλότερη μηχανή που προφανώς είναι το άθροισμα του χρόνου που χρειάζονται οι δύο αλγόριθμοι που συνδυάζει.

2.2.2.5 Αλγόριθμος Πρόβλεψης

Τέλος ο Αλγόριθμος Πρόβλεψης είναι μία παραλλαγή του Αλγόριθμου Παρατήρησης, καθώς ενεργεί με τον ίδιο τρόπο με τη διαφορά ότι οι μηχανές σταδιακά επανελέγχονται εάν όντως εξακολουθούν να παρουσιάζουν τα ίδια επίπεδα αποδοτικότητας και επανανακατατάσσονται. Ο Αλγόριθμος Πρόβλεψης φαίνεται να είναι ελαφρώς αποδοτικότερος από τον Αλγόριθμος Παρατήρησης.

Στο σημείο αυτό αξίζει να αναφερθεί ότι οι στατικοί αλγόριθμοι έχουν ευκολότερη εφαρμογή και μικρή επιβάρυνση χρόνου εκτέλεσης, runtime overhead, ενώ οι δυναμικοί χαρακτηρίζονται από καλύτερη επίδοση.

2.2.3 Προσαρμοστικοί Αλγόριθμοι

Στην κατηγορία των προσαρμοστικών αλγορίθμων κατατάσσονται οι αλγόριθμοι που προσαρμόζουν τη συμπεριφορά και τη δράση τους αναλόγως της κατάστασης του συστήματος. Σε κάθε περιοχή του δικτύου εφαρμόζεται ο αλγόριθμος που είναι πιο αποδοτικός. Ένας προσαρμοστικός αλγόριθμος μπορεί να αποτελείται για παράδειγμα από δύο δυναμικούς που ο καθένας ενεργοποιείται οποτεδήποτε κρίνεται πιο αποτελεσματικός. Συνδυασμοί δυναμικών αλγορίθμων σε προσαρμοστικούς είναι οι Αλγόριθμοι Αρχικοποίησης Παραλήπτη , Receiver Initiated, οι Αλγόριθμοι Αρχικοποίησης Αποστολέα, Sender Initiated, και Αλγόριθμοι Συμμετρικής Αρχικοποίησης, Symmetrical Initiated Algorithms.

2.2.3.1 Αλγόριθμοι Αρχικοποίησης Αποστολέα

Οι Αλγόριθμοι Αρχικοποίησης Αποστολέα ενεργοποιούνται όταν στο δίκτυο παρουσιαστεί μια υπερφορτωμένη μηχανή που προσπαθεί να αποστείλει πακέτο σε κάποια λιγότερο ενεργή. Σε ένα τέτοιο σενάριο η υπερφορτωμένη μηχανή είναι ο αποστολέας ενώ η λιγότερο βεβαρημένη ο παραλήπτης. Σε γενικές γραμμές τα χαρακτηριστικά των Αλγορίθμων Αρχικοποίησης Αποστολέα είναι τα εξής:

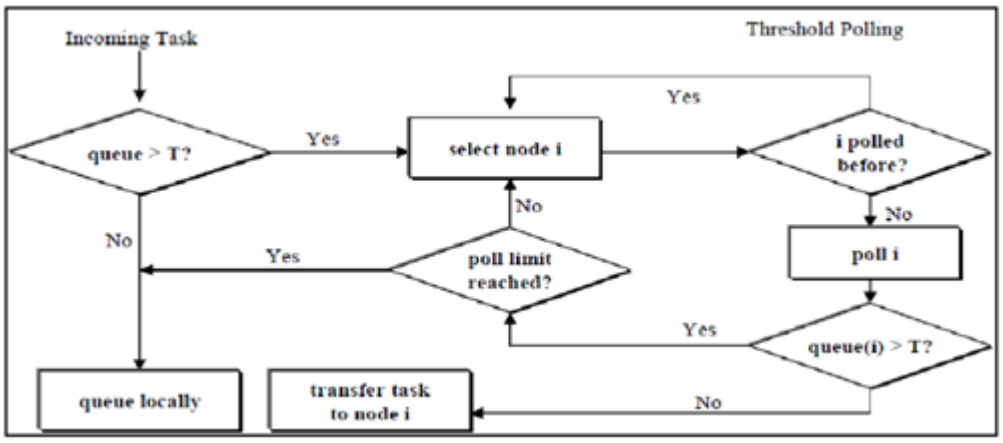
1. Μηχανή στην ουρά της οποίας υπάρχουν περισσότερα από ένα όριο αιτημάτων Τ χαρακτηρίζεται ως αποστολέας ενώ σε αντίθετη περίπτωση, μηχανή δηλαδή με λιγότερα από Τ αιτήματα στην ουρά της, χαρακτηρίζεται ως παραλήπτης.
2. Η πολιτική επιλογής αιτήματος που θα εξυπηρετηθεί βασίζεται στο χρόνο που αυτό καταφθάνει. Προτεραιότητα έχουν πάντα τα πιο πρόσφατα αιτήματα.
3. Οι Πολιτικές Ανάθεσης του αιτήματος που καταφθάνει σε κατάλληλη μηχανή προς εξυπηρέτηση διαφέρει σε κάθε Αλγόριθμο Αρχικοποίησης Αποστολέα. Συγκεκριμένα υπάρχουν τρεις τέτοιες πολιτικές. Η Πολιτική Τυχαίας Τοποθεσίας, Random Location Policy, η Πολιτική Τοποθεσίας Κατωφλίου, Threshold Location Policy, και η πολιτική συντομότερης τοποθεσίας, Shortest Location Policy.

Στην Πολιτική Τυχαίας τοποθεσίας το αίτημα ανατίθεται σε τυχαίο παραλήπτη. Στην Πολιτική Τοποθεσίας Κατωφλίου το αίτημα αποστέλλεται με πληροφορίες σε πιθανό παραλήπτη. Τέλος, στην Πολιτική Συντομότερης Τοποθεσίας το αίτημα αποστέλλεται με πληροφορίες σε Κ πιθανούς παραλήπτες και το εξυπηρετεί ο λιγότερο βαρυφορτωμένος.

Εδώ αξίζει να σημειωθεί ότι η αποδοτικότητα της Πολιτικής Συντομότερης Τοποθεσίας , δεν είναι κατά πολύ μεγαλύτερη από αυτή της Τοποθεσίας Κατωφλίου.

Παρατηρούμε ότι στους Αλγόριθμους Αρχικοποίησης Αποστολέα η ενέργεια επικεντρώνεται στον αποστολέα, ο οποίος προσπαθεί να βρει κατάλληλο παραλήπτη για τα αιτήματα που ο ίδιος αδυνατεί να διεκπεραιώσει.

Η Αλγόριθμοι Αρχικοποίησης Αποστολέα δεν ενδείκνυνται σε περιπτώσεις συστημάτων με μεγάλο φόρτο εργασίας, καθώς η εξεύρεση παραλήπτη, μονάδα με χαμηλό φορτίο, θα είναι δύσκολη υπόθεση.



Σχήμα 2.1 Αλγόριθμος Αρχικοποίησης Αποστολέα με εφαρμογή της Πολιτικής Κατωφλίου

2.2.3.2 Αλγόριθμοι Αρχικοποίησης Παραλήπτη

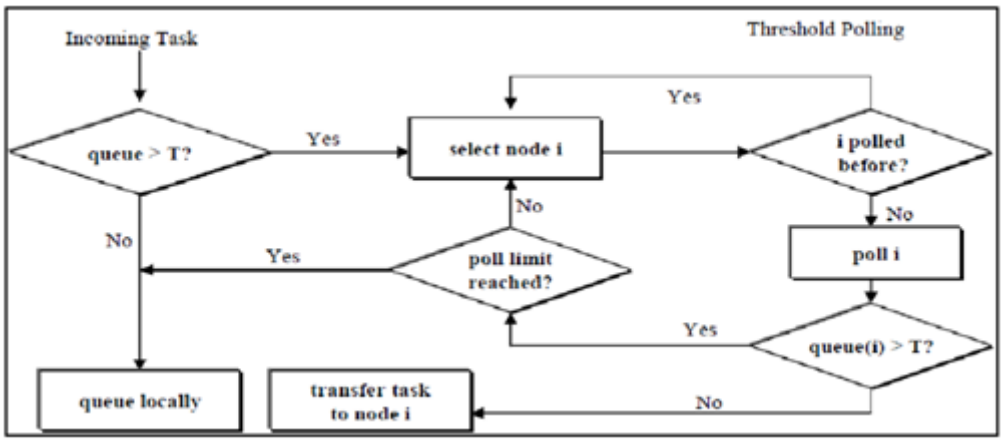
Οι Αλγόριθμοι Αρχικοποίησης Παραλήπτη ενεργοποιούνται όταν στο δίκτυο παρουσιαστεί μια μηχανή με χαμηλό φορτίο που προσπαθεί να λάβει πακέτο από κάποια με υψηλότερο. Και πάλι σε ένα τέτοιο σενάριο η υπερφορτωμένη μηχανή

είναι ο αποστολέας ενώ η λιγότερο βεβαρημένη ο παραλήπτης. Ο διαχωρισμός παραλήπτη- αποστολέα γίνεται και πάλι με γνώμονα ένα όριο Τ που υποδηλώνει το μέγεθος της ουράς, δηλαδή τον αριθμό των αιτημάτων που εξυπηρετεί η μηχανή.

Αντίθετα με τους Αλγόριθμους Αρχικοποίησης Αποστολέα, στους Αλγόριθμους Αρχικοποίησης Παραλήπτη η ενέργεια επικεντρώνεται στον παραλήπτη που ψάχνει κατάλληλο αποστολέα, ο οποίος με την παραλαβή του αιτήματος θα καταστεί κι αυτός παραλήπτης.

Η διαδικασία που ακολουθεί ο αλγόριθμος είναι απλή. Μετά την ολοκλήρωση του αιτήματος η μηχανή ελέγχει το μέγεθος της ουράς. Εάν αυτό είναι μικρότερο από ένα όριο Τ, τότε η μηχανή χαρακτηρίζεται και δρα πλέον ως παραλήπτης και αναλαμβάνει να βρει κατάλληλο αποστολέα, αρχικά επιλέγοντας τυχαία. Κατάλληλος αποστολέας κρίνεται η μηχανή η οποία με την αποστολή του αιτήματος στον συγκεκριμένο παραλήπτη θα γίνει κι αυτή παραλήπτης. Αυτό συμβαίνει όταν με την αποστολή του αιτήματος στον παραλήπτη το μέγεθος της ουράς αναμονής των αιτημάτων της γίνεται μικρότερο από το όριο Τ. Εάν με την αποστολή πακέτου ο αποστολέας δε «μετατρέπεται» σε παραλήπτη το αίτημα αποστέλλεται πίσω και ο παραλήπτης αναζητά άλλο αποστολέα. Η διαδικασία συνεχίζεται έως ότου βρεθεί κατάλληλος αποστολέας ή οι προσπάθειες εντοπισμού αποστολέα από τον παραλήπτη ξεπεράσουν ένα όριο προσπαθειών, poll limit.

Στο σημείο αυτό πρέπει να αναφερθεί ότι οι Αλγόριθμοι Αρχικοποίησης Παραλήπτη είναι αποδοτικοί σε περιπτώσεις συστημάτων με μεγάλο φόρτο εργασίας, καθώς η εξεύρεση αποστολέα, μονάδα με υψηλό φορτίο, γίνεται με μικρό αριθμό αναζητήσεων.



Σχήμα 2.2 Αλγόριθμος Αρχικοποίησης Παραλήπτη

2.2.3.3 Αλγόριθμοι Συμμετρικής Αρχικοποίησης

Οι Αλγόριθμοι Συμμετρικής Αρχικοποίησης, Symmetrical Initiated Algorithms, αποτελούν συνδυασμό των Αλγορίθμων Αρχικοποίησης Αποστολέα και Παραλήπτη. Η ευθύνη των ενεργειών για εξισορρόπηση του φόρτου του συστήματος δεν υπόκεινται εξ ολοκλήρου στον αποστολέα ή τον παραλήπτη αλλά και στους δύο από κοινού. Αποτέλεσμα της συνεργασίας των δύο κατηγοριών αλγορίθμων είναι η εμφάνιση των πλεονεκτημάτων και των μειονεκτημάτων και των δύο.

2.3 Κεντρικοποιημένοι και Κατανεμημένοι Αλγόριθμοι

Μια άλλη κατηγοριοποίηση αλγορίθμων εξισορρόπησης φορτίου είναι ο διαχωρισμός μεταξύ Κεντρικοποιημένων, Centralized, και Κατανεμημένων Distributed, Αλγορίθμων.

Στους Κεντρικοποιημένους αλγόριθμους υπάρχει ένας κόμβος ο οποίος δρομολογεί και αποφασίζει για την ανάθεση φόρτου εργασίας στους υπόλοιπους κόμβους οι οποίοι κάθε φορά του αποστέλλουν τις πληροφορίες του δικτύου που συγκεντρώνουν.

Αντιθέτως, στους Κατανεμημένους αλγόριθμους δεν υπάρχει κεντρικός κόμβος αλλά όλοι οι κόμβοι λαμβάνουν μέρος στην κατανομή του φορτίου και καθένας από αυτούς επωμίζεται το κόστος της αποθήκευσης και ανανέωσης των πληροφοριών.
Στην παρούσα Διπλωματική Εργασία, θα ασχοληθούμε με κατανεμημένους αλγόριθμους.

Πιο κάτω παρουσιάζεται αναλυτικά η μελέτη άρθρων που παρουσιάζουν θεωρητικές προσεγγίσεις αλγορίθμων που προτείνονται γα Εξισορρόπηση Φόρτου Εργασίας.

2.4 Distributed Algorithms for QoS Load Balancing [1]

2.4.1 Περιγραφή Έρευνας

Σε μεγάλα Κατανεμημένα Συστήματα, όπως ασύρματα δίκτυα, Wireless Networks, είναι δύσκολο να πετύχουμε Εξισορρόπηση Φόρτου Εργασίας, Load Balancing, με τον κλασσικό τρόπο του αλγορίθμου που να μαζεύει πληροφορίες του δικτύου και να

βρίσκει τη λύση της κατανομής του φόρτου. Σε μεγάλα συστήματα είναι πιο προσοδοφόρο να έχουμε Κατανεμημένα Πρωτόκολλα, Distributed Protocols , μέσω των οποίων θα ανακτούμε πληροφορίες σχετικές με το φόρτο του δικτύου και βάση αυτών θα προχωρούμε στην κατανομή του φόρτου από μια κεντρική οντότητα προς ανεξάρτητους χρήστες.

Σε Κατανεμημένα Συστήματα η απόφαση του χρήστη για το που θα αναθέσει την εργασία του προς εξυπηρέτηση είναι ανεξάρτητη. Οι χρήστες διαθέτουν μόνο μια γενική πληροφορία για τον φόρτο και την καθυστέρηση στο δίκτυο που χρησιμοποιούν για να πάρουν αποφάσεις. Επιπλέον δε χρειάζεται να γίνεται γενικός συντονισμός κάθε φορά που καταφθάνει διεργασία προς εξυπηρέτηση

Στη παρούσα έρευνα μελετείται το Πρόβλημα Κατανομής Πόρων με Ποιότητα Υπηρεσίας (Resource Allocate Problems with QoS). Υπάρχουν ποικίλα είδη πρωτοκόλλων για ρυθμίσεις κατανομής φόρτου στα οποία οι χρήστες δεν έχουν γενική γνώση του δικτύου αλλά μόνο τοπική. Σε γενικές γραμμές τέτοια πρωτόκολλα βοηθούν τους χρήστες να βελτιώσουν την επίδοσή τους προβαίνοντας στις κατάλληλες επιλογές του πόρου που θα τους εξυπηρετήσει. Για παράδειγμα μία προσέγγιση είναι όταν ο χρήστης καταλήξει σε ένα πόρο, να μην επαναπαύεται με την επιλογή του, αλλά να δοκιμάζει και άλλους γειτονικούς με κάποια πιθανότητα μετανάστευσης, εάν εξακριβώσει ότι κάποιος από τους γείτονες μπορεί να τον εξυπηρετήσει καλύτερα. Υπάρχει ακόμη και η προσέγγιση της γνώσης μέρους της γενικής πληροφορίας του δικτύου από τους χρήστες, όπως για παράδειγμα να γνωρίζουν τους πόρους που είναι υποφορτωμένοι, underloaded και επομένως μπορούν να τους εξυπηρετήσουν χωρίς πρόβλημα.

Στο συγκεκριμένο πρωτόκολλο ο χρήστης επιλέγει τους πόρους που θα τον εξυπηρετήσουν αφού μελετήσει την επίδοσή τους. Εν τέλει επιλέγει αυτόν με κάποιο όριο εξυπηρέτησης, threshold, που να τον ικανοποιεί.

2.4.2 Προηγούμενα Μοντέλα που Προτάθηκαν για Έλεγχο της Συμφόρησης μη Συντονισμένων Εγωιστών Χρηστών

Στο σημείο αυτό παρεμβάλλονται και επεξηγούνται δύο πρωτόκολλα που προτάθηκαν σχετικά με συμφόρηση μη συντονισμένων εγωιστών χρηστών.

Ο Goldberg σε πρωτόκολλο του εισηγείται οι χρήστες να μπορούν να μεταναστεύσουν τυχαία σε άλλο κόμβο εάν αυτός ελαχιστοποιεί το λάθος του, την καθυστέρηση του. Αρχικά κάθε χρήστης ανατίθεται τυχαία σε κάποιο κόμβο και περιοδικά εξετάζει για καλύτερο κόμβο εξυπηρετητή.

Οι Even-Dar και Monsour εισηγούνται πρωτόκολλο στο οποίο οι χρήστες να έχουν πλήρη γνώση της κατάστασης του δικτύου. Να μπορούν να αναγνωρίζουν τις φορτωμένες από τις μη φορτωμένες γραμμές. Στο συγκεκριμένο πρωτόκολλο, κάθε χρήστης επιλέγει άλλο χρήστη και δοκιμάζει την δική του τεχνική απόφασης κόμβου εξυπηρέτησης. Την υιοθετεί εάν βελτιώνει την καθυστέρησή του, το λάθος του. Ο αλγόριθμος αποδίδει γρήγορη σύγκλιση στις σχεδόν ισορροπημένες καταστάσεις στηριζόμενος στη γνώση των πόρων των στρατηγικών-δειγμάτων.

2.4.3 Περιγραφή Μοντέλου Κατανομής Φόρτου Εργασίας

Στο νέο μοντέλο που προτείνεται οι χρήστες του συστήματος δε μεταναστεύουν αμέσως μόλις διαπιστώσουν ότι ο κόμβος που εξετάζουν είναι καταλληλότερος από τον τρέχον που τους εξυπηρετεί, όπως στο πρωτόκολλο του Goldberg, αλλά με κάποια πιθανότητα. Επιπλέον οι χρήστες δεν έχουν ολική γνώση του δικτύου, όπως προτείνουν στο πρωτόκολλό τους οι Even-Dar και Monsour, αλλά μερική.

Συγκεκριμένα, στο συγκεκριμένο μοντέλο, κάθε χρήστης επιχειρεί να δεσμεύσει τον πόρο που μπορεί να τον εξυπηρετήσει με μέγιστη επίδοση. Η επίδοση του πόρου είναι ανάλογη των χρηστών που τον μοιράζονται και είναι ανεκτή για ένα χρήστη εάν υπερβαίνει το κατώτατο όριό του. Σε αντίθετη περίπτωση προβαίνει σε αλλαγή πόρου εξυπηρέτησης. Οι χρήστες δεν έχουν γενική ιδέα του συστήματος αλλά χρησιμοποιούν μόνο τη διαθέσιμη τοπική πληροφορία για τις επιλογές τους.

Κάθε μηχανή, recourse, έχει διαφορετική συνάρτηση λάθους, δηλαδή καθυστέρηση που ανέχεται, και κάθε χρήστης, user, διαφορετικό όριο λάθους που ανέχεται, το οποίο διαφοροποιείται αναλόγως της μηχανής. Διαφορετικές μηχανές έχουν διαφορετική ποιότητα υπηρεσίας.

Επιπλέον, θεωρείται ότι το σύστημα είναι ανεξάρτητο-χρήστη, user-independent, εάν για κάθε μηχανή ν υπάρχει όριο τέτοιο Tr ώστε να υπάρχει όριο λάθους που να ανέχεται ένας χρήστης i. Σε αντίθετη περίπτωση το ονομάζουμε συγκεκριμένου-χρήστη , user-specific.

Το σύστημα είναι ανεξάρτητο-πόρου, resource-independent, εάν για κάθε χρήστη i υπάρχει όριο λάθους T τέτοιο ώστε να υπάρχει μηχανή r που να ικανοποιεί αυτό το όριο. Σε αντίθετη περίπτωση το ονομάζουμε ανεξάρτητο-ομοιομορφίας , uniform.

2.4.4 Μελέτη Αλγορίθμων Καταστάσεων Δικτύου

2.4.4.1 Περίπτωση Ανεξάρτητων Χρηστών

Σε περιπτώσεις ανεξαρτησίας των χρηστών, μελετούμε την περίπτωση όπου όλοι οι χρήστες έχουν το ίδιο όριο T.

Ο αλγόριθμος λειτουργεί με απλό τρόπο. Για όλους τους χρήστες που ζητούν να εξυπηρετηθούν παράλληλα αναθέτουμε μια πηγή που θα αναλάβει να εξυπηρετήσει τον καθένα από αυτούς. Εάν ο αριθμός χρηστών που εξυπηρετεί η μηχανή στην οποία αναθέσαμε ένα χρήστη υπερβαίνει το όριο που ανέχεται ο χρήστης, τότε ο αλγόριθμος επιλέγει ομοιόμορφα και τυχαία κάποια μηχανή για να μεταναστεύσει.

Algorithm 1 Threshold-Balancing Protocol

for all users i in parallel do
  Let $r(i)$ be the resource currently allocated by user i.
  if $x_{r(i)} > T_{r(i)}$ then
    With probability $\alpha \cdot \frac{x_{r(i)} - T_{r(i)}}{x_{r(i)}}$ migrate to a resource chosen uniformly at random.
  end if
end for

Σχήμα 2.3 Αλγόριθμος Ανεξάρτητων χρηστών [1]

2.4.4.2 Περίπτωση Εξειδικευμένων Χρηστών

Για να επιτευχθεί ταχεία σύγκλιση, ελαχιστοποίηση δηλαδή του χρόνου που χρειαζόμαστε για να επέλθει το δίκτυο σε ισορροπία, πρέπει να είναι γνωστό το T των χρηστών που εξυπηρετούνται από τον ίδιο πόρο. Στη συνέχεια, η πιθανότητα μετανάστευσης των ανικανοποίητων χρηστών αυξάνεται μόνο όταν εξασφαλιστεί ότι οι υπόλοιποι μένουν ικανοποιημένοι.

Αρχικά οι χρήστες κάθε πόρου ταξινομούνται σε φθίνουσα σειρά του T . Στη συνέχεια επιλέγονται αυτοί των οποίων το όριο λάθους του πόρου ξεπερνά το δικό τους όριο λάθους. Τέλος, οι χρήστες που εκδιώχθηκαν ομοιόμορφα και τυχαία σε τοποθετούνται σε άλλο πόρο.

Algorithm 2 Kick-The-Complainer

for all resources r in parallel do
  Sort users on resource r in decreasing order of T_r^i.
  Let $U_r = \{i : T_r^i > i\}$ denote the set of excess users.
  Remove U_r from r and reassign them uniformly at random.
end for

Σχήμα 2.4 Αλγόριθμος Εξειδικευμένων χρηστών [1]

2.4.4.3 Περίπτωση Αγνώστου Αριθμού Χρηστών

Στους παραπάνω αλγόριθμους, θεωρείται ότι το σύστημα γνωρίζει επακριβώς τον αριθμό των χρηστών. Σε περίπτωση που ο αριθμός των χρηστών είναι άγνωστος για το σύστημα, στόχος μας είναι να αποδεχθεί ένα όριο T στους χρήστες για να επιτευχτεί γρήγορα σύγκλιση. Για αυτό, χρησιμοποιείται ο παρακάτω αλγόριθμος:

Algorithm 3 Exponential search

1: for all users i in parallel do
2: set $\tilde{n} \leftarrow 1$
3: set $t \leftarrow 1$
4: repeat
5: Execute Algorithm 1 for $c \cdot \frac{1}{\epsilon\,\alpha} \cdot \log \tilde{n}$ rounds
6: $t \leftarrow t + 1$
7: $\tilde{n} \leftarrow (1+\epsilon)^t$
8: $T \leftarrow (1+\epsilon)\frac{\tilde{n}}{m}$
9: until indefinitely
10: end for

Σχήμα 2.5 Αλγόριθμος αγνώστου αριθμού χρηστών [1]

2.5 Distributed Selfish Load Balancing [2]

Η συγκεκριμένη μελέτη επικεντρώνεται στην εύρεση εποικοδομητικής κατανομής φόρτου εργασίας, που να επιλύνει αποδοτικά το πρόβλημα της ισάριθμης κατανομής μιας ομάδας διαδικασιών στους πόρους-εξυπηρετητές του συστήματος. Στόχος είναι να βρεθεί η κατάλληλη κατανομή που να συνδέει κάθε διεργασία με εγωιστή χρήστη, αυτόν δηλαδή που θα αποσκοπεί να βελτιώσει μόνο τη δική του κατάσταση, αδιαφορώντας για τη γενική κατάσταση που θα επιφέρει το δίκτυο με την ενέργειά του, και να απαιτεί από αυτόν να επιλέξει κατάλληλο πόρο που θα τον εξυπηρετήσει. Το κόστος κάθε πόρου ορίζεται ως ο αριθμός των χρηστών που τον επιλέγουν. Οι χρήστες, μεταναστεύουν από τους υπερφορτωμένους στους λιγότερο επιβαρυμένους πόρους, που ονομάζουμε υποφορτωμένους. Το πρωτόκολλο μπορεί να εφαρμοστεί σε Ισχυρά Κατανεμημένη Ρύθμιση, Strongly Distributed Setting, χωρίς κεντρική διαχείριση και έχει καλή σύγκλιση.

2.5.1 Περιγραφή Έρευνας

Στα σενάρια που μελετούνται, εάν το κόστος για τον καταναλωτή είναι μεγαλύτερο στον τρέχον εξυπηρετικό πόρο συγκριτικά με κάποιο άλλο, τότε αυτός μεταναστεύει σε αυτόν με κάποια πιθανότητα. Εάν η εναλλαγή επιφέρει μικρή διαφορά στην επίδοση, μειώνοντας ελάχιστα το λάθος, τότε το σύστημα δεν επωμίζεται το κόστος της εναλλαγής.

Η παρούσα μελέτη εστιάζει στο χρόνο σύγκλισης που χρειάζεται ώστε το σύστημα να επέλθει σε ισορροπία και όχι στην ποιότητα ισορροπίας. Μιλώντας για ποιότητα ισορροπίας εννοούμε το πόσο ορθή είναι η ισορροπία, εάν υπήρχε καλύτερος τρόπος κατανομής του φόρτου εργασίας, κτλ.

2.5.2 Μελέτη Αλγορίθμου και Βελτιστοποίηση

Στον αλγόριθμο που χρησιμοποιείται, κάθε διεργασία επιλέγει τυχαίο πόρο να την εξυπηρετήσει και υπολογίζει το φορτίο του. Στη συνέχεια επιλέγει άλλο τυχαίο κόμβο και τον εξετάζει. Εάν το φορτίο του είναι μικρότερο από το φορτίο του πόρου που ήδη ευρίσκεται , τότε μεταναστεύει εκεί με κάποια πιθανότητα.

Το υπολογιστικό κόστος του αλγόριθμου είναι περίπου το ίδιο με την περίπτωση του εξ αρχής υπολογισμού και απόφασης του βέλτιστου πόρου που μπορεί να εξυπηρετήσει καλύτερα τον αντιπρόσωπο σε όλα τα αιτήματα που θα του αναθέσει.

Ενδεικτικά ο αλγόριθμος παρατίθεται παρακάτω:

For each task b **do** in parallel
  Let i_b be the current resource of task b
  Choose resource j_b uniformly at random
  Let $X_{i_b}(t)$ be the current load of resource i
  Let $X_{j_b}(t)$ be the current load of resource j
  If $X_{i_b}(t) > X_{j_b}(t)$ **then**
    Move task b from resource i_b to j_b with probability $1 - X_{j_b}(t)/X_{i_b}(t)$

Σχήμα 2.6 Πρωτόκολλο που επιτρέπει «ουδέτερες» κινήσεις στους χρήστες [2]

Η διαδικασία του αλγόριθμου μέχρι το σύστημα να επέλθει σε ισορροπία όσον αφορά την δίκαιη κατανομή διεργασιών στους πόρους του συστήματος, είναι χρονοβόρα. Ωστόσο τα θετικά από τη χρήση του αλγορίθμου μπορεί να αντισταθμίσουν το υπολογιστικό κόστος αφού δε χρειαζόμαστε πλήρη, γενική γνώση του δικτύου, πλην των διαθέσιμων πόρων που πρέπει να γνωρίζουμε εκτενώς. Επομένως δεν απαιτείται κάθε διεργασία να γνωρίζει το πλήθος των ολικών διεργασιών και να βασίζει σε αυτή την πληροφορία τους υπολογισμούς της. Επιπλέον, για τη μετανάστευση, μια διεργασία χρειάζεται να ρωτήσει το φόρτο εργασίας μόνο ενός πόρου, του υποψήφιου προς μετανάστευση. Η πολιτική μετανάστευσης που χρησιμοποιείται επαναλαμβάνεται έως ότου στο σύστημα επέλθει σε κατάσταση στην οποία κανένα χρήστη δεν συμφέρει να αλλάξει πόρο εξυπηρέτησης, δηλαδή έως ότου στο σύστημα έχουμε Ισορροπία Nash.

Μια βελτιστοποίηση του αλγόριθμου, η οποία δίνει και καλύτερα αποτελέσματα είναι η διεργασία να μεταναστεύει σε άλλο κόμβο μόνο εάν αυτός έχει αρκετή διαφορά φόρτου εργασίας και όχι απλά εάν έχει κάποια διαφορά. Μόνο δηλαδή με αρκετή διαφορά φόρτου αξίζει η διεργασία να επωμιστεί το κόστος της μετανάστευσης. Ο βελτιστοποιημένος αλγόριθμος παρατίθεται παρακάτω

For each task b **do** in parallel
  Let i_b be the current resource of task b
  Choose resource j_b uniformly at random
  Let $X_{i_b}(t)$ be the current load of resource i
  Let $X_{j_b}(t)$ be the current load of resource j
  If $X_{i_b}(t) > X_{j_b}(t) + 1$ **then**
    Move task b from resource i_b to j_b with probability $1 - X_{j_b}(t)/X_{i_b}(t)$

Σχήμα 2.7 Πρωτόκολλο που δεν επιτρέπει «ουδέτερες» κινήσεις στους χρήστες[2]

2.6 Distributed Selfish Load Balancing with Weights and Speeds [5]

2.6.1 Περιγραφή Έρευνας

Σε αυτό το άρθρο παρουσιάζεται η μελέτη της περίπτωσης όπου οι διεργασίες έχουν διαφορετικά βάρη, όπως ίσχυε στους αλγόριθμους που έως τώρα μελετήσαμε, αλλά και οι κόμβοι επίσης είναι διαφορετικοί μεταξύ τους καθώς έχουν διαφορετική ταχύτητα εξυπηρέτησης, speed .

Η διαφορά λοιπόν με την προηγούμενη προσέγγιση του προβλήματος, Distributed Selfish Load Balancing, είναι ότι τώρα λαμβάνονται υπ' όψιν και τα βάρη των διεργασιών πριν προβούμε σε μετανάστευση. Επιπλέον, η διεργασία μπορεί να μεταναστεύσει μόνο σε γειτονικό της κόμβο και όχι σε οποιοδήποτε άλλο που επιλέγει τυχαία. Περιορίζεται δηλαδή και ο αριθμός των υποψήφιων κόμβων από τους οποίους θα κάνει τυχαία επιλογή και θα επιχειρήσει να μεταναστεύσει.

2.6.2 Μελέτη Αλγορίθμου

Σε κάθε γύρο, κάθε διεργασία μπορεί να ζητήσει το φόρτο ενός γειτονικού κόμβου-επεξεργαστή που επιλέγει τυχαία. Εάν ο φόρτος είναι μικρότερος από το φόρτο του κόμβου-επεξεργαστή που ήδη ευρίσκεται, τότε μπορεί να μεταναστεύσει εκεί με κάποια πιθανότητα. Η πιθανότητα αυτή είναι συναρτήσει της διαφοράς του φόρτου εργασίας των δυο κόμβων-επεξεργαστών, του υπεύθυνου για τη διεργασία και του υποψήφιου προς μετανάστευση, αλλά και του βάρους των διεργασιών. Ο βαθμός της συνάρτησης αυξάνεται εάν η διεργασία με το μεγαλύτερο βάρος θα μετανάστευε στο συγκεκριμένο κόμβο.

2.6.3 Σχολιασμός Αλγορίθμου

Εδώ αξίζει να σημειωθεί ότι με μεγάλη πιθανότητα μετανάστευσης, το σύστημα δε θα φτάσει ποτέ σε κατάσταση ισορροπίας καθώς οι κόμβοι θα μεταναστεύουν συνεχώς. Επιπλέον κάθε κόμβος στέλλει την προσδοκώμενη εκροή, flow, του στους γείτονές του για να μπορούν πιο εύκολα οι διεργασίες να λάβουν τις αποφάσεις τους.

Με τον αλγόριθμο που λαμβάνει υπ όψιν και τα βάρη των διεργασιών πετυχαίνουμε καλύτερο χρόνο σύγκλισης, οπότε το σύστημα επέρχεται γρηγορότερα σε κατάσταση ισορροπίας.

Algorithm 2: Distributed Selfish Load Balancing for weighted tasks

begin
  foreach *task* ℓ *in parallel* **do**
    Let $i = i(l)$ be the current machine of task ℓ
    Choose a neighboring machine j uniformly at random
    if $\ell_i - \ell_j > \frac{1}{s_j}$ **then**
      Move task ℓ from node i to node j with probability

$$p_{ij} := \frac{\deg(i)}{d_{i,j}} \cdot \frac{W_i - W_j}{2\alpha \cdot W_i}$$

    end
  end
end

Σχήμα 2.7 Αλγόριθμος Εγωιστών Χρηστών με βάρη στις διεργασίες [5]

2.7 Uncertainly in Spatially Explicit Population Models [4]

2.7.1 Περιγραφή Έρευνας

Σε αυτό το άρθρο μελετάται το ποσοστό αβεβαιότητας, καθώς επίσης και τους παράγοντες που την προκαλούν, σε μοντέλα που διαχωρίζουν σε τοποθεσίες-μέρη το χώρο που κυμαίνεται η μετανάστευση των διεργασιών προς εξισορρόπηση του φόρτου εργασίας.

Στη συγκεκριμένη μελέτη, διεργασίες θεωρούνται τα πουλιά-κάτοικοι και επεξεργαστές-κόμβοι οι διάφορες περιοχές της χώρας, patches στις οποίες

κυμαίνονται. Συγκεκριμένα εξετάζονται και αναλύονται έξι σημαντικές πηγές αβεβαιότητας: 1. Χάρτης Κατοικίας, habitat map, 2. Αλγόριθμος Διασκορπισμού, Dispersal Algorithm, 3. Μέγεθος Πληθυσμού σε αυγά, clutch size, 4. Απόσταση μεταξύ των Περιοχών που εξετάζονται, edge effect, 5. Απόσταση Διασκορπισμού-απόσταση θα διασχίσουν τα πουλιά για να φτάσουν στον προορισμό τους- Dispersal Distance, και 6. Εσωτερική Μεταβλητότητα του Μοντέλου, Intrinsic Variability.

2.7.2 Είδος Πουλιών

Ως κεντρικό είδος παρατήρησης και μελέτης συμπεριφοράς χρησιμοποιούμε ένα ιδιαίτερο είδος πουλιού, το wood thrush. Χρησιμοποιούμε μοντέλο που προσομοιώνει τη χρήση της κατοικίας, τη γονιμότητα, το διασκορπισμό και τη θνησιμότητα των πουλιών του είδους. Συγκεκριμένα, προσομοιώνουμε τη συμπεριφορά τριών ειδών πουλιών του wood thrush: ενηλίκων που γεννούν αυγά και ζουν σε συγκεκριμένα μέρη αποφεύγοντας τη μετανάστευση, breeders, ενηλίκων που δε γεννούν αυγά και δεν τείνουν να ζουν σε μια σταθερή περιοχή , floaters και των ανήλικων πουλιών, juveniles.

2.7.3 Μοντέλο Αναπαραγωγής

Στο μοντέλο που χρησιμοποιήθηκε, αρχικά η χώρα που έλαβε μέρος αρχικοποιείται με breeders. Κάθε breeder γεννά αυγά σε αναλογία που καθορίζεται από κάποιες παραμέτρους όπως: μέγεθος περιοχής που ζει, αριθμό κλωσσοπούλων που ήδη φροντίζει, ποσοστό παρασιτισμού, των εξωτερικών οργανισμών που παρασιτούν στις φωλιές των πουλιών και επηρεάζουν τη θνησιμότητα και ποσοστό πουλιών που γίνονται θύματα αρπαγής από άλλα πουλιά ή από ανθρώπινη παρέμβαση παράγοντας που επίσης επηρεάζει τη θνησιμότητα .

Σχετικά με την αναπαραγωγή, κάθε άτομο έχει πιθανότητα να πεθάνει την επόμενη χρονική στιγμή και η πιθανότητα αυτή βασίζεται στην αναμενόμενη μακροζωία του. Τα breeders έχουν χαμηλότερη θνησιμότητα από τα floaters και juveniles. Εάν μια περιοχή έχει περισσότερα πουλιά από όσα μπορεί να φιλοξενήσει, τα πουλιά υπερβαίνουν την χωρητικότητά της τότε γίνεται διασκορπισμός, Dispersal. Διασκορπισμός είναι τα αποτέλεσμα της συνάρτησης με δύο εισόδους-παραμέτρους: (α) μέγιστης απόστασης που το πουλί θα διασκορπιστεί, range και (β) του αριθμού

των συνολικών φορών που ένα πουλί θα επιχειρήσει να διασκορπιστεί προτού κατασταλάξει, προτού αποφασίσει για τη μόνιμη κατοικία του, mobility. Εάν το πουλί φθάσει σε περιοχή με διαθέσιμο έδαφος εγκατάστασης, εγκατασταίνεται. Εάν όχι, τότε είτε θα μετατραπεί σε floater, είτε θα επιχειρήσει να διασκορπιστεί σε νέα περιοχή, ενέργεια στην οποία μπορεί να μεταβεί μόνο εάν δεν έχει ξεπεράσει το όριο φορών που μπορεί να διασκορπιστεί. Τα floaters και juveniles έχουν μεγαλύτερη πιθανότητα διασκορπισμού από τα breeders διότι τα δεύτερα έχουν μεγαλύτερη «πιστότητα» στην τοποθεσία τους , site fidelity, κι έτσι τείνουν να παραμένουν στο μέρος τους και τον επόμενο χρόνο. Με τον πρώτο τους διασκορπισμό τα juveniles γίνονται ενήλικα breeders και ξεκινά νέος κύκλος ζωής.

Σε κάθε χρονική στιγμή, το μέγεθος του πληθυσμού μιας περιοχής ορίζεται ως το άθροισμα (α) των επιζώντων ενήλικων κατοίκων (β) των juveniles – μεταναστών που ήρθαν από άλλες περιοχές και (γ) των juveniles που έφυγαν από τη συγκεκριμένη περιοχή (τη γενέθλιό τους). Τα floaters δε συμπεριλαμβάνονται στην καταμέτρηση του πληθυσμού της περιοχής που διανέμουν αλλά συμπεριλαμβάνονται στην καταμέτρηση του ολικού πληθυσμού της χώρας.

Ως τιμές εξόδου του μοντέλου παίρνουμε: (α) τον ολικό πληθυσμό, (β) τον αριθμό των περιοχών της χώρας, (γ) τον αριθμό των πουλιών κάθε περιοχής και (δ) τον αριθμό των φορών που κάθε πληθυσμός, breeders , floaters και juveniles, περιοχής διασκορπίστηκε και επανακατοικήθηκε. Το μοντέλο χρησιμοποιήθηκε για την πρόβλεψη της αφθονίας των ειδών και την μελέτη της μακροχρόνιας εμμονής του είδους σε συγκεκριμένες περιοχές καθώς επίσης και για την εύρεση περιοχών διαδραματίζουν σημαντικό ρόλο στην συνεκτικότητα.

Η προσομοίωση ξεκινά μία χώρα, landscape, της οποίας κάθε περιοχή έχει ορισμένη χωρητικότητα οποία εξαρτάται από την έκταση τη περιοχής και το μέγεθος του ολικού πληθυσμού, breeders , floaters και juveniles.

Παρακάτω παρουσιάζεται ο αλγόριθμος σε μορφή ψευδικώδικα

```
BEGIN
  Initialize half of landscape's total carrying capacity
  For each population do
    While stopping conditions do not true do
    Begin
      While population not in load balance do
      Begin
        Become a floater with probability P1
        Else/or
        Begin //start dispersal procedure
          If breeder Then
            Produce offspring at rate r (function of parameters of Table_1)
          Bird dies with probability P2 (based on expected longevity, depends on species)
          Let i(b) be the patch that bird b lives in
          Let xi(b) is the current load of resource i
          Let T(i) is the maximum load that patch I can support (entertain)
          If xi(b) > T(i) Then
          Begin
            Decide whether to dispersal or not (floaters: possibility→0.1, breeders: possibility→0.9)
            Make dispersal decision with probability =a_j exp(-θd_ij)
             Choose dispersal method between a) Euclidian distance
                                             b) Least-cost path
          End
          If <dispersal> AND <juveniles> Then
            become <breeders>
        End
      Reparameterizing the model AND renew the Table based on new results
      If <escape condition> Then
        Escape
    End
  End
END.
```

2.7.4 Χάρτες Πληθυσμού

Το μοντέλο έτρεξε σε δύο χάρτες πληθυσμο, generous map και strict map που παρουσίαζαν διαφορετικά χαρακτηριστικά όπως ποσοστό αραιής βλάστησης, ανεπτυγμένης περιοχής, δασικής περιοχής ,hardwook, δασικής περιοχής με πεύκα, νερού και βλάστησης. Κάθε σημείο, pixel, της περιοχής που πληρούσε συγκεκριμένα ποσοστά των παραπάνω παραμέτρων θεωρήθηκε τόπος κατοικίας, habitat area, και πολλοί κατοικήσιμοι τόποι στη συνέχεια ομαδοποιήθηκαν σε περιοχές, patches.

Οι χάρτες πληθυσμού ήταν ο πλούσιος χάρτης, generous map, που περιλάμβανε 832 περιοχές με σύνολο 3230 κατοικίες και μέσο μέγεθος περιοχής 4ha, από τις οποίες το 60% περιείχαν μόνο ένα ζευγάρι breeders και ο αυστηρός χάρτης, strict map, που

περιλάμβανε 306 περιοχές με σύνολο 794 κατοικίες και μέσο μέγεθος περιοχής 2.5 ha, από τις οποίες το 48% περιείχαν ένα ζευγάρι breeders.

2.7.5 Μοντέλα Διασκορπισμού

Ο τρόπος διαχωρισμού της χώρας, landscape, σε περιοχές γίνεται σύμφωνα με την πιθανότητα μετακίνησης πουλιών μεταξύ δύο περιοχών. Αρχικά ορίζεται εκθετικά αρνητική συνάρτηση απόστασης του κοντινότερου γείτονα, μεταξύ δύο άκρων – περιοχών, και του μεγέθους της περιοχής και βάση αυτής της συνάρτησης υπολογίζεται η πιθανότητα διασκορπισμού από μια περιοχή i σε μια περιοχή j, με τη συνάρτηση:

$$p_{ij} = a_j \exp(-\theta d_{ij})$$

Μεγαλύτερες σε έκταση περιοχές είναι πιθανότεροι μεταναστευτικοί προορισμοί διασκορπισμού για τα πουλιά. Χρησιμοποιούνται δύο μέθοδοι διασκορπισμού μεταξύ των περιοχών:

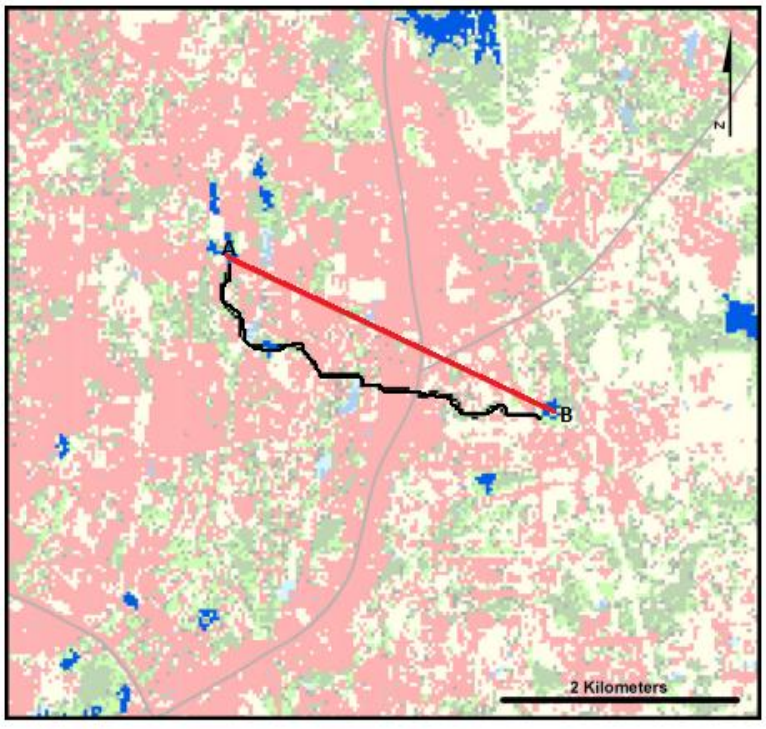
1. Η Απόσταση Διασποράς Περιοχών: βασίζεται στην Ευκλείδεια, Euclidian, απόσταση μεταξύ των περιοχών και του μεγέθους του πληθυσμού της περιοχής που ο πληθυσμός πρόκειται να μεταναστεύσει.
2. Μονοπάτι με το Μικρότερο Κόστος, Least Cost. Με τη μέθοδο αυτή ουσιαστικά υπολογίζεται το ελάχιστο «κόστος - ταξιδιού», travel cost, ανάμεσα σε κάθε ζευγάρι γειτονικών περιοχών.

Για τον υπολογισμό του κόστους μεταβίβασης σε μια περιοχή λαμβάνεται υπ όψιν η προσπάθεια που καταβάλλει το πουλί για να διανύσει την απόσταση μέχρι να φτάσει στην περιοχή που εξετάζουμε.

Επιπλέον, η προσπάθεια που καταβάλλει το πουλί, επιτρέπει στο μοντέλο να αναγνωρίσει τις προτιμήσεις «πτήσεων» των πουλιών. Για παράδειγμα παρατηρήθηκε ότι τα πουλιά προτιμούν να πετούν διαμέσου των «δασικών διαδρόμων», wooded corridors, παρά να πετούν σε ευθεία γραμμή, Euclidean distance, για να φτάσουν σε γείτονα.

Επίσης, αναθέτουμε συγκεκριμένες τιμές κόστους σε κάθε περιοχή που καλύπτει τη χώρα ανάλογα με τα χαρακτηριστικά της, όπως αραιή βλάστηση, ρηχά νερά, ανεπτυγμένη ή όχι, κτλ. Όσο ψηλότερο κόστος, τόσο μεγαλύτερη και η δυσκολία να διασχίσει το πουλί την περιοχή.

Στο μοντέλο θεωρούμε ότι τα πουλιά ακολουθούν τη διαδρομή ελαχίστου κόστους, cheapest path, για να φτάσουν στον προορισμό τους, στην περιοχή που θα μεταναστεύσουν. Η διαδρομή αυτή πολλές φορές παρόλο που είναι αυτή με το ελάχιστο κόστος – προσπάθεια για τα πουλιά, εντούτοις δεν είναι η συντομότερη. Αυτό συμβαίνει διότι οι ανεπτυγμένες – πυκνοκατοικημένες περιοχές έχουν μεγαλύτερο κόστος και αποφεύγονται από τα πουλιά, τα οποία προτιμούν να διασχίσουν τις δασικές περιοχές. Αποτέλεσμα είναι αντί να πετούν σε ευθεία γραμμή προς τον προορισμό τους, να κάνουν γύρο έξω από την παρεμβαλλόμενη περιοχή διότι χρειάζεται να καταβάλουν λιγότερη προσπάθεια. Η σύγκριση φαίνεται στην παρακάτω εικόνα.



Σχήμα 2.8 Τα πουλιά πετούν από την περιοχή Α προς Β ακολουθώντας την «μπερδεμένη» περιοχή (μαύρο διάνυσμα) και όχι την ανεπτυγμένη (κόκκινη ευθεία). [4]

Μετά από κάθε εκτέλεση, το μοντέλο επαναμετρίζει τις περιοχές σύμφωνα με τις νέες πληροφορίες που αποκόμισε από το αποτέλεσμα της εκτέλεσης και ανανεώνει τους πίνακες στοιχείων για τις περιοχές.

Σημαντική είναι η διευκρίνιση ότι πριν ξεκινήσει τη δοκιμή, δεν υπάρχει πληροφορία σχετικά με την καταλληλότητα των μεθόδων καθορισμού πιθανότητας διασκορπισμού. Επομένως δεν υπάρχουν στοιχεία στο δίκτυο που να υποδηλώνουν την καταλληλότητα της κάθε μεθοδολογίας.

2.7.6 Παράμετροι Μοντέλων

Οι παράμετροι που μελετούνται στις προσομοιώσεις που παρουσιάζονται είναι οι εξής:

1. Ποσοστό παρασιτισμού στις φωλιές των πουλιών - οι ξένοι οργανισμοί που παρασιτούν στις φωλιές των πουλιών και αυξάνουν την θνησιμότητά τους, Nest parasitism rate.
2. Ποσοστό θήρευσης της φωλιάς, Nest predation rate
3. Απόσταση μεταξύ των περιοχών της χώρας, Edge distance
4. Ετήσια επιβίωση breeders, Annual adult survival
5. Ετήσια επιβίωση juveniles και floaters, Juvenile/floater survival
6. Μέγεθος περιοχής – αριθμός αυγών που υπάρχουν στην περιοχή, Clutch size
7. Αριθμός νεογνών – κλωσσοπούλων , Number of broods attempted
8. Μέγεθος συνολικού εδάφους, Territory size
9. Απόσταση διασκορπισμού, Dispersal distance
10. Πιστότητα juveniles και floaters - κατά πόσο παραμένουν στην γενέτειρα περιοχή τους, Juvenile/floater fidelity
11. Πιστότητα breeder, Breeder site fidelity

2.7.7 Προσομοιώσεις - Αποτελέσματα

Με τη χρήση του μοντέλου διεξήχθηκαν πολλές προσομοιώσεις, αλλάζοντας κάθε φορά τις παραπάνω παραμέτρους με όλους τους πιθανούς συνδυασμούς, πρώτα έχοντας διαφορετική μόνο μία παράμετρο και μετά όλους τους συνδυασμούς τους – πρώτη και δεύτερη μόνο, πρώτη, δεύτερη, Τρίτη παράμετρο κοκ… σε ένα παραγοντικό σχεδιασμό 32 καταστάσεων. Συνολικά έτρεξαν 3200 μοντέλα. Κάθε προσομοίωση έτρεχε για 100 χρόνια ώστε να εξακριβωθεί η τάση του πληθυσμού, ξεφεύγοντας από την αρχική του κατάσταση. Στη συνέχεια επιλέχθηκαν 100 αντίγραφα από κάθε προσομοίωση και υπολογίστηκε ο Μέσος Όρος. Ξεχωριστά εξετάστηκαν οι παράμετροι που επέφεραν σημαντικές διαφορές στα αποτελέσματα των δύο χαρτών, του πλούσιου χάρτη, και του αυστηρού χάρτη. Οι παράμετροι αυτοί

ήταν ο αριθμός πουλιών της χώρας και ποσοστό χωρητικότητας που κατέλαβαν. Το συνολικό λάθος του μοντέλου εξετάστηκε με τη χρήση 3 τυχαίων αντιγράφων για κάθε μια από τις 32 περιπτώσεις ώστε να υπάρχουν ίδιες πιθανότητες ορθότητας.

2.7.8 Συμπεράσματα

Σε γενικές γραμμές, αυτό που παρατηρήθηκε σχετικά με τη συμπεριφορά των πουλιών είναι ότι όταν υπάρχει μεγάλη απόσταση μεταξύ των περιοχών που κατοικούν, ως μέθοδο διασκορπισμού τα πουλιά χρησιμοποιούν το μονοπάτι με το μικρότερο κόστος. Εντούτοις, οι μελετητές δεν κατάφεραν να εξακριβώσουν το πότε ακριβώς επιχειρεί κάθε πληθυσμός να μεταναστεύσει: προτού ή αφότου η περιοχή που διανέμουν φτάσει σε μέγιστη χωρητικότητα.

Επιπλέον, το μέγεθος της περιοχής σε πληθυσμό, κατά πόσο η περιοχή είναι ή όχι πυκνοκατοικημένη – αναπτυγμένη, φαίνεται να επηρεάζει την αβεβαιότητα. Αυτό συμβαίνει διότι τα πουλιά προτιμούν τις λιγότερο ανεπτυγμένες περιοχές για διασκορπισμό.

Τέλος, ο παράγοντας της απόστασης διασκορπισμού, φαίνεται να επηρεάζει τις ενέργειες διασκορπισμού, αλλά όχι στο βαθμό που επηρεάζει το μέγεθος της περιοχής.

2.8 Σύγκριση Μοντέλων Συμπεριφοράς Πουλιών και Κόμβων Δικτύου

Αντιπαραβάλλοντας τη συμπεριφορά των πουλιών στις προσομοιώσεις του μοντέλου διακρίνουμε ομοιότητες και διαφορές με τα προηγούμενα μοντέλα που προτάθηκαν για εξισορρόπηση φόρτου εργασίας στους κόμβους ενός δικτύου.

Ξεκινώντας με το μοντέλο που προτάθηκε για εξισορρόπηση φόρτου εργασίας σε κατανεμημένα συστήματα με ποιότητα υπηρεσίας, μπορούμε να αντιστοιχίσουμε την απόφαση μετανάστευσης των πουλιών όταν η περιοχή που διαμένουν είναι ανεπτυγμένη – πυκνοκατοικημένη, με την απόφαση της διεργασίας να αλλάξει κόμβο εξυπηρέτησης εάν ο τωρινός που την εξυπηρετεί έχει ξεπεράσει το όριο χρηστών που ανέχεται η διεργασία. Επιπλέον, και στους δύο αλγορίθμους υπάρχει τοπική και όχι ολική πληροφόρηση για την κατάσταση. Τα πουλιά δε γνωρίζουν το βαθμό ανάπτυξης όλων των περιοχών αλλά μόνο της δικής τους και αυτής στην

οποία εξετάζουν το ενδεχόμενο διασκορπισμού. Αντίστοιχα, οι διεργασίες των χρηστών στα κατανεμημένα συστήματα με ποιότητα υπηρεσίας, δεν κατέχουν τη γενική κατάσταση του συστήματος αλλά στηρίζουν τις αποφάσεις τους μόνο στη διαθέσιμη τοπική πληροφορία.

Η μόνη διαφορά στη συμπεριφορά των αλγορίθμων είναι ότι τα πουλιά λαμβάνουν υπ όψιν και την απόσταση που θα χρειαστεί να διανύσουν για να φτάσουν στην νέα περιοχή ενώ οι διεργασίες λαμβάνουν υπ όψιν μόνο το όριο ανεκτικότητας κάθε χρήστη για εξυπηρέτηση.

Προχωρώντας με τον αλγόριθμο εξισορρόπησης φόρτου εργασίας σε κατανεμημένους εγωιστές χρήστες, μπορούμε να διακρίνουμε δύο βασικές διαφορές με τον αλγόριθμο που χρησιμοποιούν τα πουλιά: 1. Κανένας πληθυσμός πουλιών δεν εξετάζει την περίπτωση διασκορπισμού εάν δεν αντιμετωπίζει προβλήματα χωρητικότητας στην περιοχή που διανέμει. Αντίθετα, μια διεργασία σε περιβάλλοντα με εγωιστές χρήστες, εξετάζει την πιθανότητα μετανάστευσης σε κόμβο όταν διαπιστώσει ότι μπορεί να την εξυπηρετήσει καλύτερα. 2. Τα πουλιά μεταναστεύουν μόνο σε γειτονικές περιοχές ενώ μια διεργασία μπορεί να μεταβεί και σε μη γειτονικό κόμβο προς επεξεργασία εάν εξακριβώσει ότι αυτός είναι καταλληλότερος από τον τρέχων που την εξυπηρετεί.

Αντιστοιχία στους δύο αλγόριθμους θα μπορούσε να θεωρηθεί η εξέταση του χρόνου μετανάστευσης της διεργασίας με την απόσταση διασκορπισμού των πουλιών. Οι διεργασίες, όταν αναφερόμαστε σε εγωιστές χρήστες, λαμβάνουν υπ όψιν τους το χρόνο που θα χρειαστούν για τη μεταβίβαση στο νέο κόμβο. Αντίστοιχα, τα πουλιά λαμβάνουν υπ όψιν τους την απόσταση που θα χρειαστεί να διανύσουν για να φτάσουν στη νέα περιοχή

Συνεχίζοντας με την εξελιγμένη μορφή του παραπάνω αλγόριθμου, προσθέτοντας δηλαδή ταχύτητα και στους κόμβους εξυπηρέτησης των διεργασιών, έχουμε σενάρια όπου οι εγωιστές χρήστες έχουν βάρη, κατώτατο όριο εξυπηρέτησης που ανέχονται και οι κόμβοι εξυπηρέτησης ταχύτητες. Αντίστοιχα, στον αλγόριθμο που χρησιμοποιούν τα πουλιά, έχουμε διαφορετικούς πληθυσμούς πουλιών οι οποίοι αποφασίζουν να μεταναστεύσουν κάτω από διαφορετικές συνθήκες και περιοχές διασκορπισμού με διαφορετικές «προδιαγραφές», όπως έκταση, μέγεθος-φορτίο,

μορφολογία. Επιπλέον, όπως οι διεργασίες έτσι και τα πουλιά, μεταναστεύουν σε γειτονικούς κόμβους και περιοχές αντίστοιχα.

Οι δύο αλγόριθμοι διαφέρουν στη χρονική στιγμή που εξετάζουν την περίπτωση μετανάστευσης. Οι διεργασίες των εγωιστών χρηστών αποφασίζουν να μεταναστεύσουν με κάποια πιθανότητα εάν εξακριβώσουν ότι ένας από τους γείτονές τους μπορεί να τους εξυπηρετήσει καλύτερα. Αντιθέτως, τα πουλιά εξετάζουν περίπτωση διασκορπισμού μόνο όταν διαπιστώσουν ότι η περιοχή που βρίσκονται δεν τους εξυπηρετεί επαρκώς.

Πλησιέστερες με τη συμπεριφορά των πουλιών μπορούμε να πούμε ότι είναι οι στρατηγικές Βαθμίδας Μοντέλου και Διάχυσης Αρχικοποίησης Αποστολέα της κατανομής φόρτου στον παράλληλο προγραμματισμό, καθώς οι υπερφορτωμένες περιοχές προσπαθούν να αποφορτισθούν με μετακινήσεις πληθυσμών. Διαφέρουν με τη Στρατηγική Διάχυσης Αρχικοποίησης Παραλήπτη στην οποία η αποφορτωμένοι κόμβοι αιτούνται φόρτο εργασίας και με την Ιεραρχική Μέθοδο Ισορροπίας και τη Μέθοδο Διάστασης Ανταλλαγής καθώς δεν έχουμε δενδρική μορφή στο δίκτυο των πουλιών.

2.9 Balls and Bins Problem [7]

2.9.1 Επεξήγηση προβλήματος

Στο άρθρο τους «*Balls into Non-uniform Bins*» οι ερευνητές μελετούν την εξισορρόπηση φορτίου στο παιχνίδι balls into bins με διάφορες επιλογές.

Υπάρχουν m μπάλες και n κάδοι. Κάθε μπάλα μπορεί επιλέγει d κάδους. Στη συνέχεια τοποθετείται σε αυτόν με το μικρότερο φόρτο εργασίας. Κάθε κάδος i έχει χωρητικότητα C_i . Και C ορίζεται η συνολική χωρητικότητα των κάδων του παιχνιδιού. Κάθε κάδος i έχει πιθανότητα να επιλεγεί από την μπάλα ως ένας από τις επιλογές της $\frac{C_i}{C}$. Ο φόρτος εργασίας του κάδου ορίζεται ως $\frac{αριθμός\ μπαλών\ που\ τον\ επέλεξαν}{C_i}$. Στόχος της στρατηγικής αυτής είναι η ελαχιστοποίηση του μέγιστου αριθμού μπαλών που ανατίθενται σε κάθε κάδο, που ισοδυναμεί με την ελαχιστοποίηση του φόρτου εργασίας.

2.9.2 Αποτελέσματα Ανάλυσης προβλήματος

Τα αποτελέσματα από την ανάλυση και τις προσομοιώσεις της στρατηγικής, έδειξαν ότι:

1. το μέγιστο φορτίο σε ανόμοια περιβάλλοντα, περιβάλλοντα όπου οι κάδοι έχουν διαφορετική χωρητικότητα, είναι ανεξάρτητο από την ολική χωρητικότητα του συστήματος
2. Με μεγαλύτερη χωρητικότητα στους κάδους, πετυχαίνουμε καλύτερο διαμοιρασμό φορτίου.

2.9.3 Προσομοιώσεις για Ανάλυση Προβλήματος

Για την απόδειξη των παραπάνω έγιναν προσομοιώσεις για τις εξής κατηγορίες σεναρίων παιχνιδιού:

1. Πολλοί κάδοι μεγάλης χωρητικότητας, που μπορούν να έχουν μέχρι 13 μπάλες και λιγότεροι με ελάχιστη χωρητικότητα 1. Για την ανάλυση αυτής της περίπτωσης έγινε μια σειρά προσομοιώσεων όπου κάθε φορά ο αριθμός των κάδων μεγάλης χωρητικότητας μειωνόταν και των κάδων ελάχιστης χωρητικότητας αυξανόταν, αφήνοντας σταθερό τον ολικό αριθμό κάδων. Οι μετρήσεις που πάρθηκαν ήταν ο φόρτος εργασίας κάθε κάδου. Η συμπεριφορά του παιχνιδιού έδειξε ότι όσους περισσότερους κάδους με μεγάλη χωρητικότητα είχαμε τόσο πιο ομοιόμορφη ήταν η κατανομή του φόρτου εργασίας και τόσο μικρότερος ήταν.
2. Πολλοί κάδοι μικρής χωρητικότητας, που μπορούν να έχουν μέχρι και 8 μπάλες και λιγότεροι με ελάχιστη χωρητικότητα 1. Και πάλι στις προσομοιώσεις που έγιναν ο αριθμός κάδων μεγαλύτερης και μικρότερης χωρητικότητας αυξομειωνόταν κρατώντας σταθερό τον ολικό αριθμό κάδων του παιχνιδιού. Από τις μετρήσεις του φόρτου εργασίας κάθε κάδου σε κάθε σενάριο έδειξε να έχουμε και πάλι τα καλύτερα αποτελέσματα κατανομής και μέγιστου φόρτου εργασίας με περισσότερους κάδους μικρής κι όχι ελάχιστης χωρητικότητας. Εδώ πρέπει να σημειωθεί ότι ο μέγιστος φόρτος εργασίας με περισσότερους κάδους μικρής χωρητικότητας ήταν μεν μικρότερος από τα σενάρια με περισσότερους κάδους ελάχιστης χωρητικότητας, αλλά μεγαλύτερος από τα τον μέγιστο στην προηγούμενη κατηγορία σεναρίων.
3. Μικρός αριθμός κάδων με διπλάσια χωρητικότητα από την ελάχιστη, δηλαδή που μπορούν να εξυπηρετήσουν μέχρι και 2 μπάλες και λιγότεροι με ελάχιστη

χωρητικότητα. Όπως και παραπάνω, στις προσομοιώσεις που έγιναν ο αριθμός κάδων μεγαλύτερης και μικρότερης χωρητικότητας αυξομειωνόταν κρατώντας σταθερό τον ολικό αριθμό κάδων του παιχνιδιού. Οι μελετητές εδώ ήθελαν να μελετήσουν την συμπεριφορά του παιχνιδιού με μικρό αριθμό κάδων.

4. Υπερφορτωμένο δίκτυο. Σε αυτή την κατηγορία σεναρίων κάθε φορά ο αριθμός των μπαλών δεκαπλασιαζόταν ενώ η χωρητικότητα όλων των κόμβων ήταν η ίδια και παρέμενε σταθερή. Οι μελετητές σε αυτή την κατηγορία σεναρίων ήθελαν να ερευνήσουν τη συμπεριφορά του παιχνιδιού με πολλές μπάλες.

Κεφάλαιο 3

Αλγόριθμοι Εξισορρόπησης Φόρτου Εργασίας

Σε αυτό το Κεφάλαιο παρουσιάζονται Αλγόριθμοι Εξισορρόπησης Φόρτου Εργασίας που έχουν προταθεί και υλοποιηθεί όπως περιγράφονται σε άρθρα που μελετήσαμε.

3.1 Artificial Life Techniques for Load Balancing Computing Grids [4]

3.1.1 Περιγραφή Έρευνας

Σε αυτό το άρθρο παρουσιάζονται οι δύο βασικοί αλγόριθμοι, ο Γενετικός Αλγόριθμο, Genetic Algorithm, και ο Ερευνητικός Αλγόριθμος Ταμπού, Tabu search που χρησιμοποιούνται σε υπολογιστικό πλέγμα το οποίο ,ενισχύει σημαντικά την επίδοση σε κατανεμημένη αλγοριθμική επεξεργασία Στόχος κάθε μηχανισμού είναι να μεγιστοποιήσει τη χρησιμότητα, utilization, και να ελαχιστοποιήσει το χρόνο εκτέλεσης, execution time, της κάθε διεργασίας.

Οι δύο Αλγόριθμοι, Γενετικός και Ερευνητικός Ταμπού, αντιπαραβάλλονται με τους Καλύτερη Προσαρμογή, Best Fit, Μικρότερο-Μέγιστο, Min-Max, Μεγαλύτερο-Ελάχιστο, Max-Min και Δυστυχής, Sufferage. Από τα αποτελέσματα βλέπουμε τη διαφορά στην αποδοτικότητα των πρώτων δύο συγκριτικά με τους υπόλοιπους.

Πιο κάτω περιγράφεται η λειτουργία κάθε αλγορίθμου και στη συνέχεια θα συγκρίνουμε τις επιδόσεις τους.

3.1.2 Ο Αλγόριθμος Καλύτερης Προσαρμογής

Στο αλγόριθμο Καλύτερης Προσαρμογής οι διεργασίες εκτελούνται με τη σειρά που καταφθάνουν. Η διεργασία ανατίθεται στον κόμβο που θα την εκτελέσει γρηγορότερα. Μόλις ένας κόμβος απελευθερωθεί από τη διεργασία που του ανατέθηκε να εξυπηρετήσει, ανατίθεται σε αυτόν νέα. Η διεργασία δεν έχει δυνατότητα αλλαγής κόμβου εξυπηρέτησης. Εάν όλοι οι κόμβοι είναι απασχολημένοι με δεσμευμένες διεργασίες που εξυπηρετούν, και μία νέα διεργασία καταφθάσει, τότε αυτή μπαίνει σε ουρά αναμονής έως ότου μια διεργασία ολοκληρώσει τη λειτουργία της και απελευθερώσει τον κόμβο που είχε δεσμεύσει για να την εξυπηρετήσει.

3.1.3 Οι Αλγόριθμοι Μικρότερο-Μέγιστο, Μεγαλύτερο-Ελάχιστο και Δυστυχής

Στους αλγόριθμους τρεις Αλγορίθμους, ο τρόπος με τον οποίο γίνεται η διαχείριση της ανάθεσης διεργασιών στους κόμβους, διαφέρει στον τρόπο που ο κάθε απελευθερωμένος κόμβος επιλέγει διεργασία που θα εξυπηρετήσει από το σύνολο των διεργασιών που περιμένουν. Συγκεκριμένα, υπάρχει ένα σύνολο διεργασιών που περιμένουν να εκτελεστούν. Κάθε διεργασία ανατίθεται σε ένα κόμβο ο οποίος αναλαμβάνει να την εξυπηρετήσει. Αρχικά υπολογίζεται ο χρόνος ολοκλήρωσης όλων των διεργασιών και επιλέγεται η διεργασία με τον ελάχιστο χρόνο ολοκλήρωσης. Στη συνέχεια, υπολογίζεται το μετρικό, metric, και εξυπηρετείται η διεργασία με το καλύτερο.

Οι αλγόριθμοι διαφέρουν στον τρόπο υπολογισμού του μετρικού. Συγκεκριμένα, στον Αλγόριθμο Μικρότερο-Μέγιστο κόμβος που καλείται να επιλέξει διεργασία επιλέγει αυτή με το μικρότερο μετρικό, στον Αλγόριθμο Μεγαλύτερο-Μικρότερο, ο κόμβος επιλέγει τη διεργασία με το μεγαλύτερο μετρικό, ενώ στον Δυστυχή, κάθε κόμβος που καλείται να επιλέξει διεργασία, επιλέγει αυτή με τη μεγαλύτερη διαφορά ανάμεσα στα δύο καλύτερα μετρικά, το καλύτερο μετρικό και το αμέσως επόμενο καλύτερο μετρικό. Ουσιαστικά ο αλγόριθμος αντί να λάβει υπ όψιν του μόνο μία

παράμετρο, αυτήν του ελάχιστου χρόνου ολοκλήρωσης, κοιτάζει και τον αμέσως επόμενο καλύτερο χρόνο για να πάρει πιο αντιπροσωπευτικές πληροφορίες για την απόφαση της επιλογής του.

3.1.3 Γενετικός Αλγόριθμος

Ο Γενετικός Αλγόριθμος, χρησιμοποιεί καθοδηγούμενη-τυχαία τεχνική αναζήτησης, guided random search technique. Στόχος είναι η εύρεση έξυπνων λύσεων που να εξυπηρετούν τις διεργασίες που καταφθάνουν πληρώντας τους περιορισμούς σε λογικό χρόνο και κόστος. Χρησιμοποιούνται σε μελέτη μεγάλων περιβαλλόντων όπου η συνεχής έρευνα είναι χρονοβόρα και κοστίζει υπολογιστικά. Οι Γενετικοί αλγόριθμοι είναι αποδοτικοί σε προβλήματα βελτιστοποίησης μηχανικής μάθησης, machine learning.

3.1.3.1 Λειτουργία Γενετικού Αλγορίθμου

Με την ενεργοποίηση του αλγορίθμου, ο πληθυσμός που μελετούμε θεωρούμε ότι τυγχάνει μιας μορφής αρχικοποίηση κατά την οποία διαμορφώνονται τα χρωματοσώματα. Στη συνέχεια αξιολογείται ο πληθυσμός, η γενιά που μελετούμε. Ακολουθεί μια επαναληπτική δομή 200 γενεών. Σε κάθε γύρο δημιουργείται νέα γενιά ως εξής: Αρχικά επιλέγονται k άτομα από τον πληθυσμό που θα συμμετάσχουν στη δημιουργία της νέας γενιάς. Στη συνέχεια εφαρμόζεται η κατάλληλη τεχνική επιλογής γονιδίων για να γίνει το ζευγάρωμα των χρωματοσωμάτων και να παραχθούν τα νέα άτομα, οι απόγονοι, οι οποίοι σε αριθμό είναι κατά k από τον προηγούμενο. Για το κάθε άτομο αποφασίζεται ποια χαρακτηριστικά-χρωματοσώματα θα κληρονομήσει από ποιο γονέα και ακολούθως εφαρμόζεται μετάλλαξη με κάποια πιθανότητα. Η μετάλλαξη αποτρέπει τη σύγκλιση του πληθυσμού. Εάν ο ρυθμός μετάλλαξης είναι μεγάλος, τότε η έννοια της αναπαραγωγής εξασθενεί διότι τα παιδιά δε φέρουν-κληρονομούν τα χαρακτηριστικά των γονέων τους. Ακολουθεί και πάλι αξιολόγηση του πληθυσμού. Εάν ο αριθμός των γενεών από την τελευταία φορά που είχαμε βελτίωση ξεπερνά ένα όριο T, τότε η αναπαραγωγή σταματά. Διαφορετικά συνεχίζεται έως ότου ξεπεράσουμε ένα προκαθορισμένο αριθμό γενεών.

Ο αλγόριθμος μπορεί να χρησιμοποιήσει μία από τις ακόλουθες τεχνικές επιλογής ατόμων ζευγαρώματος.Στην τεχνική Επιλογής Αναλόγως Κατάστασης, Fitness-

proportionate selection, η επιλογή του ζευγαριού γίνεται πιθανοτικά βάση του βαθμού υγείας του, fitness value, του ατόμου που υπολογίζεται μέσω συνάρτησης. Μεγαλύτερος βαθμός υγείας ισοδυναμεί με μεγαλύτερη πιθανότητα. Στην τεχνική Επιλογής Κατάστασης, Rank Selection, τα άτομα κατατάσσονται σε σειρά και επιλέγονται τα πρώτα n από αυτά.
Η τεχνική Επιλογής Τουρνουά, Tournament Selection, τα άτομα κατατάσσονται σε δυάδες και επιλέγονται οι πρώτες n/2 από αυτές. Τέλος, η Τεχνική Ελιτισμού, Elitism, χρησιμοποιείται σε συνδυασμό με μία από τις παραπάνω και ουσιαστικά στο νέο πληθυσμό μεταφέρονται οπωσδήποτε αυτούσια κάποια γονίδια που χαρακτηρίζονται ως «σημαντικά».

3.1.5 Ερευνητικός Αλγόριθμος Ταμπού

Ο Ερευνητικός Αλγόριθμος Ταμπού χρησιμοποιεί προσαρμοστική μνήμη, adaptive memory, για επίλυση προβλημάτων. Χρησιμοποιεί στρατηγικές και επιβεβαιωμένες πληροφορίες για να μιμηθεί ανθρώπινες συμπεριφορές. Ο Αλγόριθμος εξετάζει τη λύση που χρησιμοποιεί ο γείτονάς του σε περίπτωση που είναι καλύτερη από τη δική του και την υιοθετεί.

3.1.5.1 Λειτουργία Ερευνητικού Αλγόριθμου Ταμπού

Αρχικά παράγεται μια αρχική λύση. Ενώσω δεν υπάρχουν καταστάσεις που να σταματούν την επεξεργασία του, σε κάθε γύρο ο αλγόριθμος εξετάζει τις λύσεις που χρησιμοποιούν οι γείτονές του προς επίλυση του ίδιου προβλήματος και τις συγκρίνει με τη δική του, επιλέγοντας να συνεχίσει με την ιδανικότερη από αυτές συμπεριλαμβανομένης και αυτής που ήδη χρησιμοποιεί. Στη συνέχεια αναβαθμίζει τη μνήμη του κατακρατώντας σε λίστα τις προτεινόμενες λύσεις των γειτόνων του.

Υπάρχουν δύο είδη λίστας, Tabu list, που προτάθηκαν στον αλγόριθμο. Αρχικά χρησιμοποιήθηκε η λίστα λύσεων , Solution list, που περιείχε τις τελευταίες k λύσεις που ο αλγόριθμος είχε μελετήσει και απορρίψει. Με αυτό τον τρόπο αποφευγόταν η επανεξέταση των ίδιων λύσεων που μπορεί να οδηγούσε τον αλγόριθμο σε κύκλους, δηλαδή σε περίπτωση που δύο λύσεις κρίνονταν το ίδιο ιδανικές ο αλγόριθμος να

μεταβαίνει συνεχώς από τη μια στην άλλη. Η λύση όμως αυτή ήταν αρκετά χρονοβόρα και είχε μεγάλες ανάγκες σε μνήμη.

Η επόμενη λύση που προτάθηκε ήταν η χρήση μιας λίστας κινήσεων , move list, στην οποία αποθηκεύονταν μόνο οι αλλαγές μεταξύ της υπάρχουσας λύσης και της αναβαθμισμένης στην οποία θα μετέβαινε ο αλγόριθμος.

3.1.6 Σύγκριση Επιδόσεων

Μετά από 200 πειράματα σε πλέγματα εστιάζοντας στην παράμετρο της εξισορρόπησης φορτίου, με τη χρήση των αλγορίθμων, οι μελετητές κατέληξαν στο συμπέρασμα ότι οι Γενετικός Αλγόριθμος και Ερευνητικός Αλγόριθμος Ταμπού, είναι οι ιδανικότεροι συγκριτικά με τους υπόλοιπους. Μειονεκτήματά τους είναι η μνήμη που σπαταλούν για την αποθήκευση των δεδομένων που χρειάζονται για την επεξεργασία τους και το κόστος για τους υπολογισμούς τους στον κόμβο χρονοδρομολόγησης. Οι αποθηκεύσεις μπορεί να προκαλέσουν υπερφόρτωση στο σύστημα ειδικά εάν οι αλγόριθμοι εφαρμοστούν σε μεγάλα συστήματα.

3.2 Strategies for Dynamic Load Balancing on Highly Parallel Computers

3.2.1 Περιγραφή Έρευνας

Η συγκεκριμένη μελέτη αναφέρεται σε τεχνικές δυναμικής κατανομής φορτίου που χρησιμοποιούνται σε παράλληλο προγραμματισμό, όπου η διαχείριση του φόρτου εργασίας γίνεται δυσκολότερη καθώς στο δίκτυο προστίθενται συνεχώς νέοι κόμβοι. Παρουσιάζονται οι στρατηγικές δυναμικής κατανομής εστιάζοντας σε δύο παραμέτρους-κλειδιά για την ορθότητα των ενεργειών.

Αυτά είναι α) η γνώση που υπάρχει για την κατάσταση του δικτύου, που βοηθά στην ακρίβεια του υπολογισμού του φόρτου εργασίας κάθε φορά που λαμβάνονται αποφάσεις για ανάθεση διεργασίας σε κόμβο και β) η υπερφόρτωση του δικτύου εξαιτίας της επεξεργασίας για αποφάσεις μετανάστευσης διεργασιών σε άλλους, καταλληλότερους κόμβους και της επεξεργασίας για την επικοινωνία από την προσπάθεια κατανομής του ολικού φόρτου εργασίας του συστήματος.

Εδώ αξίζει να σημειωθεί ότι η συγκεκριμένη μελέτη εστιάζει σε παράλληλα συστήματα με πολλούς επεξεργαστές. Οπότε αναφερόμαστε σε δυναμική κατανομή φορτίου που ελαχιστοποιούν το χρόνο εκτέλεσης και σε κατανεμημένα συστήματα διότι αυτά μπορούν να εφαρμοστούν στα συστήματα που μελετούμε.

3.2.2 Περιγραφή Μοντέλου

Πιο κάτω παρουσιάζεται το μοντέλο που χρησιμοποιεόται, το οποίο χωρίζεται σε τέσσερις φάσεις.

1. Αξιολόγηση φόρτου εργασίας επεξεργαστή, Processor Load Evaluation: Σε πρώτη φάση γίνεται ο υπολογισμός του φορτίου κάθε επεξεργαστή έτσι ώστε να μπορεί ο κόμβος- υπεύθυνος για την κατανομή του φορτίου να πάρει κατάλληλες αποφάσεις για τις μεταναστεύσεις διεργασιών από τον κόμβο που τις εξυπηρετεί σε άλλο που να μπορεί να τις εξυπηρετήσει γρηγορότερα.
2. Καθορισμός ωφελιμότητας επεξεργαστή για κατανομή του φόρτου εργασίας, Load Balancing Profitability Determination: Σε αυτή τη φάση αποφασίζεται πόσο προσοδοφόρος είναι κάθε επεξεργαστής σε μια δεδομένη χρονική στιγμή ούτως ώστε να αποφασίσουμε εάν είναι αποδοτικό να γίνει κατανομή φόρτου εργασίας. Εάν αξίζει το σύστημα να επομιστεί την υπερφόρτωση επεξεργασίας για την κατανομή. Επιπλέον εδώ καθορίζεται ποιος επεξεργαστής διαθέτει την πληροφορία που χρειαζόμαστε για τους υπολογισμούς μας.
3. Στρατηγική μετανάστευσης, Task Migration Strategy: Καθορισμός της ποσότητας των πόρων και κατ επέκταση του αριθμού των επεξεργαστών και του φόρτου εργασίας των προορισμών για μετανάστευση.
4. Επιλογή καταλληλότερης στρατηγικής για κάθε διεργασία , Task Selection Strategy: Οι επεξεργαστές κάθε πόρου επιλέγουν τις καταλληλότερες διεργασίες που θα διασφαλίσουν αποτελεσματική κατανομή του ολικού φορτίου του συστήματος και τις αποστέλλουν στον κατάλληλο προορισμό.

Παρακάτω ακολουθεί περιγραφή των πέντε στρατηγικών από τις οποίες οι επεξεργαστές θα επιλέξουν την καταλληλότερη για κάθε διεργασία στην φάση 4, στην προσπάθεια τους να φέρουν το σύστημα σε ισορροπία.

3.2.2.1 Gradient Model (GM)

Η στρατηγική, θεωρεί υποφορτωμένους τους επεξεργαστές που ο φόρτος εργασίας τους είναι κάτω από ένα όριο και υπερφορτωμένους τους επεξεργαστές των οποίων ο φόρτος εργασίας είναι πάνω από ένα όριο Οι υπόλοιποι χαρακτηρίζονται ωε μέτριοι

Η συγκεκριμένη στρατηγική κρατά χάρτη με τους υποφορτωμένους επεξεργαστές του συστήματος ώστε να καθοδηγεί τις μεταναστεύσεις. Οι διεργασίες μεταναστεύουν σε γειτονικό επεξεργαστή έως ότου καταλήξουν σε υποφορτωμένο ή σε επεξεργαστή που δεν έχει γείτονα με μικρότερη εγγύτητα.

Με αυτή τη στρατηγική μπορεί να προκληθεί υπερφόρτωση από τις συνεχείς ανανεώσεις του χάρτη δρομολόγησης των διεργασιών από τους υπερφορτωμένους στους υποφορτωμένους επεξεργαστές. Οι ενδεχόμενες αλλαγές στην κατανομή φόρτου μεταξύ των επεξεργαστών, διαδίδεται σταδιακά σε ολόκληρο το χάρτη προς ανανέωση της κατάστασης του συστήματος. Η μετανάστευση των διεργασιών από υπερφορτωμένους σε υποφορτωμένους επεξεργαστές προκαλεί υπερφόρτωση με την αποστολή μεριδίου του φόρτου ενός υπερφορτωμένος επεξεργαστή στον πλησιέστερο υποφορτωμένο. Επιπλέον, κατά τη διάρκεια μιας μετανάστευσης ενδέχεται να αλλάξει η εγγύτητα του χάρτη εάν πολλοί υπερφορτωμένοι επεξεργαστές επιχειρήσουν να αποστείλουν μέρος του φορτίου τους στον ίδιο υποφορτωμένο επεξεργαστή. Σε μία τέτοια περίπτωση θα προκληθεί υπερχείληση από τη μετανάστευση Υπάρχει επίσης η περίπτωση ένας υπερφορτωμένος επεξεργαστής να μη στείλει αρκετό φορτίο σε άλλο γειτονικό επεξεργαστή και να μην αποφορτιστεί, οπότε το σύστημα δεν επέρχεται σε ισορροπία.

3.2.2.2 Στρατηγική Αρχικοποίησης Αποστολέα, Sender Initiated Diffusion (SID)

Στη Στρατηγική Αρχικοποίησης Αποστολέα, κάθε επεξεργαστής ενεργεί ανεξάρτητα, κατανέμοντας το επιπλέον φορτίο του στους γείτονες. Οι πληροφορίες φόρτου εργασίας που κατακρατούνται για κάθε επεξεργαστή υπολογίζονται από το δικό του φορτίο και αυτό των άμεσων γειτόνων του. Οι υπερφορτωμένοι επεξεργαστές προσπαθούν να αποστείλουν φορτίο στους υποφορτωμένους. Σημαντική παράμετρος για την ταχύτητα σύγκλισης αποτελεί η τοπολογία του δικτύου.

3.2.2.3 Στρατηγική Αρχικοποίησης Παραλήπτη, Receiver Initiated Diffusion (RID)

Αντίθετα με την στρατηγική του Αρχικοποίησης Αποστολέα, στη Στρατηγική Αρχικοποίησης Παραλήπτη, οι υποφορτωμένοι επεξεργαστές ζητούν φορτίο από τους υπερφορτωμένους. Συγκεκριμένα, η εκτέλεση ξεκινά με κάθε επεξεργαστή του οποίου ο φόρτος εργασίας είναι κάτω από ένα όριο L. Κάθε υποφορτωμένος επεξεργαστής αποστέλλει αίτημα σε κάποιο υπερφορτωμένο γείτονα να του αποστείλει φορτίο. Ο επεξεργαστής που παραλαμβάνει αίτημα αποστολής φορτίου το εκπληρώνει μόνο εάν αυτό δεν υπερβαίνει το μισό φορτίο από αυτό που ήδη έχει. Δηλαδή κανένας επεξεργαστής δε μπορεί να αποστείλει περισσότερο από το μισό φορτίο του. Με αυτό τον τρόπο οι υποφορτωμένοι επεξεργαστές αναλαμβάνουν την «περίσσεια» φόρτου εργασίας από τους υπερφορτωμένους.

Η διαφορά των δύο στρατηγικών είναι στο μεταναστευτικό μέρος. Στην στρατηγική που ακολουθεί ο Παραλήπτης, ο υποφορτωμένος επεξεργαστής πρώτα αποστέλλει το αίτημα παραλαβής φορτίου και για κάθε αίτημα λαμβάνει έγκριση. Αποστέλλονται δηλαδή δύο μηνύματα έναντι ενός που αποστέλλονται στην Στρατηγική Αποστολέα για τη μετανάστευση της διεργασίας.

3.2.2.4 Ιεραρχική Μέθοδος Ισορροπίας, Hierarchical Balancing Method (HBM)

Η στρατηγική αυτή δημιουργεί ιεραρχικά υποσυστήματα. Η κατανομή του φόρτου εργασίας ξεκινά από τα χαμηλότερα στρώματα , τα φύλλα του δέντρου, με μια μικρή ομάδα επεξεργαστών που τη διαχειρίζονται, και προχωρεί προς τα πιο πάνω, αυξάνοντας συγχρόνως με την ενημέρωση, τη γνώση των ψηλότερων επιπέδων. Επομένως η Μέθοδος της Ιεραρχικής Ισορροπίας οργανώνει ένα πολυεπεξεργαστικό σύστημα, multicomputer system, ιεραρχικά αποκεντρώνοντας τη διαδικασία εξισορρόπησης. Υπάρχουν συγκεκριμένοι-ειδικοί επεξεργαστές που είναι σχεδιασμένοι να ελέγχουν τη λειτουργία της ισορροπίας στα διάφορα επίπεδα ιεραρχίας. Η διαδικασία μετανάστευσης ελέγχεται από τους ενδιάμεσους κόμβους. Οι υπερφορτωμένοι κόμβοι του δυαδικού δέντρου, binary tree, μεταφέρουν το φόρτο εργασίας τους στους υποφορτωμένους. Το μειονέκτημα αυτής της στρατηγικής είναι ότι πολλές φορές η κατανομή του φορτίου δεν είναι ομοιόμορφη

και επιπλέον υπάρχει το κόστος της ενημέρωσης όλων των ανώτερων στρωμάτων κάθε φορά που αναλαμβάνει να εξυπηρετήσει διεργασία ένας κόμβος με χαμηλή θέση στην ιεραρχία.

3.2.2.5. Μέθοδος Διάστασης Ανταλλαγής, Dimension Exchange Method (DEM)

Η Μέθοδος Διάστασης Ανταλλαγής, μοιάζει κατά πολύ με αυτή της Ιεραρχικής Ισορροπίας. Μικρά τμήματα του δικτύου εξισορροπούν το φόρτο εργασίας τους και στη συνέχεια συγχωνεύονται σε μεγαλύτερα μέχρι τελικά σε ολόκληρο το σύστημα να επέλθει ισορροπία. Συγκεκριμένα έχουμε N επεξεργαστές σε logN διαστάσεις. Σε κάθε γύρο γίνεται εξισορρόπηση μιας διάστασης. Η διαφορά με την προηγούμενη στρατηγική είναι ότι τώρα έχουμε προσέγγιση συγχρονισμού. Όλα τα ζεύγη επεξεργαστών στην πρώτη διάσταση εξισορροπούν μεταξύ τους το φόρτο εργασίας τους. Ακολουθούν τα ζεύγη της δεύτερης διάστασης και η διαδικασία συνεχίζεται μέχρι την τέταρτη διάσταση οπότε και το σύστημα επέρχεται σε πλήρη ισορροπία, αφού όλοι οι επεξεργαστές εξισορροπούν το φορτίο με τους γείτονές τους. Επεξεργαστές της ίδιας διάστασης θεωρούνται αυτοί των οποίων οι διευθύνσεις τους διαφέρουν μόνο στο τελευταίο bit. Η στρατηγική αυτή επιφέρει επιβάρυνση στο σύστημα που μοιράζεται ομοιόμορφα ανάμεσα στους επεξεργαστές του συστήματος, εξ αιτίας των μηνυμάτων για ενημέρωση της νέας κατάστασης του συστήματος και των αιτημάτων για μεταφορά φορτίου. Επιπλέον υπάρχει και η επιβάρυνση της μεταφοράς του φορτίου από τον υπερφορτωμένο επεξεργαστή στον υποφορτωμένο.

3.2.3. Μηχανισμός Ανανέωσης Στρατηγικής Ισορροπίας, Load Update Strategy

Για την καλύτερη δυναμική κατανομή του φόρτου εργασίας σε συστήματα προτάθηκε ο μηχανισμός Ανανέωσης Στρατηγικής Ισορροπίας, με τη χρήση του οποίου οι αποφάσεις για κατανομή του φορτίου παίρνονται βάση του μεγέθους του φορτίου ενός συνόλου επεξεργαστών του συστήματος. Σύνολο επεξεργαστών μπορεί να αποτελείται από ένα γείτονα μέχρι και όλους τους επεξεργαστές του συστήματος. Η ποιότητα υπηρεσίας εξαρτάται από:

1. Την ακρίβεια του επεξεργαστή στον υπολογισμό του φόρτου εργασίας.
2. Την «παλαίωση» της πληροφορίας εξ αιτίας της λανθασμένης διασύνδεσης δικτύου και τον προορισμό της πληροφορίας για το φόρτο εργασίας. Ο

παράγοντας αυτός εξαρτάται από την αρχιτεκτονική της μηχανής και τη στρατηγική για κατανομή του φορτίου που χρησιμοποιείται.

3. Τη συχνότητα που στέλλονται μηνύματα για ανανέωση της κατανομής του φόρτου εργασίας του συστήματος.

3.2.4. Αποτελέσματα - Συμπεράσματα

Μελετητές που έκαναν δοκιμές σε παράλληλα συστήματα με δυναμική κατανομή φορτίου με τη χρήση των στρατηγικών που προαναφέρθηκαν, κατέληξαν στο συμπέρασμα ότι καλύτερη στρατηγική είναι η Μέθοδος Διάστασης Ανταλλαγής με την αποδοτικότητά της να εξαρτάται σε μεγάλο βαθμό από την τοπολογία του δικτύου. Επιπλέον, η Στρατηγική Αρχικοποίησης Παραλήπτη αποδεικνύεται κατάλληλη για απλές τοπολογίες και μπορεί να διατηρεί την τοπικότητα της διεργασίας. Καλύτερα αποτελέσματα έχει σε συστήματα ευρύτερου μεγέθους με μεγάλη ποικιλία εφαρμογών.

3.3 Σύγκριση Μοντέλων Συμπεριφοράς Πουλιών και Κόμβων Δικτύου

Συγκρίνοντας τη συμπεριφορά των πουλιών με τον Αλγόριθμο Εύρεσης Ταμπού που χρησιμοποιείται σε υπολογιστικά πλέγματα βλέπουμε ότι και στις δύο περιπτώσεις υπάρχει είδος πίνακα στον οποίο κρατούνται πληροφορίες που βοηθούν τις διεργασίες - πουλιά να αποφασίσουν ποιος κόμβος - περιοχή είναι καταλληλότερος να μεταναστεύσουν/ διασκορπιστούν με κάποια πιθανότητα. Συγκεκριμένα στον Αλγόριθμο Εύρεσης Ταμπού, κρατούνται σε λίστα οι κόμβοι που η διεργασία έχει εξετάσει και απορρίψει τη μετανάστευση στο παρελθόν για να αποφύγει μελλοντική επανεξέταση και επομένως επιπλέον άσκοπη επεξεργασία. Αντίστοιχα, στον αλγόριθμο που χρησιμοποιούν τα πουλιά υπάρχει πίνακας με τα «κόστη - ταξιδιού», travel cost, κάθε περιοχής, τα οποία υπολογίζονται αναλόγως των χαρακτηριστικών της περιοχής όπως έδαφος, συνθήκες κτλ και τη συμπεριφορά των πουλιών μετά από κάθε προσομοίωση. Τα πουλιά πριν από κάθε διασκορπισμό, «συμβουλεύονται» τον πίνακα για να επιλέξουν τη διαδρομή - μονοπάτι με το μικρότερο κόστος για να ακολουθήσουν και να φτάσουν στον προορισμό τους.

Επιπλέον, στον Αλγόριθμο Εύρεσης Ταμπού, υπάρχει και ένα άλλο είδος λίστας στην οποία αποθηκεύονται οι διαφορές της τρέχουσας λύσης με αυτών που εξετάστηκαν

και απορρίφθηκαν, πράγμα που δε συναντούμε στον αλγόριθμο που χρησιμοποιούν τα πουλιά για το διασκορπισμό τους.

Τέλος, όσο αφορά τις τεχνικές δυναμικής κατανομής φορτίου που χρησιμοποιούνται σε παράλληλο προγραμματισμό, συγκρίνοντάς τις με τη συμπεριφορά των πουλιών μπορούμε να πούμε ότι χρησιμοποιούν σε γενικές γραμμές το ίδιο μοντέλο. Διαφέρουν μόνο στην τέταρτη φάση, αυτή της επιλογής καταλληλότερης στρατηγικής για κάθε διεργασία. Στον παράλληλο προγραμματισμό οι επεξεργαστές είναι αυτοί που θα επιλέξουν τις καταλληλότερες διεργασίες και θα τις αποστείλουν στον κατάλληλο προορισμό, ενώ όσον αφορά τα πουλιά – διεργασίες, αυτά είναι που επιλέγουν την καταλληλότερη περιοχή διασκορπισμού.

Κεφάλαιο 4

Εξισορρόπηση Φορτίου με Ικανότητα Εξυπηρέτησης

4.1 Εισαγωγή – Το πρόβλημα

Σε αυτή τη φάση της Διπλωματικής Εργασίας ασχολήθηκα με τη μελέτη του προβλήματος εξισορρόπησης φορτίου σε διαφορετικά δίκτυα με κόμβους εξυπηρέτησης χρηστών.

Κάθε δίκτυο ήταν ένας γράφος με n μηχανές - κόμβους εξυπηρέτησης και m εργασίες - χρήστες που απαιτούν να εξυπηρετηθούν. Ισχύει ότι m>n. Κάθε κόμβος έχει κάποια χωρητικότητα, capacity, C_i ενώ κάθε χρήστης έχει διαφορετικό φόρτο εργασίας , load, W^j. Θεωρώ επίσης ότι κάθε χρήστης συμπεριφέρεται εγωιστικά, δηλαδή έχει ως στόχο να βελτιστοποιήσει τη δική του κατάσταση και όχι το δίκτυο στο σύνολο. Κάθε κόμβος θεωρείται υπερφορτωμένος εάν ο συνολικός φόρτος εργασίας του τη δεδομένη χρονική στιγμή ξεπερνά τη χωρητικότητα του. Τέλος, πρέπει να αναφερθεί ότι το δίκτυο μας είναι ένας γράφος και η γειτνίαση προκύπτει μέσω αυτού. Επομένως γείτονας θεωρείται ο κόμβος που συνδέεται άμεσα με τον κόμβο εξυπηρέτησης. Για κάθε κόμβο i γράφουμε neigh(i) για το σύνολο με τους γείτονες του i στον γράφο. Στόχος του προβλήματος με το οποίο ασχολούμαστε είναι

η εύρεση μιας ανάθεσης των χρηστών στους κόμβους του δικτύου έτσι ώστε κανένας κόμβος να μην είναι βαρυφορτωμένος.
Θα προτείνουμε δύο αλγόριθμους για επίλυση αυτού του προβλήματος: Ο πρώτος αλγόριθμος αποτελεί παραλλαγή του αλγόριθμου του [] στον οποίο εισάγεται η έννοιας της χωρητικότητας των κόμβων ενώ ο δεύτερος αλγόριθμος είναι εμπνευσμένος από τον αλγόριθμο μετανάστευσης των πουλιών που μελετήσαμε στο προηγούμενο κεφάλαιο.

4.2 Παραλλαγή Αλγορίθμου The protocol with "neutral moves" allowed – Αλγόριθμος 1 [2]

Όπως έχουμε αναφέρει, ο πρώτος αλγόριθμος που θα μελετήσουμε, και τον οποίο αποκαλούμε Αλγόριθμο 1, αποτελεί παραλλαγή του αλγορίθμου που παρουσιάζεται στο [2].

Ο Αλγόριθμος, θεωρεί μια αρχική ανάθεση των χρηστών στους κόμβους του δικτύου και ακολουθεί επαναληπτική διαδικασία έως ότου κανένας από τους χρήστες δεν επιθυμεί να μεταναστεύσει..

Στον Αλγόριθμο 1 εισάγεται η έννοια της χωρητικότητας, η οποία δε φαίνεται στον πρωτότυπο αλγόριθμο του άρθρου. Ένας χρήστης, θεωρούμε ότι επιθυμεί να μεταναστεύσει εάν ο γείτονας που επέλεξε τυχαία μπορεί να τον εξυπηρετήσει καλύτερα από αυτόν που ήδη βρίσκεται και αν ισχύει ότι ο φόρτος εργασίας του κόμβου που βρίσκεται ξεπερνά το μέγιστο που μπορεί να αντέξει, τη χωρητικότητα του.

For all nodes *i* let
  L_i = load on node *i*
While there exists node *i*with $L_i > C_i$ **do**
  for each user *j* do in parallel
    let *i* be the current node of user *j*
    choose neighbor *k* with probability $\frac{c_k}{\sum_{l \in neigh(j)} c_l}$
    If $\left(L_k + W^j < L_i\right)$ and $(L_i > C_i)$ **then**
      with probability $1 - \frac{L_k}{L_i}$
      Move user *j* from node *i* to *k*
      and set
        $L_k = L_k + W^j$
        $L_i = L_i - W^j$

Η παραπάνω διαδικασία αποτελεί μια φάση επαναληπτικής δομής, στην οποία κάθε χρήστης j, επιλέγει ομοιόμορφα, τυχαίο γείτονα k που προτίθεται να μεταναστεύσει. Στην συνέχεια ελέγχει την συνθήκη μετανάστευσης, εάν δηλαδή ο φόρτος εργασίας του γείτονα k που επέλεξε είναι λιγότερος από το φόρτο εργασίας του κόμβου i που τον εξυπηρετεί τη δεδομένη χρονική στιγμή t και εάν ο φόρτος εργασίας του κόμβου i που τον εξυπηρετεί ξεπερνά τον μέγιστο που μπορεί να αντέξει, τη χωρητικότητα του. Σε περίπτωση που η συνθήκη ισχύει, τότε ελέγχει την πιθανότητα μετανάστευσης $(1 - L_k^t / L_i^{j(t)})$ και αν αυτή ισχύει τότε μεταναστεύει. Η επανάληψη σταματά όταν φτάσουμε σε φάση όπου κανένας από τους χρήστες δεν μεταναστεύει σε γείτονα.

4.3 Αλγόριθμος Εμπνευσμένος από τη Μετανάστευση των Πουλιών Αλγόριθμος 2 [6]

Ο Αλγόριθμος 2, ο οποίος αποτελεί παραλλαγή του πρώτου, είναι εμπνευσμένος από τη διαδικασία που ακολουθούν τα πουλιά για να μεταναστεύσουν όπως παρουσιάζεται στο άρθρο «Uncertainly in Spatially Explicit Population Models» [6]. Σε αρχική φάση οι χρήστες ανατίθενται στους κόμβους εξυπηρέτησης ομοιόμορφα τυχαία. Στη συνέχεια, ακολουθεί επαναληπτική διαδικασία έως ότου κανένας από τους κόμβους δεν είναι υπερφορτωμένος, πράγμα που σημαίνει ότι ο συνολικός φόρτος των χρηστών που εξυπηρετεί δε ξεπερνά τη χωρητικότητα του.

For all nodes i let
L_i = load on node i
While there exists node i with $L_i > C_i$ **do**
For each user j do in parallel
Let i be the current node of user j
If $L_i > C_i$ choose to disperse with probability p_1
If chose to disperse
Move user j from node i to k with probability $p_2 = \frac{c_k}{\sum_{l \in neigh(j)} c_l}$
and set
$L_k = L_k + W^j$
$L_i = L_i - W^j$

Ο Αλγόριθμος 2 όπως παρουσιάζεται παραπάνω, εκτελεί μιαν επαναληπτική δομή, μέχρι το σύστημα να επέλθει σε ισορροπία, δηλαδή όλοι οι κόμβοι του δικτύου να έχουν φόρτο εργασίας που να μην ξεπερνά το μέγιστο που μπορούν να ανεχτούν, τη χωρητικότητα τους.

Πρωτίστως, αρχικοποιούνται η χωρητικότητα των κόμβων, $C_i \ \forall \ \text{i} \in n$, ο φόρτος εργασίας των χρηστών, $W^j \ \forall \ j \in m$, και κάθε χρήστης επιλέγει τυχαία κόμβο εξυπηρέτησης.

Ακολουθεί μια επαναληπτική δομή έως ότου το σύστημα βρεθεί σε κατάσταση ισορροπίας όπου κανένας κόμβος δε θα έχει φόρτο εργασίας μεγαλύτερο από το μέγιστο που μπορεί αν ανεχτεί, τη χωρητικότητα του. Σε κάθε επανάληψη, υπολογίζεται ο φόρτος εργασίας κάθε κόμβου στο δίκτυο που ισούται με το άθροισμα του φόρτου εργασίας όλων των κόμβων που εξυπηρετεί τη δεδομένη χρονική στιγμή, L_i^t . Εάν $L_i^t > C_i$, τότε με πιθανότητα p1, οι χρήστες του κόμβου i θα επιχειρήσουν να μεταναστεύσουν. Σε περίπτωση επιλογής προς μετανάστευση, κάθε κόμβος $j \in i \ C_i$ επιλέγει υποψήφιο γείτονα προς μετανάστευση και μεταναστεύει σε αυτόν με πιθανότητα p2= $\frac{\sum_{i \in n} C_i}{\sum_{j \in m} W^j}$.

Κεφάλαιο 5

Πειραματική Αξιολόγηση Αλγορίθμων

5.1 Υλοποιήσεις

Για την υλοποίηση του προβλήματος και την δημιουργία προσομοιώσεων προς εξεύρεση συμπερασμάτων, δημιουργήθηκαν οι παρακάτω αλγόριθμοι όπως έχουν περιγραφεί και παρουσιαστεί στο Κεφάλαιο 4.

Αλγόριθμος 1 – Παραλλαγή Αλγορίθμου [2]

For all nodes *i* let
  L_i = load on node *i*
While there exists node *i*with $L_i > C_i$ **do**
  for each user *j* do in parallel
    let *i* be the current node of user *j*
    choose neighbor *k* with probability $\frac{c_k}{\sum_{l \in neigh(j)} c_l}$
    If $\left(L_k + W^j < L_i\right)$ and $(L_i > C_i)$ **then**
      with probability $1 - \frac{L_k}{L_i}$
      Move user *j* from node *i* to *k*
      and set
        $L_k = L_k + W^j$
        $L_i = L_i - W^j$

Αλγόριθμος 2 – Αλγόριθμος εμπνευσμένος από τη μετανάστευση των πουλιών [6]

For all nodes *i* let
L_i = load on node *i*
While there exists node *i* with $L_i > C_i$ **do**
For each user *j* do in parallel
Let *i* be the current node of user *j*
If $L_i > C_i$ choose to disperse with probability p_1
If chose to disperse
Move user *j* from node *i* to *k* with probability $p_2 = \frac{c_k}{\sum_{l \in neigh(j)} c_l}$
and set
$L_k = L_k + W^j$
$L_i = L_i - W^j$

5.1.3 Περιγραφή Δομών Αλγορίθμων

Για την υλοποίηση των δύο αλγορίθμων δημιουργήθηκαν τρεις βασικές δομές, όπως παρουσιάζονται παρακάτω:

```
typedef struct _Nodetype {
    int node_id;              //identity of node
    int num_of_users;         //number of users that node is
                               serving
    int capacity;
    int current_load;
}Nodetype;
```

Η δομή του κόμβου `_Nodetype`, αποτελείται από το `node_id`, που αποτελεί τον μοναδικό αριθμό του κόμβου εξυπηρέτησης, το `num_of_users` , που αντιστοιχεί με τον αριθμό των χρηστών που εξυπηρετεί μια δεδομένη χρονική στιγμή, το `capacity` που αποτελεί τη χωρητικότητα του και το `current_load` που δείχνει το φόρτο εργασίας του μια δεδομένη χρονική στιγμή.

```
typedef struct _Usertype {
    int num;                //number_of_user
    int x;                  //position x
```

```
    int y;                  //position y
    int load;               //weight of task
    int node_id;            //which node belongs to
}Usertype;
```

Η δομή του χρήστη `_Usertype`, αποτελείται από το `num` που αποτελεί τον αύξον αριθμό του χρήστη, τα `x` και `y` που μας δείχνουν την τοποθεσία του στο δίκτυο, το `load` που δείχνει το φόρτο εργασίας που απαιτεί για να εξυπηρετηθεί από έναν κόμβο και το `node_id` που φανερώνει τον κόμβο στον οποίο ανήκει μια δεδομένη χρονική στιγμή.

```
typedef struct _MatrixGraph {
     int **matrix;              // The adjacency matrix
     int numOfVertices;         // The number of nodes in graph
     int *visited;              // Indicated if  node was
                                visited during a search or not
     Nodetype *values;          // The values for each node
}MatrixGraph;
```

Η δομή του γράφου `_MatrixGraph`, αποτελείται από τον πίνακα γειτνίασης, `**matrix`, που φανερώνει τη διασύνδεση των κόμβων του δικτύου μεταξύ τους, το `numOfVertices`, που δείχνει τον αριθμό των κόμβων του δικτύου, τον πίνακα `*visited` που μας δείχνει ποιούς κόμβους του δικτύου έχουν επισκεφτεί χρήστες, και τον πίνακα των κόμβων εξυπηρέτησης, `*values`.

Το πρόγραμμα υλοποιήθηκε σε γλώσσα Προγραμματισμού C και πλατφόρμα CodeBlocks.

5.2 Σύγκριση Αλγορίθμων 1 και 2

Η διαφορά τον δύο αλγορίθμων βρίσκεται κυρίως στο σημείο απόφασης προς μετανάστευση.

Στον Αλγόριθμο 1, την παραλλαγή του αλγορίθμου στο άρθρο των Petra Berenbrink, Tom Friedetzky, Leslie Ann Goldberg, Paul W. Goldberg, Zengjian Hu, Russell A. Martin, [2] ο χρήστης αποφασίζει να μεταναστεύσει με κάποια

πιθανότητα εάν ο φόρτος εργασίας του κόμβου που τον εξυπηρετεί είναι μεγαλύτερος από αυτόν του τυχαίου υποψήφιου γείτονα που επέλεξε προς μετανάστευση και ισχύει ότι ο φόρτος εργασίας του κόμβου που τον εξυπηρετεί ξεπερνά το μέγιστο που μπορεί να αντέξει, τη χωρητικότητα του, $\left(L_k^t < L_i^{j(t)}\right) and \left(L_i^{j(t)} > C_i\right)$. Στον Αλγόριθμο 2, η περίπτωση μετανάστευσης εξετάζεται μόνο εάν ο ο φόρτος εργασίας του κόμβου που τον εξυπηρετεί είναι μεγαλύτερος από αυτόν του τυχαίου υποψήφιου γείτονα που επέλεξε προς μετανάστευση, $L_i^t > C_i$

Επιπλέον, στον Αλγόριθμο 1, η επαναληπτική δομή αφορά τους χρήστες και ο έλεγχος ικανοποίησης ή μη γίνεται για κάθε χρήστη. Για κάθε χρήστη εξετάζεται εάν στον κόμβο που βρίσκεται ικανοποιείται βάση του φόρτου εργασίας του κόμβου και τις δικές του ανάγκες. Σε αντίθετη περίπτωση επιλέγει να μεταναστεύσει με κάποια πιθανότητα. Στον Αλγόριθμο 2, η επαναληπτική δομή αφορά τους κόμβους και σε κάθε επανάληψη, ελέγχεται για κάθε κόμβο κατά πόσο είναι ή όχι υπερφορτωμένος. Για κάθε κόμβο εξετάζεται εάν είναι υπερφορτωμένος, δηλαδή εάν ο συνολικός φόρτος εργασίας των χρηστών που εξυπηρετεί είναι μεγαλύτερος από τη χωρητικότητα του. Σε μια τέτοια περίπτωση, όλοι οι χρήστες που εξυπηρετεί μεταναστεύουν σε γειτονικούς με κάποια πιθανότητα.

Στα δίκτυα που υλοποιήθηκαν θεωρώ ότι:
→Όλοι οι χρήστες έχουν ίση προτεραιότητα εξυπηρέτησης.
→Το δίκτυο είναι διανεμημένο και ταυτόχρονο. Κάθε χρονική στιγμή πολλοί χρήστες μπορούν να εξετάζουν περίπτωση διασκορπισμού.
→Στο δίκτυο δεν υπάρχει κεντρικός μηχανισμός ελέγχου (centralized control mechanism)

5.2.1 Σενάρια

Οι αλγόριθμοι χρησιμοποιήθηκαν σε δίκτυα με τα εξής χαρακτηριστικά:

1. 10 χρήστες σε δίκτυο με 6 κόμβους εξυπηρέτησης.
2. 15 χρήστες σε δίκτυο με 6 κόμβους εξυπηρέτησης.
3. 40 χρήστες σε δίκτυο με 15 κόμβους εξυπηρέτησης.
4. 100 χρήστες σε δίκτυο με 15 κόμβους εξυπηρέτησης.
5. 200 χρήστες σε δίκτυο με 15 κόμβους εξυπηρέτησης.

Στις προσομοιώσεις που ακολούθησαν φαίνεται η σύγκριση του χρόνου σύγκλισης που απαιτείται έως ότου το σύστημα φτάσει σε κατάσταση ισορροπίας. Ο χρόνος σύγκλισης εκφράζεται σε αριθμό επαναλήψεων που χρειάζεται ο κάθε αλγόριθμος για να επιφέρει ισορροπία στο δίκτυο.

Στους παρακάτω πίνακες παρουσιάζονται τα αποτελέσματα από τις προσομοιώσεις που πάρθηκαν με τη χρήση των αλγορίθμων 1 και 2 για κάθε είδος δικτύου [1-5.] Για κάθε ένα από τα δίκτυα, οι αλγόριθμοι έτρεξαν για διαφορετικά σενάρια, με δίκτυα βαρυφορτωμένα και μη. Βαρυφορτωμένο θεωρείται ένα δίκτυο στο οποίο το άθροισμα του φόρτου εργασίας όλων των χρηστών, ξεπερνά το άθροισμα της χωρητικότητας όλων των κόμβων του δικτύου, $\sum_{i\in n} C_i > \sum_{j\in m} W^j$ Επομένως, $\frac{\sum_{i\in n} C_i}{\sum_{j\in m} W^j} < 1$. Σε κάθε σενάριο για κάθε είδος δικτύου [1-5], οι κόμβοι είχαν διαφορετική χωρητικότητα C_i και οι χρήστες διαφορετικό φόρτο εργασίας W^j.

$\sum_{i\in n} C_i$	$\sum_{j\in m} W^j$	$\frac{\sum_{i\in n} C_i}{\sum_{j\in m} W^j}$	Επαναλήψεις Αλγόριθμος 1	Επαναλήψεις Αλγόριθμος 2
120	150	0.8	>4000	>4000
120	130	0.92	>4000	>4000
180	170	1.06	>4000	>4000
180	150	1.2	6	3
180	130	1.3	6	3
240	130	1.85	5	1

Πίνακας 4.1 Σενάριο 6 κόμβοι – 10 χρήστες

$\sum_{i\in n} C_i$	$\sum_{j\in m} W^j$	$\frac{\sum_{i\in n} C_i}{\sum_{j\in m} W^j}$	Αλγόριθμος 1	Αλγόριθμος 2
150	150	1	>4000	>4000
150	130	1.15	>4000	>4000
225	170	1.32	7	3
300	150	2	19	2
300	130	2.3	18	2
300	130	2.3	11	1

Πίνακας 4.2 Σενάριο 15 κόμβοι – 10 χρήστες

$\sum_{i\in n} C_i$	$\sum_{j\in m} W^j$	$\frac{\sum_{i\in n} C_i}{\sum_{j\in m} W^j}$	Επαναλήψεις Αλγόριθμος 1	Επαναλήψεις Αλγόριθμος 2
300	300	1	1532	103
300	250	1.2	48	7
450	350	1.3	430	50
450	300	1.5	24	2
600	350	1.71	9	1
450	250	1.8	17	1

Πίνακας 4.3 Σενάριο 15 κόμβοι – 20 χρήστες

$\sum_{i\in n} C_i$	$\sum_{j\in m} W^j$	$\frac{\sum_{i\in n} C_i}{\sum_{j\in m} W^j}$	Επαναλήψεις Αλγόριθμος 1	Επαναλήψεις Αλγόριθμος 2
300	400	0.75	>4000	>4000
450	500	0.9	>4000	>4000
450	450	1	400	25
450	400	1.125	51	13
600	500	1.2	28	9
600	450	1.33	22	3

Πίνακας 4.4 Σενάριο 15 κόμβοι – 30 χρήστες

$\sum_{i\in n} C_i$	$\sum_{j\in m} W^j$	$\frac{\sum_{i\in n} C_i}{\sum_{j\in m} W^j}$	Επαναλήψεις Αλγόριθμος 1	Επαναλήψεις Αλγόριθμος 2
600	600	1	>4000	53
750	700	1.07	35	25
600	500	1.2	39	5
750	600	1.25	30	5
900	700	1.3	31	4
750	500	1.5	15	1

Πίνακας 4.5 Σενάριο 15 κόμβοι – 40 χρήστες

$\sum_{i\in n} C_i$	$\sum_{j\in m} W^j$	$\frac{\sum_{i\in n} C_i}{\sum_{j\in m} W^j}$	Επαναλήψεις Αλγόριθμος 1	Επαναλήψεις Αλγόριθμος 2
1800	1800	1	>4000	75
1500	1500	1	>4000	63
1200	1200	1	68	35
1950	1800	1.08	44	8
1650	1500	1.1	26	5
1350	1200	1.12	19	4
2100	1800	1.16	17	2
1500	1200	1.25	19	1

Πίνακας 4.6 Σενάριο 15 κόμβοι – 100 χρήστες

$\sum_{i\in n} C_i$	$\sum_{j\in m} W^j$	$\frac{\sum_{i\in n} C_i}{\sum_{j\in m} W^j}$	Επαναλήψεις Αλγόριθμος 1	Επαναλήψεις Αλγόριθμος 2
2250	3500	0.64	>4000	>4000
2700	3500	0.77	>4000	>4000
2850	3500	0.81	>4000	>4000
3000	3500	0.85	>4000	>4000
3135	3500	0.9	>4000	>4000
3000	3000	1	>4000	56
3900	3900	1	>4000	>4000
3600	3500	1.02	350	12
4050	3900	1.03	81	11
3150	3000	1.05	57	15
4200	3900	1.07	41	6
3300	3000	1.1	41	4
4350	3900	1.15	41	4

Πίνακας 4.7 Σενάριο 15 κόμβοι – 200 χρήστες

5.2.2 Σχολιασμός Αποτελεσμάτων - Συμπεράσματα

Σε γενικές γραμμές ο Αλγόριθμος 2 χρειάζεται λιγότερες επαναλήψεις έως ότου το σύστημα φτάσει σε κατάσταση ισορροπίας. Επομένως αποδεικνύεται ότι είναι αποδοτικότερος από τον Αλγόριθμο 1, ο οποίος αποτελεί παραλλαγή του αλγορίθμου στο άρθρο [2]

Σημαντικό είναι το ότι σε οριακές συνθήκες, όπου η ολική χωρητικότητα των κόμβων ισούται με τον ολικό φόρτο εργασίας των χρηστών, $\frac{\sum_{i\in n} C_i}{\sum_{j\in m} W^j} = 1$, ο Αλγόριθμος 2 φτάνει σε ισορροπία κάτι που αδυνατεί να πράξει ο Αλγόριθμος 1.

Επιπλέον, σε περιπτώσεις με ίδιο λ, επομένως ίδιο λόγο χωρητικότητας προς φορτο εργασίας παρατηρούμε ότι έχουμε μεγαλύτερο χρόνο σύγκλισης με περισσότερους χρήστες. [Βλ. Πίνακας 4.6 και Πίνακας 4.7, όπου λ=1]

5.2.4 Ενδιαφέροντα Σημεία

Σημαντικό είναι το ότι σε οριακές περιπτώσεις, όπου ολική χωρητικότητα των κόμβων είναι μικρότερη από τον ολικό φόρτο εργασίας των χρηστών, $\frac{\sum_{i\in n} C_i}{\sum_{j\in m} W^j} < 1$, ο αλγόριθμος που υλοποιήθηκε κατορθώνει να βρει τρόπο εξυπηρέτησης των χρηστών, κάτι που υστερεί ο Αλγόριθμος 1. Αυτό σημαίνει ότι κάτω από οποιεσδήποτε συνθήκες, εάν υπάρχει δυνατότητα εξυπηρέτησης των χρηστών βάση των προδιαγραφών του συστήματος, την τροφοδοσία, ο Αλγόριθμος 2 θα βρει τον τρόπο να εξυπηρετήσει όλους τους χρήστες.

5.3 Πειραματική Αξιολόγηση Αλγόριθμου 2

Στη συνέχεια, εμπνευσμένη από το άρθρο «*Balls into Non-uniform Bins*» των Petra Berenbrink, Andre Brinkmann, Tom Friedetzky και Las Nagel [7] αποφάσισα να δημιουργήσω παρόμοια σενάρια και να αντιπαραβάλω τα αποτελέσματα που θα προέκυπταν από τις προσομοιώσεις με αυτά που αναγράφονται στο άρθρο.

Στη δική μου περίπτωση, ήθελα να μελετήσω το χρόνο σύγκλισης, τον αριθμό επαναλήψεων μέχρι το σύστημα να επέλθει σε ισορροπία, σε δίκτυα με διαφορετικά χαρακτηριστικά.

Για τη μελέτη διαφορετικών τοπολογιών δικτύου δημιουργήθηκαν τα εξής δίκτυα:

1. Δίκτυο 100, 49 και 25 κόμβων σε square topology
2. Δίκτυο 101, 31 και 60 κόμβων σε star topology

5.3.1 Παράμετροι αλλαγής

Οι παράμετροι που διαφοροποιούνταν σε κάθε προσομοίωση είναι οι εξής:

- ➔ Αριθμός χρηστών δικτύου
- ➔ Αριθμός κόμβων εξυπηρέτησης
- ➔ Χωρητικότητα κόμβων εξυπηρέτησης (ίδια για όλους τους κόμβους του δικτύου)
- ➔ Πιθανότητα μετανάστευσης ή όχι χρηστών
- ➔ Τοπολογία δικτύου (square topology/star topology).
- ➔ Ίδια ή διπλάσια πιθανότητα αρχικής τοποθέτησης σε επιλεγμένους κεντρικούς κόμβους εξυπηρέτησης .

5.3.2 Περιγραφή Σεναρίων

Στη συνέχεια έγιναν προσομοιώσεις για τα παρακάτω σενάρια:

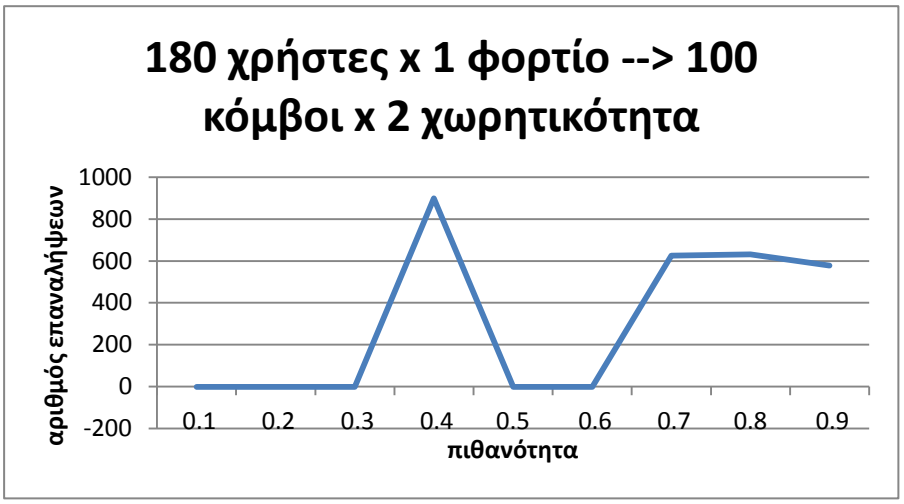
<u>Σενάριο Α:</u>

Όλοι οι κόμβοι να έχουν ίδια πιθανότητα τυχαίας αρχικής επιλογής από όλους τους χρήστες του δικτύου. Στο συγκεκριμένο Σενάριο μελετείται η συμπεριφορά κάθε τοπολογίας δικτύου, τοπολογία τετράγωνο και τοπολογία αστέρι, σε διαφορετικά μεγέθη δικτύων, 100, 49 και 25 για τοπολογία τετράγωνο και 101, 60 και 31 για τοπολογία αστέρι, και για διαφορετικό φόρτο εργασίας δικτύου, βαρυφορτωμένο και μη. Βαρυφορτωμένο θεωρείται το δίκτυο στο οποίο ο ολικός φόρτος εργασίας των χρηστών ξεπερνά την ολική χωρητικότητα των κόμβων του δικτύου. Ορίζουμε $\lambda = \frac{\sum_{i \in n} C_i}{\sum_{j \in m} W_j}$. Επομένως όσο μικρότερη είναι η τιμή του λ, τόσο βαρυφορτωμένο είναι το δίκτυο. Στις προσομοιώσεις που ακολουθούν για το Σενάριο 1 παρουσιάζεται η συμπεριφορά κάθε δικτύου για λ=1 και λ=2, βαρυφορτωμένο και μη βαρυφορτωμένο δίκτυο αντίστοιχα. Σ

Στις γραφικές παραστάσεις που ακολουθούν διευκρινίζεται ότι μεγάλος αριθμός επαναλήψεων, πέραν των 2000, σημειώνεται με μηδέν (0).

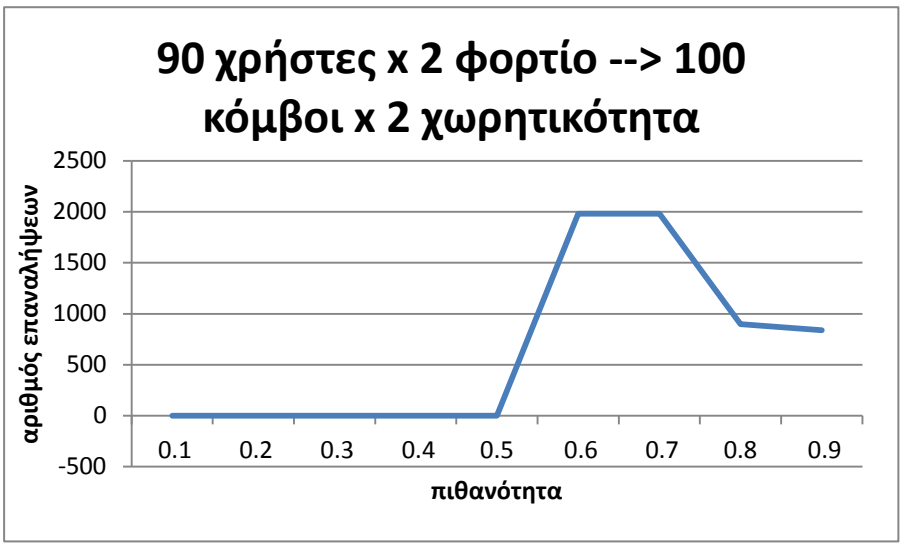
Α. Για $\lambda = \lambda = \frac{\sum_{i \in n} C_i}{\sum_{j \in m} W_j} = 1$

→ Τοπολογία τετράγωνο με 100 κόμβους



Σχήμα 5.1 Γραφική Παράσταση: 180 χρήστες x φόρτο 1
Δίκτυο τετράγωνο: 100 κόμβοι x χωρητικότητα 2

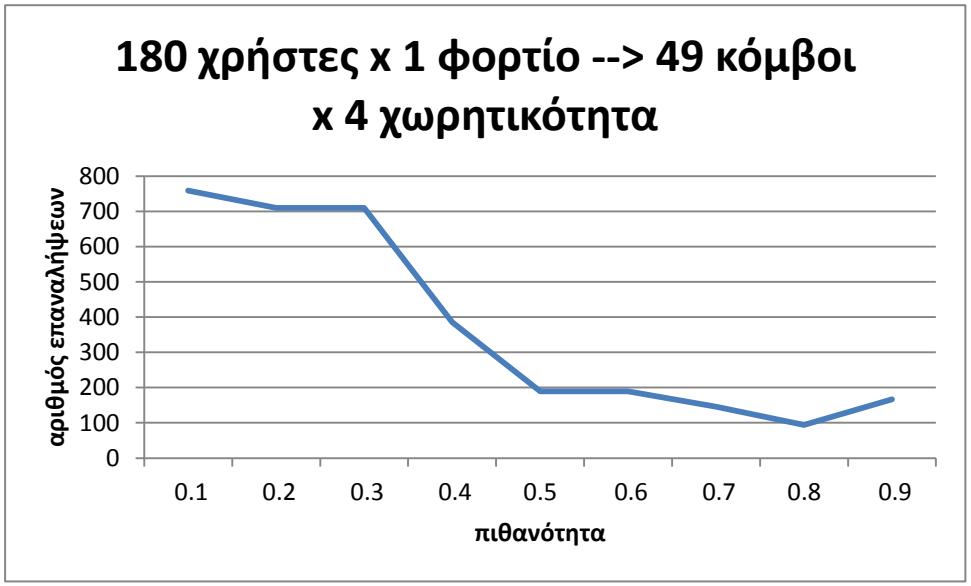
Στο σχήμα 5.1 παρατηρούμε ότι σε υπερφορτωμένο δίκτυο σε τοπολογία τετράγωνο με 100 κόμβους και 180 χρήστες με φορτίο 1, λιγότερες επαναλήψεις έχουμε με πιθανότητα μετανάστευσης p=0.9, και ακολουθούν p=0.8 και 0.7. Για πιθανότητες 0.1, 0.2, 0.3, 0.6 και 0.7 το σύστημα χρειάζεται μεγάλο αριθμό επαναλήψεων μέχρι να φτάσει σε ισορροπία, περισσότερες από 2000.



Σχήμα 5.2 Γραφική Παράσταση: 90 χρήστες x φόρτο 2
Δίκτυο τετράγωνο: 100 κόμβοι x χωρητικότητα 2

Στο σχήμα 5.2 παρατηρούμε ότι σε υπερφορτωμένο δίκτυο σε τοπολογία τετράγωνο με 100 κόμβους και 90 χρήστες με φορτίο 2, δηλαδή λιγότερους χρήστες με περισσότερο φορτίο, λιγότερες επαναλήψεις έχουμε με πιθανότητα μετανάστευσης p=0.9 και ακολουθούν p=0.8 και 0.7. Για πιθανότητες 0.1 – 0.5 το σύστημα χρειάζεται μεγάλο αριθμό επαναλήψεων μέχρι να φτάσει σε ισορροπία, περισσότερες από 2000.

→ Τοπολογία τετράγωνο με 49 κόμβους



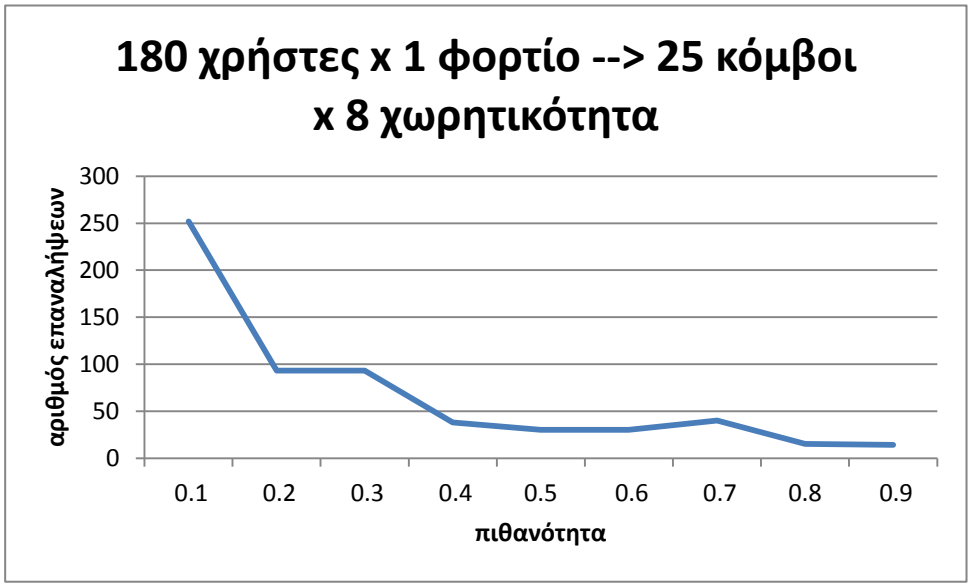
Σχήμα 5.3 Γραφική Παράσταση: 180 χρήστες x φόρτο 1
Δίκτυο τετράγωνο: 49 κόμβοι x χωρητικότητα 2

Στο σχήμα 5.3 παρατηρούμε ότι σε υπερφορτωμένο δίκτυο σε τοπολογία τετράγωνο με 49 κόμβους και 180 χρήστες με φορτίο 1, λιγότερες επαναλήψεις έχουμε με πιθανότητα μετανάστευσης p=0.8 και ακολουθούν p=0.9 και 0.7. Για πιθανότητες p=0.1 – 0.3 το σύστημα χρειάζεται περισσότερες επαναλήψεις, που είναι όμως λιγότερες από αυτές με ίδιο λ και μεγαλύτερο δίκτυο.

Σχήμα 5.4 Γραφική Παράσταση: 90 χρήστες x φόρτο 2
Δίκτυο τετράγωνο: 49 κόμβοι x χωρητικότητα 2

Στο σχήμα 5.4 παρατηρούμε ότι σε υπερφορτωμένο δίκτυο σε τοπολογία τετράγωνο με 49 κόμβους και 90 χρήστες με φορτίο 2, λιγότερες επαναλήψεις έχουμε με πιθανότητα μετανάστευσης p=0.9 και είναι πολύ λιγότερες από όλες τις προηγούμενες περιπτώσεις με ίδιο λ και διαφορετικά χαρακτηριστικά δικτύου. Ακολουθούν οι πιθανότητες p=0.4, 0.8, 0.7, 0.6, 0.5 ενώ για p=0.1 το σύστημα χρειάζεται μεγάλο αριθμό επαναλήψεων, περισσότερες από 2000.

→ Τοπολογία τετράγωνο με 25 κόμβους



Σχήμα 5.5 Γραφική Παράσταση: 180 χρήστες x φόρτο 1
Δίκτυο τετράγωνο: 25 κόμβοι x χωρητικότητα 8

Στο σχήμα 5.5 παρατηρούμε ότι σε υπερφορτωμένο δίκτυο σε τοπολογία τετράγωνο με 25 κόμβους και 180 χρήστες με φορτίο 1, λιγότερες επαναλήψεις έχουμε με πιθανότητα μετανάστευσης p=0.9 και είναι επίσης πολύ λιγότερες από όλες τις προηγούμενες περιπτώσεις με ίδιο λ και διαφορετικά χαρακτηριστικά δικτύου. Ακολουθούν οι πιθανότητες p=0.8, 0.7, 0.6, 0.5, 0.4 ενώ για p=0.1 το σύστημα χρειάζεται μόλις 250 επαναλήψεις.

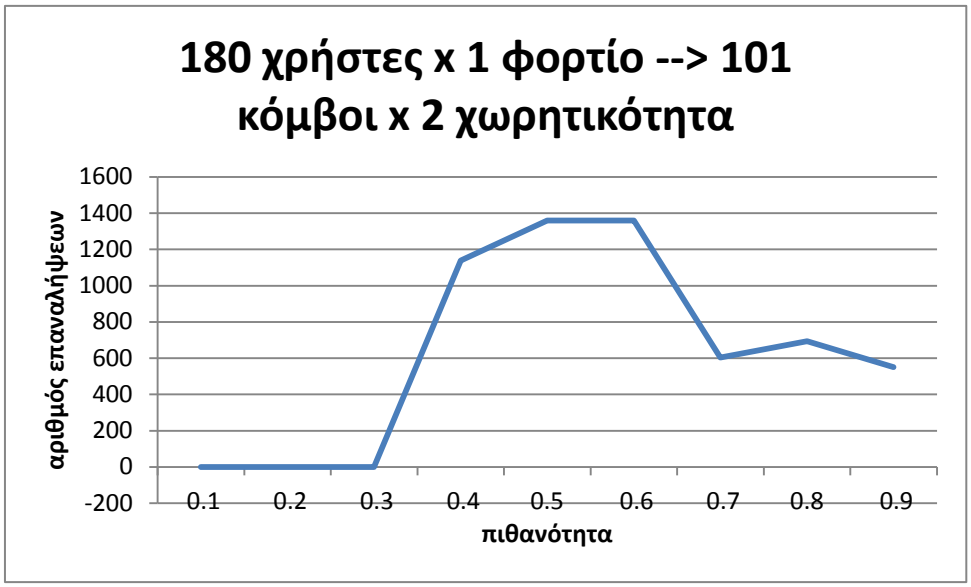
Επομένως σε τοπολογία αστέρι και μικρό δίκτυο, φτάνουμε σε ισορροπία και με πολλούς χρήστες, 180 με φορτίο 1, πράγμα που δε γίνεται σε μεγαλύτερα δίκτυα με 100 και 49 κόμβους, ακόμη και αν η ολική χωρητικότητα των κόμβων παραμένει η ίδια.

Σχήμα 5.6 Γραφική Παράσταση: 90 χρήστες x φόρτο 2
Δίκτυο τετράγωνο: 25 κόμβοι x χωρητικότητα 8

Στο σχήμα 5.6 παρατηρούμε ότι σε υπερφορτωμένο δίκτυο σε τοπολογία τετράγωνο με 25 κόμβους και 90 χρήστες με φορτίο 2, λιγότερες επαναλήψεις έχουμε με πιθανότητα μετανάστευσης p=0.9 και είναι επίσης πολύ λιγότερες από όλες τις προηγούμενες περιπτώσεις με ίδιο λ και διαφορετικά χαρακτηριστικά δικτύου. Ακολουθούν οι πιθανότητες p=0.8, 0.6, 0.5, 0.7, 0.4 ενώ για p=0.1 το σύστημα χρειάζεται μόλις 150 επαναλήψεις.

Επομένως για υπερφορτωμένα δίκτυα σε τοπολογία τετράγωνο, ένα μικρό δίκτυο με περισσότερη χωρητικότητα στους κόμβους φέρνει το σύστημα σε ισορροπία ταχύτερα ανεξάρτητα από το είδος τον χρηστών, 180 χρήστες με φορτίο 1 ή 90 χρήστες με φορτίο 2.

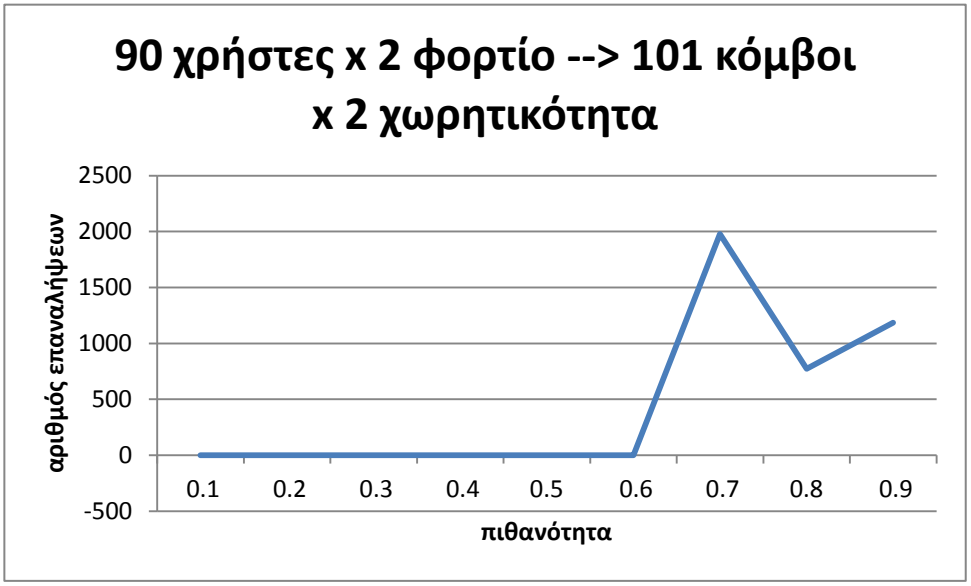
→ Τοπολογία αστέρι με 101 κόμβους



Σχήμα 5.7 Γραφική Παράσταση: 180 χρήστες x φόρτο 1
Δίκτυο αστέρι: 101 κόμβοι x χωρητικότητα 2

Στο σχήμα 5.7 παρατηρούμε ότι σε υπερφορτωμένο δίκτυο σε τοπολογία αστέρι με 101 κόμβους και 180 χρήστες με φορτίο 2, λιγότερες επαναλήψεις έχουμε με πιθανότητα μετανάστευσης p=0.9 και ακολουθούν p=0.7 και 0.8. Για πιθανότητες 0.1, 0.2, 0.3 το σύστημα χρειάζεται μεγάλο αριθμό επαναλήψεων μέχρι να φτάσει σε ισορροπία, περισσότερες από 2000.

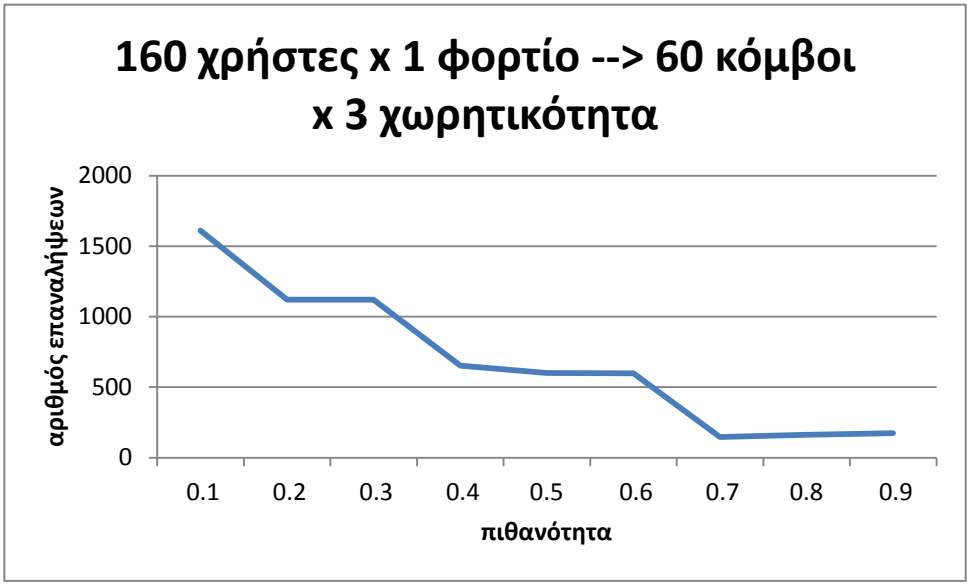
Συγκρίνοντάς το με αντίστοιχο δίκτυο σε τοπολογία τετράγωνο, παρατηρούμε ότι για λιγότερες πιθανότητες, p= 0.1 και 0.2, έναντι 0.1, 0.2, 0.3, 0.6 και 0.7 για τοπολογία τετράγωνο, το σύστημα χρειάζεται περισσότερες επαναλήψεις από 2000 για να φτάσει σε ισορροπία.



Σχήμα 5.8 Γραφική Παράσταση: 90 χρήστες x φόρτο 2
Δίκτυο αστέρι: 101 κόμβοι x χωρητικότητα 2

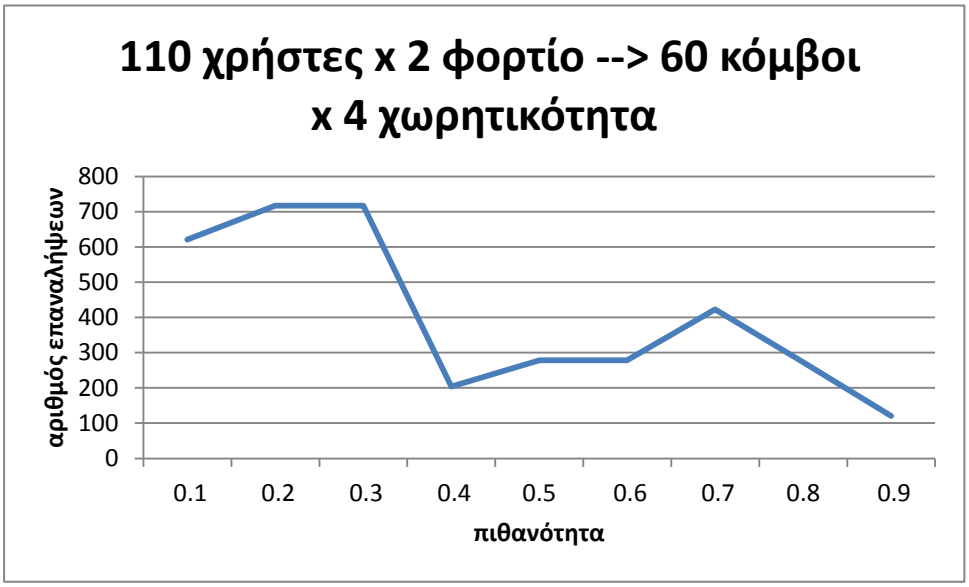
Στο σχήμα 5.8 παρατηρούμε ότι σε υπερφορτωμένο δίκτυο σε τοπολογία αστέρι με 101 κόμβους και 90 χρήστες με φορτίο 2, δηλαδή λιγότερους χρήστες με περισσότερο φορτίο, λιγότερες επαναλήψεις έχουμε με πιθανότητα μετανάστευσης p=0.8 και ακολουθούν p=0.9 και 0.7. Για πιθανότητες 0.1 – 0.6 το σύστημα χρειάζεται μεγάλο αριθμό επαναλήψεων μέχρι να φτάσει σε ισορροπία, περισσότερες από 2000.

→ Τοπολογία αστέρι με 60 κόμβους



Σχήμα 5.9 Γραφική Παράσταση: 160 χρήστες x φόρτο 1
Δίκτυο αστέρι: 60 κόμβοι x χωρητικότητα 3

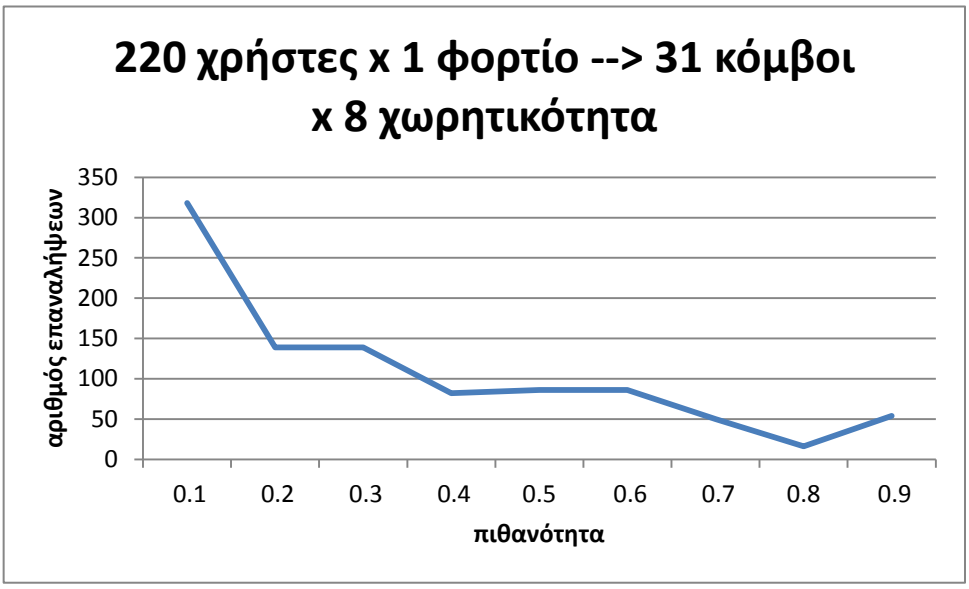
Στο σχήμα 5.9 παρατηρούμε ότι σε υπερφορτωμένο δίκτυο σε τοπολογία αστέρι με 60 κόμβους και 160 χρήστες με φορτίο 1, λιγότερες επαναλήψεις έχουμε με πιθανότητα μετανάστευσης p=0.9, 0.8 και 0.7. Ακολουθούν p=0.6, 0.5, 0.4, 0.3 και 0.2, 0.1.



Σχήμα 5.10 Γραφική Παράσταση: 110 χρήστες x φόρτο 2
Δίκτυο αστέρι: 60 κόμβοι x χωρητικότητα 4

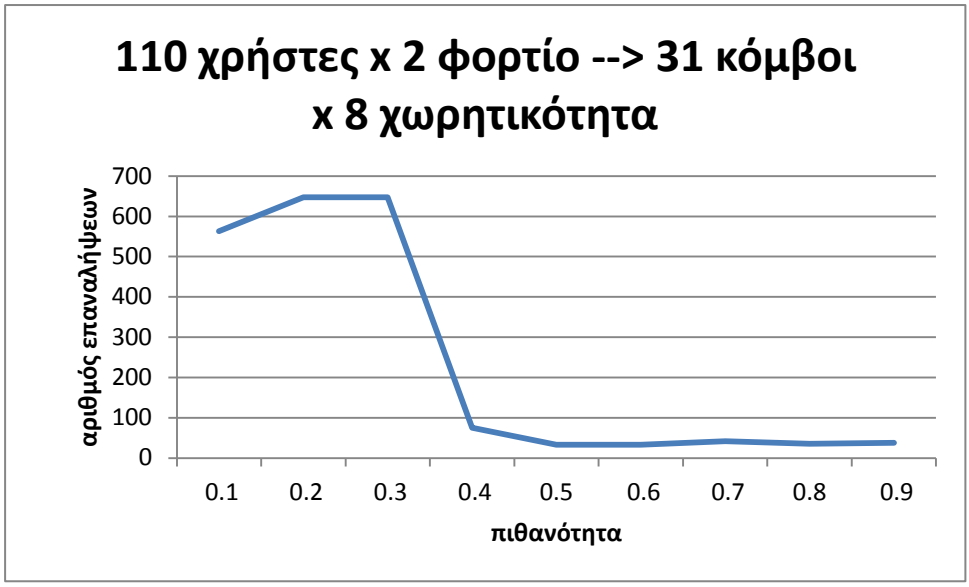
Στο σχήμα 5.10 παρατηρούμε ότι σε υπερφορτωμένο δίκτυο σε τοπολογία αστέρι, με 60 κόμβους και 110 χρήστες με φορτίο 2, λιγότερες επαναλήψεις έχουμε με πιθανότητα μετανάστευσης p=0.9. Για πιθανότητες p=0.1-0.3 το σύστημα χρειάζεται περισσότερες επαναλήψεις μέχρι να φτάσει σε ισορροπία που είναι όμως πολύ λιγότερες από τις επαναλήψεις που απαιτούνται στο αντίστοιχο σύστημα με τοπολογία τετράγωνο, 49 κόμβοι, χωρητικότητας 2, 90 χρήστες με φορτίο 2.

→ Τοπολογία αστέρι με 31 κόμβους



Σχήμα 5.11 Γραφική Παράσταση: 220 χρήστες x φόρτο 1
Δίκτυο αστέρι: 31 κόμβοι x χωρητικότητα 8

Στο σχήμα 5.11 παρατηρούμε ότι σε υπερφορτωμένο δίκτυο σε τοπολογία αστέρι με 31 κόμβους και 220 χρήστες με φορτίο 1, λιγότερες επαναλήψεις έχουμε με πιθανότητα μετανάστευσης p=0.8 και είναι πολύ λιγότερες από όλες τις προηγούμενες περιπτώσεις με ίδιο λ και διαφορετικά χαρακτηριστικά δικτύου. Ακολουθούν οι πιθανότητες p=0.9, 0.7, 0.6, 0.5, 0.4 ενώ για p=0.1 το σύστημα χρειάζεται μόλις 320 επαναλήψεις, που είναι μεν λιγότερες από τις μέγιστες των προηγούμενων περιπτώσεων με τοπολογία αστέρι, αλλά περισσότερες από το αντίστοιχο δίκτυο σε τοπολογία τετράγωνο, 49 κόμβοι, χωρητικότητας 2, 180 χρήστες με φορτίο 1.

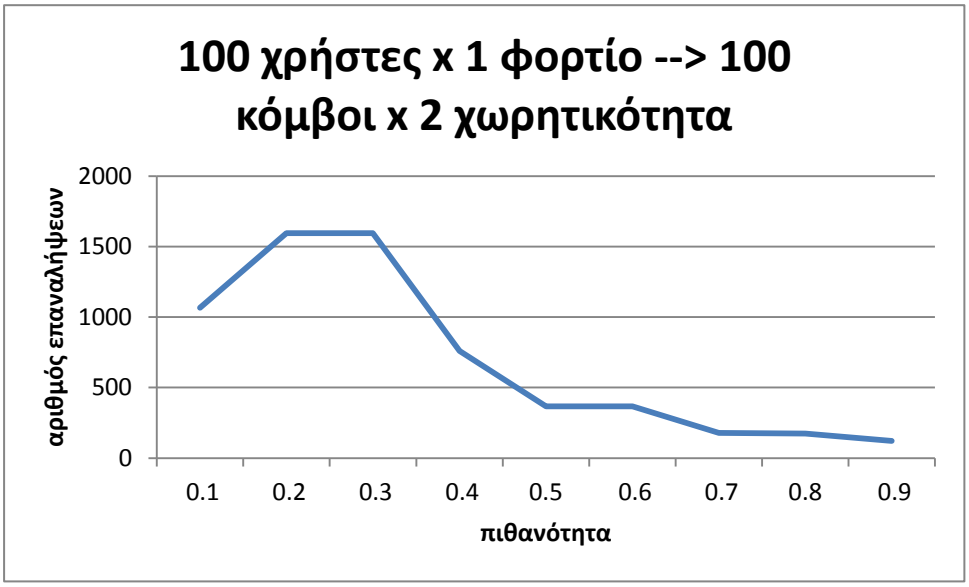


Σχήμα 5.12 Γραφική Παράσταση: 110 χρήστες x φόρτο 2
Δίκτυο αστέρι: 31 κόμβοι x χωρητικότητα 8

Στο σχήμα 5.12 παρατηρούμε ότι σε υπερφορτωμένο δίκτυο σε τοπολογία αστέρι με 31 κόμβους και 110 χρήστες με φορτίο 2, λιγότερες επαναλήψεις έχουμε με πιθανότητα μετανάστευσης p=0.9-0.5 και είναι πολύ λιγότερες από όλες τις προηγούμενες περιπτώσεις με ίδιο λ και διαφορετικά χαρακτηριστικά δικτύου. Για τις πιθανότητες που προκαλούν μεγάλο αριθμό επαναλήψεων οι επαναλήψεις είναι περισσότερες από αυτές των χείριστων περιπτώσεων σε αντίστοιχα δίκτυα τοπολογίας τετράγωνο 49 κόμβοι, χωρητικότητας 2, 90 χρήστες με φορτίο 2.

Β. Για $\lambda = \lambda = \frac{\sum_{i \in n} C_i}{\sum_{j \in m} W_j} = 2$

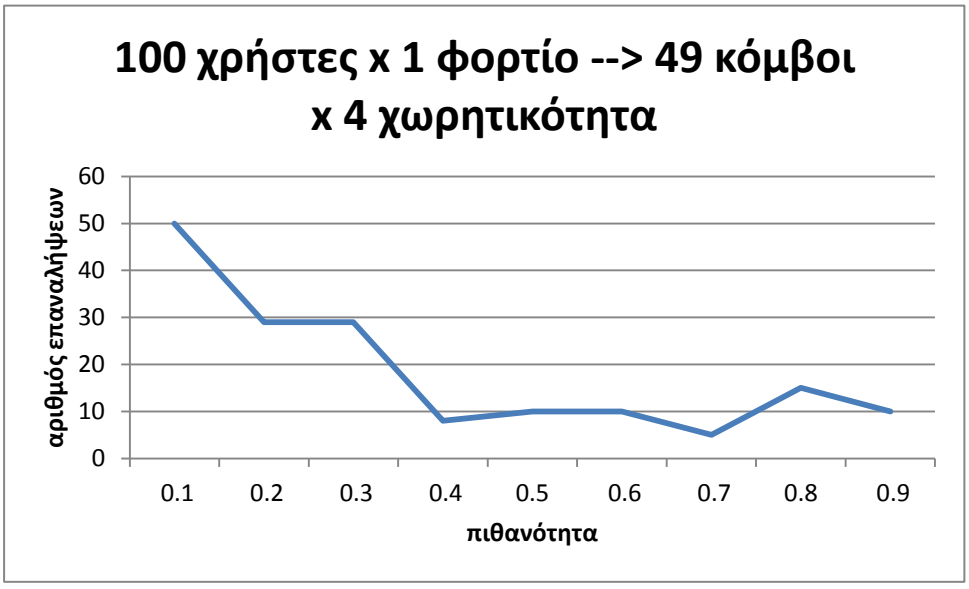
→ Τοπολογία τετράγωνο με 100 κόμβους



Σχήμα 5.13 Γραφική Παράσταση: 100 χρήστες x φόρτο 1
Δίκτυο τετράγωνο: 100 κόμβοι x χωρητικότητα 2

Στο σχήμα 5.13 παρατηρούμε ότι σε μη υπερφορτωμένο δίκτυο σε τοπολογία τετράγωνο με 100 κόμβους και 100 χρήστες με φορτίο 1, λιγότερες επαναλήψεις έχουμε με πιθανότητα μετανάστευσης p=0.9. Περισσότερες επαναλήψεις χρειάζονται για πιθανότητες p=0.2 και 0.3

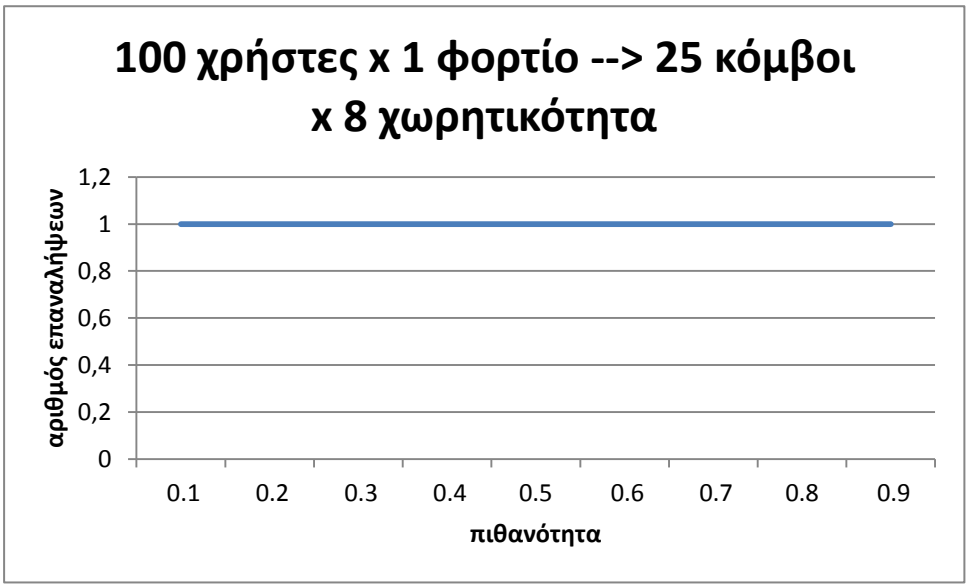
→ Τοπολογία τετράγωνο με 49 κόμβους



Σχήμα 5.14 Γραφική Παράσταση: 100 χρήστες x φόρτο 1
Δίκτυο τετράγωνο: 49 κόμβοι x χωρητικότητα 4

Στο σχήμα 5.14 παρατηρούμε ότι σε μη υπερφορτωμένο δίκτυο σε τοπολογία τετράγωνο με 49 κόμβους και 100 χρήστες με φορτίο 1, λιγότερες επαναλήψεις έχουμε με πιθανότητα μετανάστευσης p=0.7. Περισσότερες επαναλήψεις χρειάζονται για πιθανότητες p=0.1 αλλά είναι πολύ λιγότερες από τις επαναλήψεις με την ίδια πιθανότητα, ίδιο λ και ίδια τοπολογία σε μεγαλύτερο δίκτυο.

→ Τοπολογία τετράγωνο με 25 κόμβους



Σχήμα 5.15 Γραφική Παράσταση: 100 χρήστες x φόρτο 1
Δίκτυο τετράγωνο: 25 κόμβοι x χωρητικότητα 8

Στο σχήμα 5.15 παρατηρούμε ότι σε μη υπερφορτωμένο δίκτυο σε τοπολογία τετράγωνο με 25 κόμβους και 100 χρήστες με φορτίο 1 για κάθε πιθανότητα το σύστημα χρειάζεται μόνο μια επανάληψη μέχρι να φτάσει σε κατάσταση ισορροπίας.

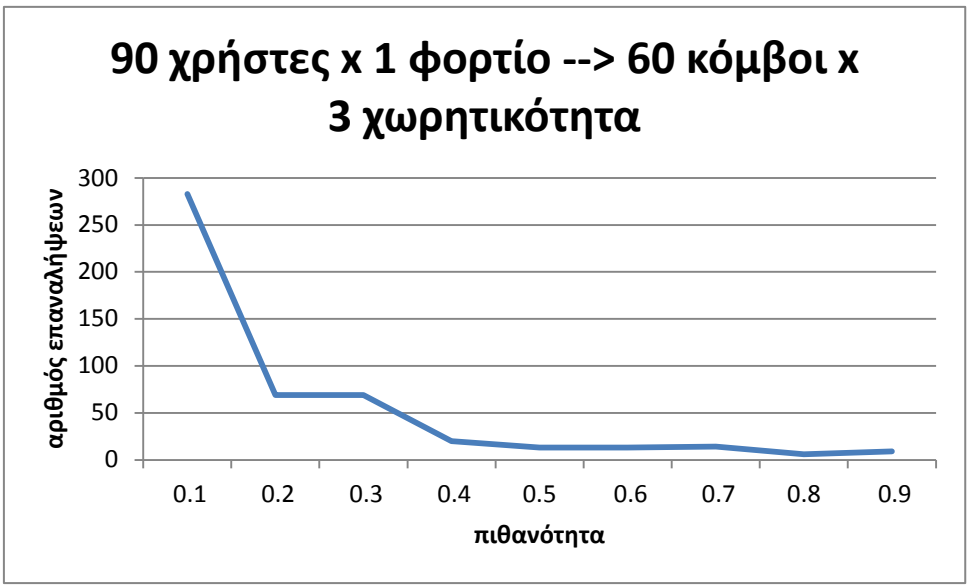
Επομένως όσο μικρότερο είναι το δίκτυο, τόσο γρηγορότερα φτάνουμε σε κατάσταση ισορροπίας στις περιπτώσεις που έχουμε μη υπερφορτωμένα δίκτυα.

→ Τοπολογία αστέρι με 101 κόμβους

Σχήμα 5.16 Γραφική Παράσταση: 100 χρήστες x φόρτο 1
Δίκτυο αστέρι: 101 κόμβοι x χωρητικότητα 2

Στο σχήμα 5.16 παρατηρούμε ότι σε μη υπερφορτωμένο δίκτυο σε τοπολογία αστέρι με 101 κόμβους και 100 χρήστες με φορτίο 1, λιγότερες επαναλήψεις έχουμε με πιθανότητα μετανάστευσης p=0.8. Περισσότερες επαναλήψεις χρειάζονται για πιθανότητες p=0.1.

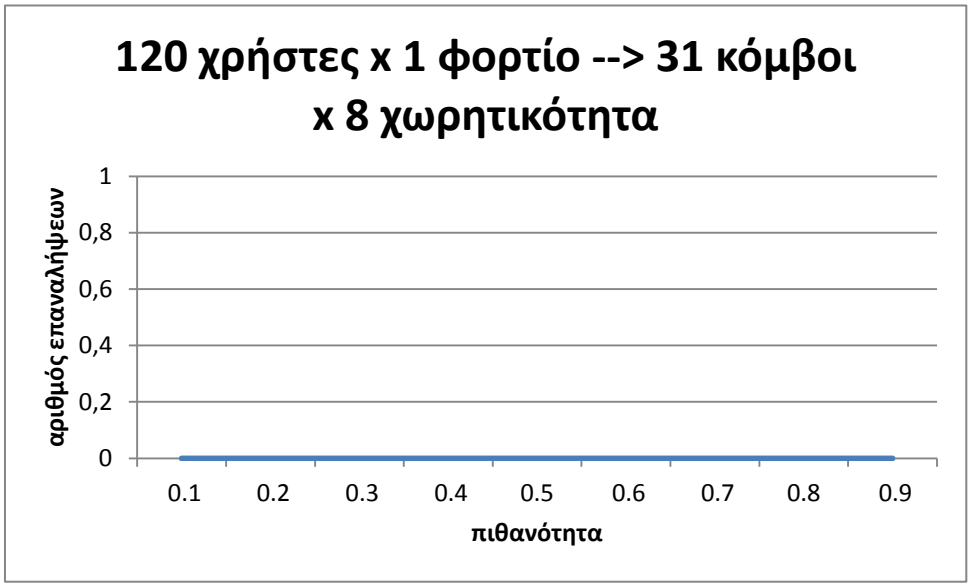
→ Τοπολογία αστέρι με 60 κόμβους



Σχήμα 5.17 Γραφική Παράσταση: 90 χρήστες x φόρτο 1
Δίκτυο αστέρι: 60 κόμβοι x χωρητικότητα 3

Στο σχήμα 5.17 παρατηρούμε ότι σε μη υπερφορτωμένο δίκτυο σε τοπολογία αστέρι με 60 κόμβους και 90 χρήστες με φορτίο 1, λιγότερες επαναλήψεις έχουμε με πιθανότητα μετανάστευσης p=0.9. Περισσότερες επαναλήψεις χρειάζονται για πιθανότητες p=0.1, που είναι όμως πολύ λιγότερες από τις επαναλήψεις με την ίδια πιθανότητα, ίδιο λ και ίδια τοπολογία σε μεγαλύτερο δίκτυο.

→ Τοπολογία αστέρι με 31 κόμβους



Σχήμα 5.18 Γραφική Παράσταση: 120 χρήστες x φόρτο 1
Δίκτυο αστέρι: 31 κόμβοι x χωρητικότητα 8

Στο σχήμα 5.18 παρατηρούμε ότι σε μη υπερφορτωμένο δίκτυο σε τοπολογία αστέρι με 31 κόμβους και 120 χρήστες με φορτίο 1, για κάθε πιθανότητα το σύστημα δε χρειάζεται καμιά επανάληψη μέχρι να φτάσει σε κατάσταση ισορροπίας. Η δυνατότητα εξυπηρέτησης 8 χρηστών κάθε κόμβου είναι αρκετή για να απορροφήσει εξ αρχής και τους 120 χρήστες.

Σενάριο Β:

Ένας αριθμός από κεντρικούς χρήστες έχει διπλάσια πιθανότητα τυχαίας αρχικής επιλογής για τον κάθε χρήστη του δικτύου.

Για το δεύτερο Σενάριο ήθελα να μελετήσω την συμπεριφορά του δικτύου όταν δίνεται προτεραιότητα σε κόμβους με περισσότερους γείτονες, στους κόμβους που δεν βρίσκονται στα άκρα οι οποίοι είναι κάπως πιο απομονωμένοι. Στο συγκεκριμένο Σενάριο τόσο το είδος του δικτύου, τοπολογία τετράγωνο και τοπολογία αστέρι, όσο και το μέγεθος του 100, 49, 25 κόμβοι και 101, 60, 31 κόμβοι αντίστοιχα, κρατούνται σταθερά. Για κάθε περίπτωση δικτύου, βαρυφορτωμένο με λ=1 και μη βαρυφορτωμένο με λ=2, εξετάζουμε τις περιπτώσεις α) όλοι οι κόμβοι να έχουν ίση πιθανότητα αρχικής επιλογής για κάθε χρήστη του δικτύου και β) μια ομάδα κεντρικών κόμβων να έχει διπλάσια πιθανότητα αρχικής επιλογής για κάθε χρήστη του δικτύου.

Για το δίκτυα σε τοπολογία τετράγωνο ισχύουν τα εξής:

- ➔ Δίκτυο με 100 κόμβους – 25 κεντρικοί κόμβοι
- ➔ Δίκτυο με 49 κόμβους – 11 κεντρικοί κόμβοι
- ➔ Δίκτυο με 25 κόμβους – 5 κεντρικοί κόμβοι

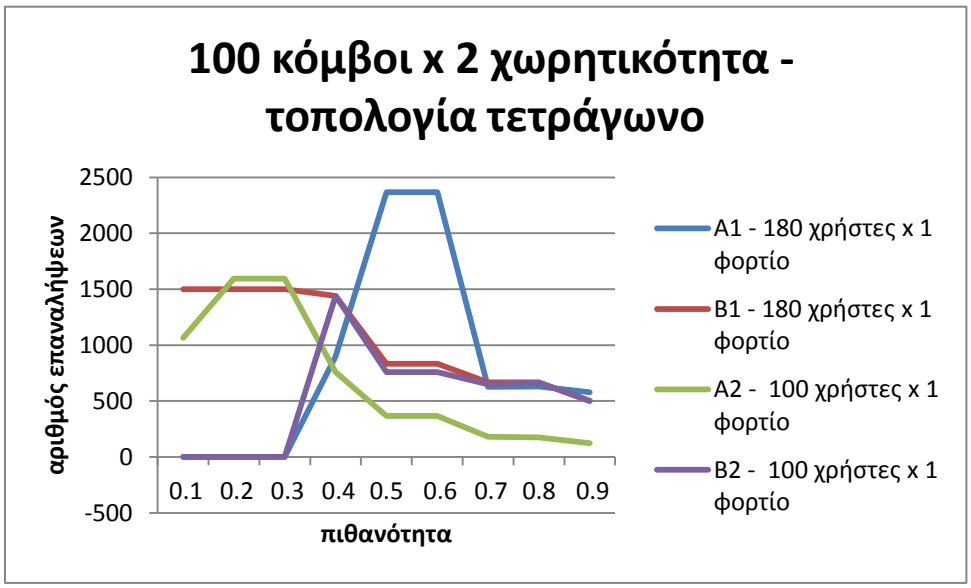
Για το δίκτυα σε τοπολογία τετράγωνο ισχύουν τα εξής:

- ➔ Δίκτυο με 101 κόμβους – 25 κεντρικοί κόμβοι
- ➔ Δίκτυο με 60 κόμβους – 16 κεντρικοί κόμβοι
- ➔ Δίκτυο με 31 κόμβους – 10 κεντρικοί κόμβοι

Η ιδέα των κεντρικών κόμβων στηρίζεται στην τρόπο διασύνδεσης των κόμβων σε ένα δίκτυο. Οι κεντρικοί κόμβοι συνδέονται με περισσότερους γείτονες, επομένως οι κόμβοι που εξυπηρετούν έχουν περισσότερες επιλογές. Επιπλέον οι κεντρικοί κατά πάσα πιθανότητα συνδέονται με γείτονες που επίσης έχουν περισσότερες επιλογές γειτόνων κοκ.

Επομένως σε αυτό το σενάριο θέλουμε να εξακριβώσουμε εάν δίνοντας στους κεντρικούς κόμβους διπλάσια πιθανότητα αρχικής επιλογής για κάθε χρήστη του δικτύου επέρχεται ταχύτερα ισορροπία φόρτου εργασίας στο δίκτυο.

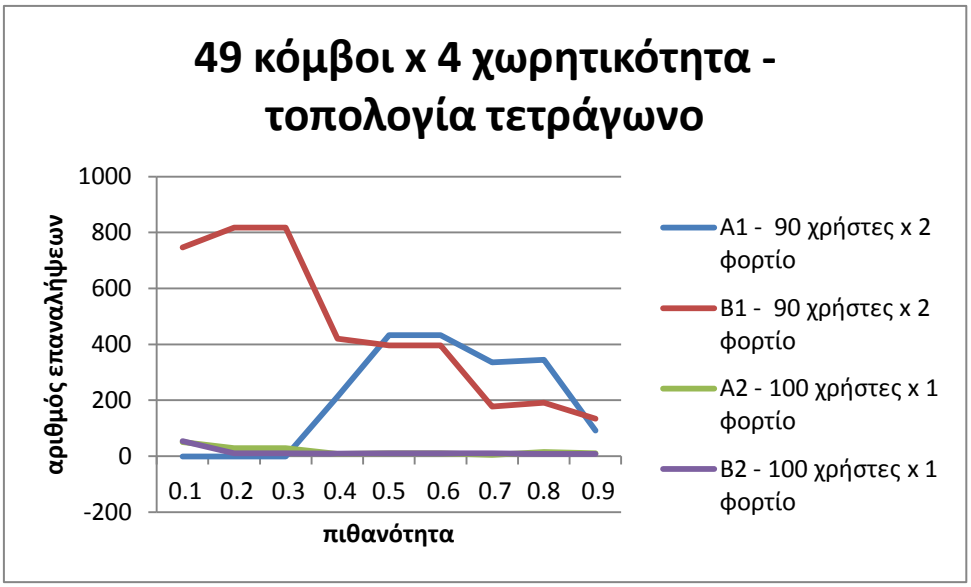
Οι γραφικές παραστάσεις που ακολουθούν παρατηρούμε ότι έχουν παρόμοια μορφή ανεξαρτήτως του δικτύου που χρησιμοποιείται.



Σχήμα 5.19 Γραφική Παράσταση: 100 χρήστες x φόρτο 1 για μη υπερφορτωμένο δίκτυο 100 κόμβων, λ=1, και 180 χρήστες x φορτίο 1 για δίκτυο υπερφορτωμένο.

Στο σχήμα 5.19 παρουσιάζεται η περίπτωση δικτύου με 100 κόμβους σε τοπολογία τετράγωνο. Στις περιπτώσεις A1 και A2 όλοι οι κόμβοι να έχουν ίση πιθανότητα αρχικής επιλογής για κάθε χρήστη του δικτύου με λ=1 και λ=2 αντίστοιχα. Στις περιπτώσεις B1 και B2 η ομάδα των 25 κεντρικών κόμβων να έχει διπλάσια πιθανότητα αρχικής επιλογής για κάθε χρήστη του δικτύου.

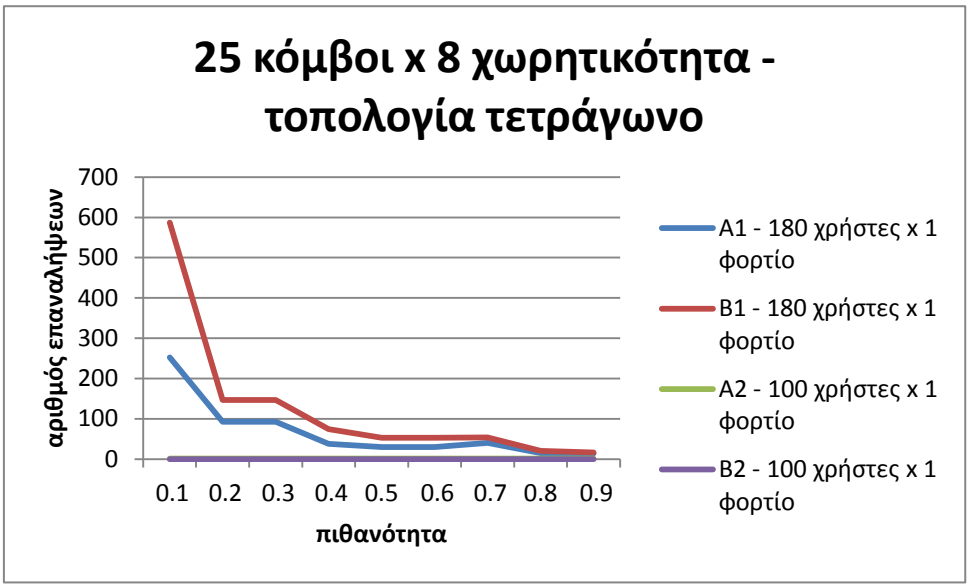
Παρατηρούμε ότι με προτεραιότητα στους κεντρικούς κόμβους δεν επέρχεται ταχύτερα ισορροπία στο σύστημα.



Σχήμα 5.20 Γραφική Παράσταση: 100 χρήστες x φόρτο 1 για μη υπερφορτωμένο δίκτυο 49 κόμβων, λ=1, και 90 χρήστες x φορτίο 2 για δίκτυο υπερφορτωμένο.

Στο σχήμα 5.20 παρουσιάζεται η περίπτωση δικτύου με 49 κόμβους σε τοπολογία τετράγωνο. Στις περιπτώσεις A1 και A2 όλοι οι κόμβοι να έχουν ίση πιθανότητα αρχικής επιλογής για κάθε χρήστη του δικτύου με λ=1 και λ=2 αντίστοιχα. Στις περιπτώσεις B1 και B2 η ομάδα των 16 κεντρικών κόμβων να έχει διπλάσια πιθανότητα αρχικής επιλογής για κάθε χρήστη του δικτύου.

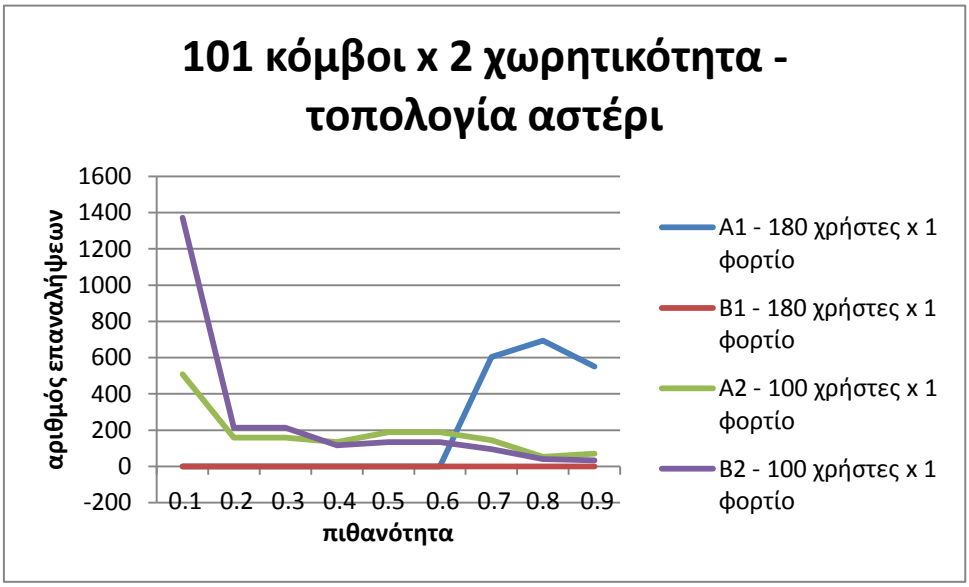
Παρατηρούμε ότι με προτεραιότητα στους κεντρικούς κόμβους δεν επέρχεται ταχύτερα ισορροπία στο σύστημα.



Σχήμα 5.21 Γραφική Παράσταση: 100 χρήστες x φόρτο 1 για μη υπερφορτωμένο δίκτυο 25 κόμβων, λ=1, και 180 χρήστες x φορτίο 1 για δίκτυο υπερφορτωμένο.

Στο σχήμα 5.21 παρουσιάζεται η περίπτωση δικτύου με 25 κόμβους σε τοπολογία τετράγωνο. Στις περιπτώσεις A1 και A2 όλοι οι κόμβοι να έχουν ίση πιθανότητα αρχικής επιλογής για κάθε χρήστη του δικτύου με λ=1 και λ=2 αντίστοιχα. Στις περιπτώσεις B1 και B2 η ομάδα των 5 κεντρικών κόμβων να έχει διπλάσια πιθανότητα αρχικής επιλογής για κάθε χρήστη του δικτύου.

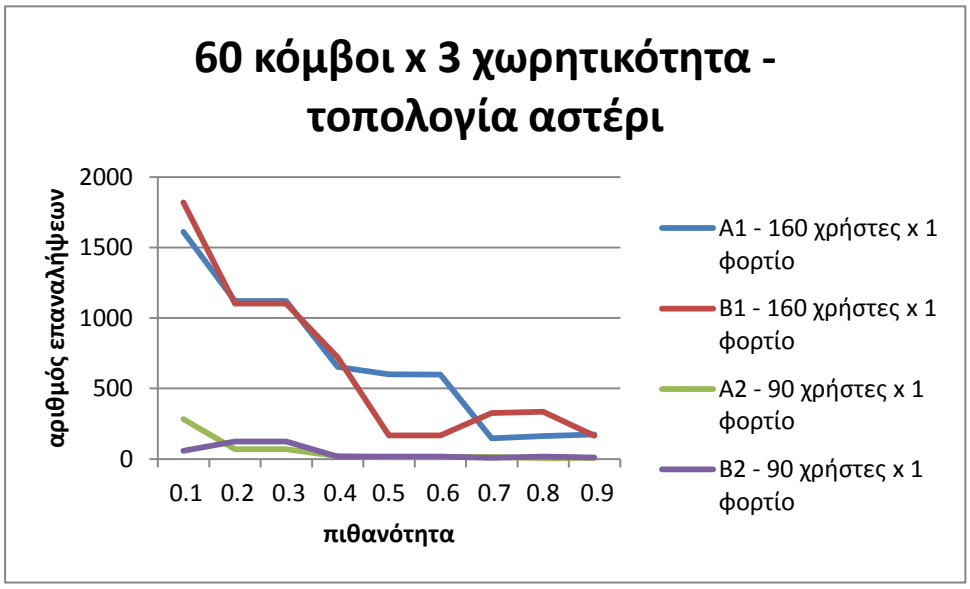
Παρατηρούμε ότι με προτεραιότητα στους κεντρικούς κόμβους δεν επέρχεται ταχύτερα ισορροπία στο σύστημα. Για τις περιπτώσεις 2, A2 και B2, που το σύστημα δεν είναι υπερφορτωμένο, για κάθε πιθανότητα το σύστημα χρειάζεται 1 και 0 επαναλήψεις αντίστοιχα για να έρθει σε κατάσταση ισορροπίας.



Σχήμα 5.22 Γραφική Παράσταση: 100 χρήστες x φόρτο 1 για μη υπερφορτωμένο δίκτυο 101 κόμβων, λ=1, και 180 χρήστες x φορτίο 1 για δίκτυο υπερφορτωμένο.

Στο σχήμα 5.22 παρουσιάζεται η περίπτωση δικτύου με 101 κόμβους σε τοπολογία αστέρι. Στις περιπτώσεις Α1 και Α2 όλοι οι κόμβοι να έχουν ίση πιθανότητα αρχικής επιλογής για κάθε χρήστη του δικτύου με λ=1 και λ=2 αντίστοιχα. Στις περιπτώσεις Β1 και Β2 η ομάδα των 25 κεντρικών κόμβων να έχει διπλάσια πιθανότητα αρχικής επιλογής για κάθε χρήστη του δικτύου.

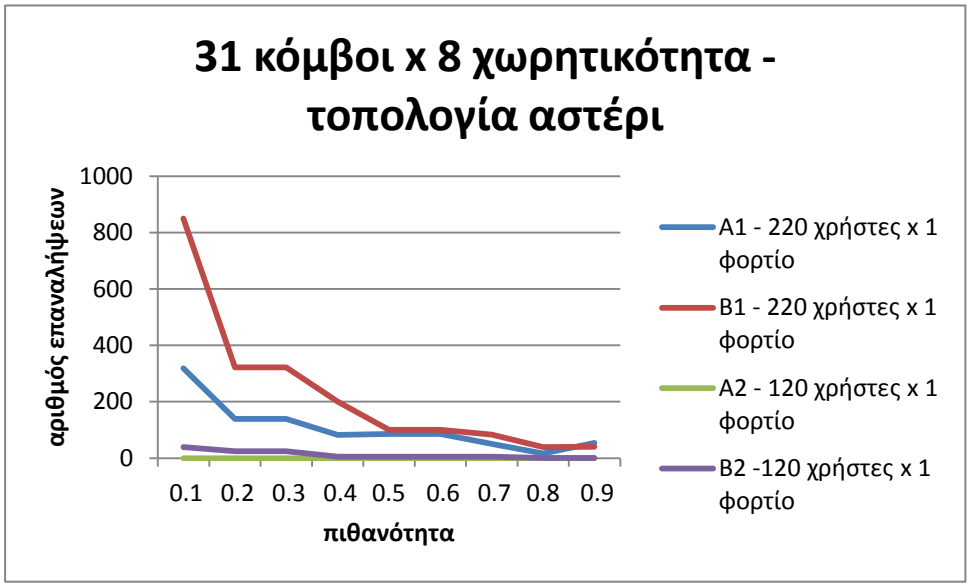
Παρατηρούμε ότι με προτεραιότητα στους κεντρικούς κόμβους δεν επέρχεται ταχύτερα ισορροπία στο σύστημα.



Σχήμα 5.23 Γραφική Παράσταση: 90 χρήστες x φόρτο 1 για μη υπερφορτωμένο δίκτυο 60 κόμβων, λ=1, και 160 χρήστες x φορτίο 1 για δίκτυο υπερφορτωμένο.

Στο σχήμα 5.23 παρουσιάζεται η περίπτωση δικτύου με 60 κόμβους σε τοπολογία αστέρι. Στις περιπτώσεις A1 και A2 όλοι οι κόμβοι να έχουν ίση πιθανότητα αρχικής επιλογής για κάθε χρήστη του δικτύου με λ=1 και λ=2 αντίστοιχα. Στις περιπτώσεις B1 και B2 η ομάδα των 16 κεντρικών κόμβων να έχει διπλάσια πιθανότητα αρχικής επιλογής για κάθε χρήστη του δικτύου.

Παρατηρούμε ότι με προτεραιότητα στους κεντρικούς κόμβους δεν επέρχεται ταχύτερα ισορροπία στο σύστημα.



Σχήμα 5.24 Γραφική Παράσταση: 120 χρήστες x φόρτο 1 για μη υπερφορτωμένο δίκτυο 31 κόμβων, λ=1, και 220 χρήστες x φορτίο 1 για δίκτυο υπερφορτωμένο.

Στο σχήμα 5.24 παρουσιάζεται η περίπτωση δικτύου με 60 κόμβους σε τοπολογία αστέρι. Στις περιπτώσεις Α1 και Α2 όλοι οι κόμβοι να έχουν ίση πιθανότητα αρχικής επιλογής για κάθε χρήστη του δικτύου με λ=1 και λ=2 αντίστοιχα. Στις περιπτώσεις Β1 και Β2 η ομάδα των 10 κεντρικών κόμβων να έχει διπλάσια πιθανότητα αρχικής επιλογής για κάθε χρήστη του δικτύου.

Παρατηρούμε ότι με προτεραιότητα στους κεντρικούς κόμβους δεν επέρχεται ταχύτερα ισορροπία στο σύστημα.

Σε γενικές γραμμές, η βελτιστοποίηση που επιφέρει στο δίκτυο είναι η μεγάλη μείωση των επαναλήψεων στις πιθανότητες που επιφέρουν μεγάλο αριθμό επαναλήψεων μέχρι το σύστημα να φτάσει σε ισορροπία και για τις δύο τοπολογίες (κυρίως για p=0.3 – 0.7).

Παρόλα αυτά δεν φαίνεται να μειώνει τον αριθμό επαναλήψεων για τις πιθανότητες που φέρουν βέλτιστα αποτελέσματα (p= 0.8 – 0.9)

Κεφάλαιο 6

Θεωρία Παιγνίων

6.1 Εισαγωγή στη Θεωρία Παιγνίων

Στο κεφάλαιο αυτό θα παρουσιαστούν σε συντομία τα βασικά χαρακτηριστικά της Θεωρίας Παιγνίων και στη συνέχεια θα αποδείξουμε ότι το πρόβλημά μας μπορεί να εκφραστεί μέσω αυτής. Περισσότερα στοιχεία και πληροφορίες γύρω από το θέμα της Θεωρίας Παιγνίων μπορεί κανείς να βρει στην αναφορά [11]

Θεωρία Παιγνίων είναι η μέθοδος ανάλυσης προβλημάτων που μελετούν τον τρόπο λήψης αποφάσεων δύο ή περισσότερων χρηστών σε περίπτωση σύγκρουσης ή συνεργασίας με τους υπόλοιπους.

Σκοπός του κάθε παίχτη είναι να εμποδίσει τον αντίπαλό του να αποκτήσει πλεονεκτήματα που θα περιορίσουν τα δικά του κέρδη. Επομένως, οι ενέργειες που

προβαίνει σχετίζονται άμεσα με τη στρατηγική που επιλέγει να ακολουθήσει ο αντίπαλος.

Τα βασικά στοιχεία ενός παιχνιδιού είναι τα εξής:

- ➔ Το σύνολο των παιχτών, players: Συμβολίζεται με *N={1, 2, … , n}* και σε περίπτωση που θέλουμε να αναφερθούμε σε ένα παίκτη ατομικά με *j*.
- ➔ Ενέργειες παικτών, actions : Συμβολίζονται με α=(α_ι , $\alpha_{\iota-1}$) όπου α_ι η ενέργεια του ι-οστού παίκτη και $\alpha_{\iota-1}$, οι ενέργειες των υπόλοιπων n-1 παικτών.
- ➔ Αποτελέσματα, outcomes
- ➔ Συναρτήσεις χρησιμότητας, Utility Functions: αντιπροσωπεύει τις κινήσεις των παιχτών. Συμβολίζουμε το διάνυσμα όλων συναρτήσεων χρησιμότητας με U=($U_1, U_2, …, U_n$)

Αναφορικά με τη Χρονική Διάταξη Αποφάσεων, τον τρόπο με τον οποίο οι παίκτες λαμβάνουν τις αποφάσεις τους στο παιχνίδι μας, ακολουθείται η σύγχρονη διαδικασία λήψης αποφάσεων, στην οποία οι κόμβοι επιλέγουν ταυτόχρονα τις ενέργειές τους.

6.2 Ορισμός 1: Ισορροπία Nash [11]

Μια ισορροπία Nash είναι ένα σύνολο στρατηγικών α* με την ιδιότητα ότι κανένας παίκτης δεν βελτιώνει τη θέση του επιλέγοντας μια ενέργεια διαφορετική από αυτή του συνόλου στρατηγικών α*, με δεδομένο ότι οι υπόλοιποι παίκτες δεν θα αλλάξουν την κίνησή τους.

6.3 Ορισμός 2: e-Ισορροπία Nash [8]

Ο ορισμός της e-Ισορροπίας Nash έχει μια πιο χαλαρή έννοια της ισορροπίας Nash, κατά την οποία είναι αρκετό να έχω φτάσει σε μια κατάσταση όπου θα έχω σχεδόν ισορροπία Nash. Η μεταβλητή e παίρνει τιμές e= 0.1 – 0.9 αναλόγως με το πόσο προσεγγιστικό θέλουμε να είναι το αποτέλεσμα

6.4 Ορισμός 3: Δυνητικό Παιχνίδι [10]

Στη Θεωρία Παιγνίων, ένα παιχνίδι ονομάζεται δυνητικό (potential game) εάν το κίνητρο όλων των παιχτών για αλλαγή στρατηγικής μπορεί να εκφραστεί με μια συνάρτηση, που ονομάζεται potential function. Μια συνάρτηση *P: Y → R* είναι

δυνητική για ένα παιχνίδι Γ εάν για κάθε $i \in N$, όπου N σύνολο των παικτών, και για κάθε $y^{-i} \in Y^{-i}$, όπου Y^i σύνολο στρατηγικών χρήστη i.

$U^i(y^{-i}, x) - U^i(y^{-i}, z) > 0$ εάν και μόνο εάν $P(y^{-i}, x) - P(y^{-i}, z) > 0$ για κάθε $x, z \in Y^i$

Τα παιχνίδια αυτά ορίστηκαν επίσημα στη δημοσίευση []

6.5 Συμβολισμοί

Για να μπορέσουμε να παρουσιάσουμε το παιχνίδι μας και να αποδείξουμε ότι είναι δυνητικό, παρουσιάζουμε παρακάτω τους συμβολισμούς που θα χρησιμοποιήσουμε.

Υποθέτουμε ότι έχουμε ένα αυθαίρετο , μη κατευθυνόμενο και συνδεδεμένο γράφο G = (V,E) με n = |V| κορυφές οι οποίες αντιπροσωπεύουν τις μηχανές εξυπηρέτησης.

Υπάρχουν m εργασίες , tasks, στο σύστημα, τις οποίες αποκαλούμε χρήστες και κατά την αρχικοποίηση του συστήματος ανατίθενται αυθαίρετα σε n μηχανές. Με x υποδηλώνουμε μία κατάσταση, state, του συστήματος, δηλαδή μια ανάθεση εργασιών στις μηχανές.

Κάθε εργασία – χρήστης j έχει ένα βάρος $W^j \in$ N εκτός κι αν υποθέσουμε εναλλακτικά ότι όλες οι εργασίες είναι πανομοιότυπες, με $W^j = 1$.

Κάθε μηχανή i έχει συγκεκριμένη χωρητικότητα, capacity, C_i που μπορεί να ανεχτεί. Ορίζουμε τη χωρητικότητα που έχει ο κόμβος στον οποίο βρίσκεται ο χρήστης j με C^j και C= $\sum_{i=1}^{m} C_i$.

Στην περίπτωση ομοιόμορφων εργασιών, όπου όλες οι διεργασίες έχουν βάρος που ισούται με ένα (1), ο φόρτος εργασίας μιας μηχανής ορίζεται ως το άθροισμα των εργασιών που ανατίθενται σε αυτή. Σε περίπτωση μας, που οι εργασίες έχουν βάρος $W^j > 1$, ο φόρτος εργασίας της μηχανής ορίζεται ως το άθροισμα των βαρών των εργασιών οι οποίες διαμοιράζονται την ταχύτητα του.

Συγκεκριμένα, με $W_i(x)$ ορίζουμε το βάρος της μηχανής i στην κατάσταση x , δηλαδή το άθροισμα των βαρών όλων των εργασιών που βρίσκονται στην μηχανή i. Για κάθε κατάσταση x ο φόρτος εργασίας ενός χρήστη συμβολίζεται με $W^j(x)$ Ο ολικός φόρτος εργασίας των κόμβων συμβολίζεται με W:= $\sum_{i \in n} W_i(x)$.

Κάθε κατάσταση έχει τη δική της ωφελιμότητα για κάθε χρήστη j, και συμβολίζεται με U_j^x.

Κάθε εργασία είναι ένας εγωιστής πράκτορας ο οποίος προσπαθεί να ελαχιστοποιήσει το δικό του φόρτο εργασίας. Η εργασία γνωρίζει μόνο το φόρτο εργασίας της μηχανής που βρίσκεται εκείνη τη χρονική στιγμή και μπορεί να ελέγξει μόνο το φόρτο εργασίας των γειτονικών. Θεωρούμε ότι έχουμε μια βασισμένη σε γύρους διαδικασία. Σε κάθε επανάληψη κάθε εργασία χρησιμοποιεί το πρωτόκολλο μετανάστευσης για να αποφασίσει σε ποια από τις γειτονικές μηχανές ενδεχομένως θα μεταναστεύσει.

6.6 Συνάρτηση (αντι)χρησιμότητας

Σε κάθε παιχνίδι, η χρησιμότητα, utility, αντιπροσωπεύει τις κινήσεις των παιχτών. Συνάρτηση χρησιμότητας για έναν παίχτη αναθέτει ένα αριθμό για κάθε πιθανό αποτέλεσμα του παιχνιδιού, όπου μεγάλος αριθμός συνεπάγεται πιο επιθυμητό αποτέλεσμα.

Στη δική μας περίπτωση ορίζουμε αντι – χρησιμότητα, disutility, θεωρώντας ότι μεγάλος αριθμός υποδηλώνει το μέγεθος της μη ικανοποίησης του χρήστη.

Επομένως με βαθμό 1 δηλώνουμε ότι ο χρήστης δεν ικανοποιείται στον κόμβο που βρίσκεται στη συγκεκριμένη κατάσταση, ενώ με βαθμό 0 θεωρούμε ότι ικανοποιείται.

Παρακάτω παρουσιάζουμε τη συνάρτηση αντι-χρησιμότητας:

$$U_j^x = \begin{cases} 0, \ if \ \frac{W^j(x)}{C^j} \leq 1 \leftrightarrow W^j(x) \leq C^j \\ 1, \ if \ \frac{W^j(x)}{C^j} > 1 \leftrightarrow W^j(x) > C^j \end{cases}$$

6.7 Δυνητική Συνάρτηση

Θα δείξουμε ότι το παιχνίδι που μελετήσαμε είναι δυνητικό, χρησιμοποιώντας δυνητική συνάρτηση P(x) = $\sum_{i \in n} overload(i, x)$ όπου

Overload (i,x) $= \begin{cases} 0, & if\ W^j(x) < C^j \\ W^j(x) - C^j(x), & if\ W^j(x) \geq C^j \end{cases}$

Ορίζουμε ως G_1 την ειδική περίπτωση των παιχνιδιών όπου οι όλες οι εργασίες είναι συμμετρικές με βάρος ίσο με 1.

6.8 Θεώρημα 1 – Απόδειξη

Θεώρημα 1: Το παιχνίδι G1 είναι ένα δυνητικό παιχνίδι.

Απόδειξη: Η απόδειξη επιβεβαιώνει ότι το παιχνίδι G_1 ικανοποιεί τον ορισμό ενός δυνητικού παιχνιδιού. (Ορισμός 3)

Έστω ότι βρίσκομαι στην κατάσταση x. Θέλουμε να δείξουμε ότι σε περίπτωση που ο χρήστης j αποφασίζει να μεταναστεύσει από κόμβο i σε κόμβο k όπου προκύπτει η κατάσταση z.
Τότε
(α) U_j^x - U_j^z < 0 αν και μόνο εάν P(x) < P(z)
Και
(β) U_j^x - U_j^z ≥ 0 αν και μόνο εάν P(x) ≥ P(z)

Για την περίπτωση α) :
Υποθέτουμε ότι $U_j^x - U_j^z$ <0 και θέλουμε να δείξουμε ότι P(x) < P(z). Για να ισχύει η περίπτωση θα πρέπει να ισχύει ότι U_j^x=0 και U_j^z=1νως $W^j(x) \leq C^j$ και $W^j(z) > C^j$.
Ας θεωρήσουμε τον κόμβο i στις καταστάσεις x και z, δηλαδή πριν και μετά την μετακίνηση του χρήστη j.
Παρατηρούμε ότι :
$W_i(x) \leq C_i$ και συνεπάγεται ότι $W_i(x) = W_i(x) - 1 < C_i$
Επομένως $overload(i,x) = overload(i,z) = 0$
Αναφορικά με τον κόμβο k στις καταστάσεις x και z παρατηρούμε ότι:
$W_k(z) = W_k(x) + 1$
Επιπλέον γνωρίζουμε ότι $W_k(z) > C_k$
Επομένως $overload(j,x) = W_k(x) + 1 - C_k$
Παρόμοια, αφού $W_k(x) + 1 = W_k(z) > C_k$ έχουμε ότι
$W_k(x) \geq C^j$ και επομένως
$overload(k,x) = W_k(x) - C_k = overload(i,z) - 1$

Επομένως P(x) - P(z) = $\sum_{i\in n} overload\ (i,x) - overload\ (i,z)$ = $\sum_{i\in n-\{i,k\}} overload(i,x) - overload(i,z) +$

$+ (overload(i,x - overload(i,z))$

$+ (overload(k,x) - overload(k,z)) =$

$\sum_{i\in n-\{i,k\}}$ $\begin{matrix}0+\\(0+0)+\\(overload(k,x) - overload(k,z))\end{matrix}$ = $overload(k,z)$ - 1 - $overload(k,z)$ =

= -1<0

Αυτό ολοκληρώνει την απόδειξη της περίπτωσης α.

Για την περίπτωση β):

Υποθέτουμε ότι $U_j^x - U_j^z \geq 0$ και θέλουμε να δείξουμε ότι P(x) ≥ P(z). Για να ισχύει η περίπτωση θα πρέπει να ισχύει ότι I) U_j^x=0 και U_j^z=0 ή

II) U_j^x=1 και U_j^z=0 ή

III) U_j^x=1 και U_j^z=1

Για την υποπερίπτωση I) ισχύει ότι U_j^x=0 και U_j^z=0 επομένως $W^j(x) \leq C^j$ και $W^j(z) \leq C^j$

Ας θεωρήσουμε τον κόμβο i στις καταστάσεις x και z, δηλαδή πριν και μετά την μετακίνηση του χρήστη j.

Παρατηρούμε ότι :

$W_i(x) \leq C_i$ και συνεπάγεται ότι $W_i(x) = W_i(x) - 1 < C_i$

Επομένως $overload(i,x) = overload(i,z) = 0$

Αναφορικά με τον κόμβο k στις καταστάσεις x και z παρατηρούμε ότι:

$W_k(z) \leq C_i$ και συνεπάγεται ότι $W_k(x) = W_k(z) - 1 < C_i$

Επομένως $overload(k,x) = overload(k,z) - 1 = 0$

Επομένως P(x) - P(z) = $\sum_{i\in n} overload\ (i,x) - overload\ (i,z)$ = $\sum_{i\in n-\{i,k\}} overload(i,x) - overload(i,z) +$

$+ (overload(i,x - overload(i,z))$

$+ (overload(k,x) - overload(k,z)) =$

$\sum_{i\in n-\{i,k\}} \begin{matrix}0+\\(0+0)\\(0+0)\end{matrix} += 0 \geq 0$

Αυτό ολοκληρώνει την απόδειξη της υποπερίπτωσης β/I

Για την υποπερίπτωση ΙΙ) ισχύει ότι U_j^x=1 και U_j^z=0 επομένως $W^j(x) > C^j$ και $W^j(z) \leq C^j$

Ας θεωρήσουμε τον κόμβο i στις καταστάσεις x και z, δηλαδή πριν και μετά την μετακίνηση του χρήστη j.

Παρατηρούμε ότι :

$W_i(\mathrm{x}) = W_i(\mathrm{z}) - 1$

Επιπλέον γνωρίζουμε ότι $W_i(\mathrm{x}) > C_i$

Αναφορικά με τον κόμβο k στις καταστάσεις x και z παρατηρούμε ότι:

$W_k(\mathrm{z}) \leq C_i$ και συνεπάγεται ότι $W_k(\mathrm{x}) = W_k(\mathrm{z}) - 1 < C_i$

Επομένως $overload(k,x) = overload(k,z) - 1 = 0$

Επομένως $W_i(\mathrm{z}) \geq C_i$

Και άρα $overload(i,z) = W_i(\mathrm{z}) - C_i = overload(i,x) - 1$

Επομένως $\mathrm{P(x)} - \mathrm{P(z)} = \sum_{i\in n} overload\,(i,x) - overload\,(i,z) =$

$\sum_{i\in n-\{i,k\}} overload(i,x) - overload(i,z) +$

$+ (overload(i,x) - overload(i,z))$

$+ (overload(k,x) - overload(k,z)) =$

$\sum_{i\in n-\{i,k\}} (overload(i,x) - \begin{matrix}0 + \\ (overload(i,x) - 1))\end{matrix} += \begin{matrix}\\ \\ (0-0)\end{matrix}$

$0 + overload(i,x) - overload(i,x) + 1 + 0 = 1 > 0$

Αυτό ολοκληρώνει την απόδειξη της υποπερίπτωσης β/ΙΙ

Για την υποπερίπτωση ΙΙΙ) ισχύει ότι U_j^x=1 και U_j^z=1 επομένως $W^j(x) > C^j$ και $W^j(z) > C^j$

Ας θεωρήσουμε τον κόμβο i στις καταστάσεις x και z, δηλαδή πριν και μετά την μετακίνηση του χρήστη j.

Παρατηρούμε ότι :

$W_i(\mathrm{x}) = W_i(\mathrm{z}) - 1$

Επιπλέον γνωρίζουμε ότι $W_i(\mathrm{x}) > C_i$

Επομένως $W_i(\mathrm{z}) \geq C_i$

Και άρα $overload(i,z) = W_i(\mathrm{z}) - C_i = overload(i,x) - 1$

Αναφορικά με τον κόμβο k στις καταστάσεις x και z παρατηρούμε ότι:

$W_k(\mathrm{z})$ $W_k(\mathrm{x}) + 1$. Επιπλέον γνωρίζουμε ότι $W_k(\mathrm{z}) > C_k$

Επομένως $overload(j,z) = W_k(\mathrm{x}) + 1 - C_k$

Παρόμοια, αφού $W_k(\mathrm{x}) + 1 = W_k(\mathrm{z}) > C_k$ έχουμε ότι

$W_k(\mathrm{x}) \geq C^j$ και επομένως

$overload(k,x) = W_k(\mathrm{x}) - C_k = overload(i,z) - 1$

Επομένως P(x) - P(z) = $\sum_{i \in n} overload\ (i,x) - overload\ (i,z)$ =

$\sum_{i \in n-\{i,k\}} overload(i,x) - overload(i,z) +$

$+ (overload(i,x - overload(i,z))$

$+ (overload(k,x) - overload(k,z)) =$

$0 +$

$\sum_{i \in n-\{i,k\}} (overload(i,x) - (overload(i,x) - 1)) +=$
$((overload(k,z) - 1) - overload(k,z))$

$0 + 1 - 1 = 0 \geq 0$

Αυτό ολοκληρώνει την απόδειξη της υποπερίπτωσης β/III

Κεφάλαιο 7

Συμπεράσματα

7.1 Γενικά Συμπεράσματα

Η Εξισορρόπηση Φόρτου Εργασίας σε δίκτυα είναι ένα μεγάλο και περίπλοκο θέμα. Κάθε δίκτυο έχει τα δικά του χαρακτηριστικά και ιδιαιτερότητες και καθώς οι απαιτήσεις των χρηστών σε υπολογιστική ισχύ και ταχύτητα επεξεργασίας αυξάνονται συνεχώς, η πρόκληση της Εξισορρόπησης Φόρτου Εργασίας γίνεται ακόμη μεγαλύτερη, αφού χωρίς τη σωστή διαχείριση οι ανάγκες των χρηστών δε μπορούν να ικανοποιηθούν.

Το πρόβλημα της διαχείρισης «Μεγάλων Δεδομένων» φανερώνει την αναποτελεσματικότητα των κεντρικοποιημένων αρχιτεκτονικών, ενώ η είσοδος στα κατανεμημένα συστήματα απαιτεί σωστή εξισορρόπηση φόρτου μεταξύ των παροχών για να μπορεί να λειτουργήσει. Τέλος, με την εισαγωγή της παράλληλης επεξεργασίας στο χώρο της Επιστήμης της Πληροφορικής, η Εξισορρόπηση Φόρτου Εργασίας γίνεται αδήριτη ανάγκη. Η Εξισορρόπηση Φόρτου Εργασίας, εν τέλει, αποτελεί τον ακρογωνιαίο λίθο της διαχείρισης των δεδομένων και χωρίς σωστή κατανομή φορτίου μεταξύ των επεξεργαστών ο όρος της Παράλληλης Επεξεργασίας γίνεται δώρον άδωρον.

Οι Αλγόριθμοι Εξυπηρέτησης Φόρτου Εργασίας που έχουν προταθεί και εφαρμοστεί μέχρι στιγμής, παρουσιάζουν προβλήματα υπερφόρτωσης των παροχών και υπερχείλισης σε μεγάλα δίκτυα, ενώ συγχρόνως έχουν μεγάλες απαιτήσεις μνήμης. [3,4] Σε Αλγορίθμους Παράλληλων Υπολογιστών η τοπολογία των κόμβων παίζει σημαντικό ρόλο στην αποδοτικότητα.

Οι Αλγόριθμοι Εξισορρόπησης Φόρτου Εργασίας που αποτελούν θεωρητικές προσεγγίσεις, [1,2,5] δεν εισάγουν την έννοια της χωρητικότητας. Κάθε χρήστης όταν επιχειρεί μετανάστευση, αρκείται στην γνώση του φόρτου εργασίας του γείτονα που επιλέγει πιθανοτικά, ενώ δε γνωρίζει τη χωρητικότητα του.

Η μελέτη του Αλγόριθμου Μετανάστευσης Πληθυσμών [6] δείχνει τα πουλιά να λαμβάνουν υπ όψιν τους τη χωρητικότητα-μέγεθος της περιοχής που πρόκειται να μεταναστεύσουν. Ο Αλγόριθμος εμπνευσμένος από τη φύση, δίνει λύση στο δικό μας πρόβλημα που εμπλέκει και τη χωρητικότητα των κόμβων ως σημαντική παράμετρο επιλογής κόμβου μετανάστευσης από το χρήστη. Επιπλέον η μελέτη του προβλήματος Μπάλων–Κάδων [7] έρχεται να επιβεβαιώσει τη σημασία της χωρητικότητας στη λήψη της απόφασης μετανάστευσης. Εδώ τίθεται και ένα άλλο ερώτημα: Ποια είναι η σωστή αναλογία μεταξύ χωρητικότητας και κόμβων ενός δικτύου ώστε να έχουμε τα βέλτιστα αποτελέσματα.

Τόσο η χωρητικότητα του κόμβου–γείτονα που επιλέγεται, όσο και η ολική χωρητικότητα των υπόλοιπων γειτόνων, παίζουν σημαντικό ρόλο στην πιθανότητα μετανάστευσης του χρήστη. Την ίδια στιγμή η σύγκριση του φόρτου εργασίας των δύο κόμβων, εξυπηρετητή και υποψήφιου προς μετανάστευση, για επιλογή μετανάστευσης του χρήστη, δε φαίνεται να μας καθοδηγεί σωστά στην απόφαση μετανάστευσης. Αυτό αποδεικνύεται με τη σύγκριση που έγινε στην παρούσα εργασία ανάμεσα σε δύο αλγορίθμους. Ο πρώτος, λάμβανε υπ όψιν τη διαφορά φόρτου εργασίας εξυπηρετητή–γείτονα υποψήφιου προς μετανάστευση στην πιθανότητα μετανάστευσης, ενώ ο δεύτερος λάμβανε υπ όψιν τη χωρητικότητα του υποψήφιου γείτονα και την ολική χωρητικότητα των γειτόνων του εξυπηρετητή.

Μετά από πολλές προσομοιώσεις, συγκρίνοντας τους χρόνους σύγκλισης μέχρι την επίτευξη ισορροπίας στο δίκτυο, καταλήξαμε στο συμπέρασμα ότι λαμβάνοντας υπ όψιν τη χωρητικότητα των κόμβων του δικτύου μπορούμε να έχουμε πολύ πιο αντιπροσωπευτική επιλογή μετανάστευσης ή όχι ενός χρήστη από τον εξυπηρετητή

του, ούτως ώστε να πετύχουμε τον ταχύτερο χρόνο σύγκλισης και κατάσταση ισορροπίας στο δίκτυο που επεξεργαζόμαστε. Επιπλέον, ακόμη και με βαρυφορτωμένο δίκτυο με την εισαγωγή της χωρητικότητας στην πιθανότητα επιλογής μετανάστευσης, μπορεί να επέλθει ισορροπία σε ανεκτό χρόνο, κάτι που δε συμβαίνει σε αντίθετη περίπτωση.

Επεκτείνοντας την πειραματική μας μελέτη στον Βέλτιστο Αλγόριθμο, εξετάζοντας διάφορες παραμέτρους του δικτύου καταλήξαμε σε συμπεράσματα που αφορούν στο μέγεθος της χωρητικότητας των κόμβων εξυπηρέτησης. Οι προσομοιώσεις καταδεικνύουν ότι με λιγότερους κόμβους μεγαλύτερης χωρητικότητας επέρχεται γρηγορότερα η ισορροπία στο δίκτυο, σε αντίθεση με περισσότερους κόμβους μικρής χωρητικότητας. Επιπλέον, με τοπολογία αστέρι έναντι τετραγώνου έχουμε καλύτερα αποτελέσματα στο χρόνο σύγκλισης και τέλος μια επιλογή κεντρικών κόμβων του δικτύου με διπλάσια πιθανότητα αρχικής επιλογής κόμβων εξυπηρέτησης από τους χρήστες μπορεί να βελτιώσει το χρόνο σύγκλισης σε περιπτώσεις που η ισοπιθανοτική επιλογή αδυνατεί.

Τέλος, με ανάλυση του προβλήματός μας μέσω της Θεωρίας Παιγνίων [8,9,11], καταλήξαμε στο συμπέρασμα ότι το Παιχνίδι μας είναι ένα Δυνητικό Παιχνίδι [10] στην ειδική περίπτωση όπου όλοι οι κόμβοι έχουν ίδιο φόρτο εργασίας και αυτός ισούται με ένα(1).

7.2 Τελική Ανασκόπηση

Η Μελέτη Εξισορρόπησης Φόρτου Εργασίας ήταν ιδιαίτερα ενδιαφέρουσα. Μέσα από αυτήν κατάφερα να αξιοποιήσω τις γνώσεις μου από το χώρο της Πληροφορικής και συγκεκριμένα τον κλάδο των Δικτύων και να τις επεκτείνω. Η πορεία της μελέτης, με έκανε να αντιληφθώ τη σημαντικότητα του θέματος Εξισορρόπησης Φόρτου Εργασίας και την επιτακτική ανάγκη της στο συνεχώς μεταβαλλόμενο κόσμο της Επιστήμης της Πληροφορικής. Η πειραματική διαδικασία μού πρόσφερε την επιβεβαίωση ότι ο τομέας που επέλεξα να εμβαθύνω τις γνώσεις μου δεν είναι καθόλου απλή υπόθεση. Απαιτεί πολλές πειραματικές προσπάθειες και μελέτη των αποτελεσμάτων πριν καταλήξουμε σε έγκυρα συμπεράσματα.

Στην πορεία αυτή της Διπλωματικής μου Εργασίας συνάντησα αρκετές δυσκολίες σε διάφορες φάσεις της εργασίας. Αρχικά έπρεπε να αφιερώσω αρκετό χρόνο στην

κατανόηση των άρθρων που απαιτούσε να μελετήσω η θεωρητική μελέτη της Εργασίας. Ιδιαίτερη δυσκολία συνάντησα στην μελέτη του άρθρου που αναφερόταν στην Μετανάστευση των πληθυσμών, αφού έπρεπε να εξοικειωθώ με βασικούς όρους της βιολογίας των πουλιών που δεν είχα ασχοληθεί στο παρελθόν. Επίσης, κατά την δημιουργία των αλγορίθμων υπήρξαν δυσκολίες όσον αφορά στην δομή που έπρεπε να ακολουθήσω για τον καθένα ώστε να μπορεί να υπάρξει μέτρο σύγκρισης ανάμεσά τους και στην πιθανότητα μετανάστευσης χρήστη που έπρεπε να οριστεί στον κάθε Αλγόριθμο ώστε η μελέτη τους να έχει ενδιαφέρον. Στη συνέχεια είχα να αντιμετωπίσω το πρόβλημα της κατανόησης του όρου «Θεωρία Παιγνίων», που σχετίζεται με εφαρμοσμένα μαθηματικά, περιοχή που δεν προσφέρεται στο πρόγραμμα σπουδών προπτυχιακού επιπέδου Τμήματος Πληροφορικής στο Πανεπιστήμιό μας. Ιδιαίτερα με δυσκόλεψε η πορεία καταγραφής και απόδειξης ότι το πρόβλημα μας είναι ένα Δυνητικό Παιχνίδι. Τελειώνοντας την Εργασία αισθάνομαι περισσότερο εξοικειωμένη με μαθηματικούς όρους και αποδείξεις και νιώθω να έχω εμπλουτίσει τις γνώσεις μου γύρω από το θέμα της Εξισορρόπησης Φόρτου Εργασίας σε δίκτυα Υπολογιστών.

7.3 Μελλοντική Έρευνα

Ο Φόρτος Εργασίας σε Κατανεμημένα Συστήματα είναι απαραίτητο να κατανέμεται σωστά ούτως ώστε να μπορεί να υπάρξει αναβάθμιση των Συστημάτων. Υπάρχουν συνεπώς πολλές προοπτικές έρευνας στο χώρο. Οι Αλγόριθμοι που προτείνουμε αποδεικνύονται προσοδοφόροι αλλά αυτό δε σημαίνει ότι δε μπορούν να επεκταθούν ή και να αναπροσαρμοστούν ώστε να συνδυάζουν κι άλλες παραμέτρους έρευνας.

Μια ιδέα αναπροσαρμογής θα ήταν η μερική γνώση του Δικτύου όπως προτείνει το [4] στο γενετικό αλγόριθμο. Σε αυτή την περίπτωση θα μπορούσε να μελετηθεί το κόστος μιας τέτοιας γνώσης και κατά πόσο αυτό το κόστος εξουδετερώνεται ή όχι από την βελτίωση του χρόνου σύγκλισης.

Άλλη ιδέα θα ήταν η πειραματική μελέτη και με άλλες τοπολογίες ή πλέγματα υπολογιστών που δεν επιφέρει οποιαδήποτε υπολογιστική επιβάρυνση και δεν απαιτεί επιπλέον μνήμη.

Πέραν από επέκταση ή αναπροσαρμογή του μοντέλου, μπορούν να γίνουν έρευνες με άλλους πληθυσμούς μετανάστευσης όπως έντομα ή μεταναστευτικά πουλιά, αφού φαίνεται οι «φυσικοί» αλγόριθμοι έχουν πολλά να μας διδάξουν.

Πάντως η βιβλιογραφία υπόσχεται προώθηση της έρευνας και οι ωφέλειες που θα προκύψουν θα είναι σίγουρα καθοριστικές για την Επιστήμη της Πληροφορικής.

Βιβλιογραφία

[1] H.Ackermann, S.Fischer, M.Schongens and M. Hoefer, “Distributed Algorithms for QoS Load Balancing,” 24 December 2010

[2] Petra Berenbrink, T.Friodetzky , L.A Goldberg, P.W. Goldberg , Zengjian HU and R.Martin. “Distributed Selfish Load Balancing,” 2 April 2007

[3] Marc H. Willebeek-LeMairand and A.P.Reevers, “Strategies for Dynamic Load Balancing on Highly Parallel Computers,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 9 , pp.979 – 993, September 1993

[4] R.Subrata, A.Y.Zomaya and B.LAndfeldt, “Artificila life techniques for load balancing in computational grids”, *journal of Computer and System Science 73,* pp. 1176 -1190, 2007 [Online]. Available: www.sciencedirect.com

[5] C.Adolphs and P.Berenbrink “Distributed Selfish Load Balancing with Weights and Speeds,” 30 September 2011

[6] E.S Minor, R.I McDonald, E.A Treml nd D.L Urban, “Uncertainly in spatially explicit population models,” *Biological Conservation 141 (2008),* pp.956-970, 7 March 2008

[7] P.Berenbrink, A.Brinkmann, T.Friedetzky and L.Nagel, “Balls into Non-uniform Bins, ” 23 April 2010

[8] P.Berenbrink, M.Hoefer and T.Sauerwald, “Distributed Selfish Load Balancing in Networks,” pp. 1487-1497, 2011

[9] M.Felegyhazi and J.P Hubaux , “Game Theory in Wireless Networks : A Tutorial,” *LCA-REPORT-2006-002* , 21 February 2006

[10] D.Monderer and l.Shapley, “Potential Games,” *Games and Economic Behavior 14,* pp 124-143, 1996

[11] Α. Μανωλάκος, «Διαστρωματικές Τεχνικές Ελέγχου Τοπολογίας για Αποδοτική Ανάθεση Πόρων σε Ασύρματα Αυτοργονούμενα Δίκτυα,» Διπλωματική Εργασία, Εθνικό Μετσόβιο Πολυτεχνείο, 2010.

Παράρτημα Α

Κώδικας Προγράμματος – Αλγόριθμοι

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "MatrixGraph_3.h"
#include "MatrixGraph_3.c"

#define NUM_USERS  100
int main(void)
{
   Usertype users[NUM_USERS];        //array of Users (stoixeia users: x, y, load)
   int i, j, node, u, k, d, p, check, n, algorithm, load;
   MatrixGraph *graph = mgCreate(NUM_OF_NODES);
   int none_dispersal = 0;
   int count_dispersal, none_loop, y_or_n;
   int load_10, capacity, capacity13, capacity1, capacity_central;
   int matrix_load[NUM_OF_NODES];
   double possibility;



   int sort(const void *x, const void *y) {
  return (*(int*)y - *(int*)x);
}

   //initialize matrix-load
   for(i=0; i<graph->numOfVertices; ++i)
   {
      matrix_load[i] = 0;
   }

 printf("Welcome to Load Balance!!!!\n");
```

```
  //create users
   printf("              Stoixeia xristwn:\n");
   printf("Number of Users in Network: %d\n", NUM_USERS);
   printf("Give load of users.\n");
   scanf("%d", &load);

   for(i = 0; i < NUM_USERS; i ++)
   {
    users[i].x = i;
    users[i].y = rand() % NUM_USERS;
    users[i].load = load;
    users[i].node_id = -1;    //user doesn not belong to any node
   }

   mgPrintUsers (users, i);

//Create a 25_node_square graph
          for (i=0; i<NUM_OF_NODES; i++)
          {
             nodes_matrix[i] = i;
          }
 ++i;

   for (j=0; j<NUM_OF_CENTRAL; ++j)
   {
      nodes_matrix[i] = central_square_25[j];
      ++i;
   }

printf("Give capacity of the nodes:\n");
 scanf ("%d", &capacity);
printf("Give capacity of central nodes:\n");
 scanf ("%d", &capacity_central);
printf("Give the possibility of dispersal: 0.1 - 0.9\n");
 scanf("%lf", &possibility);

  //for each nde in graph insert fields
  mgInitialize( graph, capacity, capacity_central);
   assign_node(graph, users, NUM_USERS);
  compute_current_load (graph, users, NUM_USERS);
  printf("          Chose algorithm: 1 OR 2. \n");
  scanf (" %d", &algorithm);
```

```
count_dispersal = 1;

if (algorithm == 1)
{
   //metanasteusi vasi algorithmou apo paper
   printf (" \n\n*************** DISPERSAL METHOD BASED ON 1st ALGORITHM ***************\n\n\n");

   while (!(network_in_balance(graph, 1) )) // ean kapoios apo tous komvous exei perissotero load apo capacity
   {
  for (u=0; u<NUM_USERS; ++u) // gia kathe user
   {
      k = choose_neighbour_with_probability(graph, users, u);
      d = check_for_dispersal(graph, users, k, u ); //elegkse ean o geitonas exei mikrotero load
      if (d == 1)                          //se tetoia periptwsi
      {
         if(check_for_dispersal_2(graph, users, users[u].node_id) == 1) //elekse an o user den eksipireteitai
            count_dispersal ++;
         p = prob_of_dispersal(graph, users[u].node_id, k); //metanasteuse ton me kapoia pithanotita
         if (p == 1)                          //ean metanasteusei auksise to count
         {
             dispersal(graph, users, u, users[u].node_id, k );
             loops_1++;
         }
      }

   } // end of for

   } //end of while
}

else
{
  //metanasteush vasi algorithmou pou dimiourgisame (load > capacity)
   printf (" \n\n*************** DISPERSAL METHOD BASED ON 2nd ALGORITHM ***************\n\n\n");

while (!(network_in_balance(graph, 2) ))
{
 //compute load for every node in every turn
for (n=0; n<graph->numOfVertices; ++n)
```

```
   matrix_load [n] = graph->values[n].current_load;


   //dispersal for users who need
    for (u=0; u<NUM_USERS; u++)
   {
      n= users[u].node_id;
      d = check_for_dispersal_2(graph, users, n);
      if (d == 1)
      {
         printf(" User %d must dispersal! \n", u);
         y_or_n = disp_or_not(possibility);
         // if it does
         if (y_or_n == 1)
         {
            k = choose_neighbour_with_probability(graph, users, u);
            dispersal(graph, users, u, users[u].node_id, k );
         }
      }
   }
}   //end of while

 //compute load for every node in last turn
  for (n=0; n<graph->numOfVertices; ++n)
  matrix_load [n] = graph->values[n].current_load;

}   //end of else

if (none_dispersal)
{
   printf (" In loop %d nobody dispersal!\n", none_loop);
   printf("Count_dispersal: %d\n", count_dispersal);
}

printf("            loop 1: %d\n", loops_1 );
printf("            loop 2: %d\n", loops_2 );

          return 0;
}
```

Παράρτημα Β

Κώδικας Προγράμματος – Συναρτήσεις

```
#include "MatrixGraph_3.h"

//print tha users Matrix
void mgPrintUsers (Usertype array_users[], int i)
{
   int j=0;
   for (j=0; j<i; j++)
   {
      printf("User %d\n", j);
      printf("Topothesia:   %d , %d     ", array_users[j].x, array_users[j].y);
      printf("load user : %d\n\n", array_users[j].load);
   }
}

// create and initialize a new graph
MatrixGraph *mgCreate(int numOfVertices)
{
	MatrixGraph *graph = (MatrixGraph *)malloc(sizeof(MatrixGraph));
	int i = 0;

	if (graph == NULL)
		printf(stderr,"Memory Allocation Error!!!");
	graph->numOfVertices = numOfVertices;

	// Init the adjacency matrix
	graph->matrix = (int **)malloc(numOfVertices * sizeof(int *));
	if (graph->matrix == NULL)
		printf(stderr,"Memory Allocation Error!!!");

	for (i = 0; i < numOfVertices; i++)	{
		int j = 0;
		graph->matrix[i] = (int *)malloc(numOfVertices * sizeof(int));
		if (graph->matrix[i] == NULL)
			printf(stderr,"Memory Allocation Error!!!");
		for (j = 0; j < numOfVertices; j++) {
			graph->matrix[i][j] = 0;
		}
	}
```

```
	// Init the visited array
	graph->visited = (int *)malloc(numOfVertices * sizeof(int));
	if (graph->visited == NULL)
		printf(stderr,"Memory Allocation Error!!!");

	for (i = 0; i < numOfVertices; i++) {
		graph->visited[i] = 0;
	}

	// Init the values array
	graph->values = (Nodetype *)malloc(numOfVertices * sizeof(Nodetype));

	for (i = 0; i < numOfVertices; i++) {
	   graph->values[i].node_id = i;
		graph->values[i].num_of_users = 0;
		graph->values[i].capacity = 0;
		graph->values[i].speed = 0;
		graph->values[i].current_load = 0;
	}

	return graph;
}

// Set an edge between vertices v1 and v2
void mgAddEdge(MatrixGraph *graph, int v1, int v2, int weight)
{
#ifdef _DEBUG
	if (weight <0)
		WARNING("Invalid weight inserted!!!");
#endif

	graph->matrix[v1][v2] = weight;
}



// initialize graph
void mgInitialize(MatrixGraph *graph, int capacity, int capacity_central)
{
   int i;
   for (i=0; i<graph->numOfVertices; ++i)
   {
      graph->values[i].speed = rand() % graph->numOfVertices * 10 +1;
```

```
        if (one_of_central(graph->values[i].node_id))
            graph->values[i].capacity = capacity_central;
        else
            graph->values[i].capacity = capacity;
    }
}


//assign user to node
void assign_node(MatrixGraph *graph, Usertype users[], int numUsers)
{
    int i,j, num, node;
    for (i=0; i<numUsers; i++)
    {
        //pich a node randomly
        j = rand();
        node = j % NUM_OF_NODES;
        users[i].node_id = nodes_matrix[node];
        printf("User %d pick %d to service.\n", i, node);
      graph->values[node].num_of_users++;
    }
}

void assign_node_double_central(MatrixGraph *graph, Usertype users[], int numUsers)
{
    int i,j, num, node;
    for (i=0; i<numUsers; i++)
    {
        //pich a node randomly
        j = rand();
        node = j % NUM_OF_ALL;
        users[i].node_id = nodes_matrix[node];
        graph->values[node].num_of_users++;
    }
}

//compute current load of each node
void compute_current_load (MatrixGraph *graph, Usertype users[], int numUsers)
{
    //for each user in array
        //find the node that belong to (node_id)
        //add to the node user's current_load the load of the user
```

```
   int i, node;
   for (i=0; i<numUsers; i++)
   {
      node= users[i].node_id;
      graph->values[node].current_load+=users[i].load;

   }

}

// check if node n need dispersal
   // where current_load > capacity
int check_for_dispersal_2(MatrixGraph *graph, Usertype *users, int n)
{
   int cur_load, cap;

      cur_load = graph->values[n].current_load;
      cap=graph->values[n].capacity;
      if (cur_load > cap)
         return 1;
      else
         return 0;



}

//check if dispersal is needed in algorothm_2 (n = neighbour, u =user)
int check_for_dispersal (MatrixGraph *graph, Usertype *users, int n, int u)
{
   int i, load_u, load_n;
   int node_u = users[u].node_id;
   if (graph->values[node_u].current_load > graph->values[n].current_load )
      return 1;
   else
      return 0;

}



//return the random neighbour to dispersal the user
int node_dispersal(int *neighbour,  int num_of_neighbours)
```

```
{
   int n, neigh;
   //choose a neighbour randomly
   n = rand();
   //neigh is the position in neighbour[] for the random neighbour, not the id of the neighbour
   neigh = n % num_of_neighbours;
   return neigh;

}

// give the user and find neighbour to dispersal (with probability)
int choose_neighbour(MatrixGraph *graph, Usertype *users, int u)
{
   int  p, n, j, num_of_neighbours, node ;
   int k=0;
   int neighbour_1[graph->numOfVertices];
   //find the node in the graph
   node = users[u].node_id;
   //find all neighbours of node from The adjacency matrix
   for (j=0; j<graph->numOfVertices; j++)
      //if j is neighbour
      if ((graph->matrix[node][j] == 1) || (graph->matrix[j][node] == 1))
      {
         //put in neighbour matrix
         neighbour_1[k]=j;
         k++;
      }

   num_of_neighbours=k;
   //choose a node which is neighbour
   n= node_dispersal(neighbour_1, num_of_neighbours);
   return neighbour_1[n]; //neighbour[n] is the node_id which user u pick to dispersal (with
probability)

}

////////////////////////////////////////////////////////////////

int choose_neighbour_with_probability(MatrixGraph *graph, Usertype *users, int u)
{
   int  c, sum, p, n, j, num_of_neighbours, node, neigh, prob, temp_prob ;
   int k=0;
   int neighbour_1[8];
```

```
   //find the node in the graph
   node = users[u].node_id;
   //find all neighbours of node from The adjacency matrix
   for (j=0; j<graph->numOfVertices; j++)
      //if j is neighbour
      if ((graph->matrix[node][j] == 1) || (graph->matrix[j][node] == 1))
      {
         //put in neighbour matrix
         sum+=graph->values[j].capacity;
         neighbour_1[k]=j;
         k++;
      }

for (p=0; p<k; ++p)
{
 printf("    %d is neighbour of  %d\n", neighbour_1[p] , node);
 neigh = neighbour_1[p];
 c =graph->values[neigh].capacity;
 prob = c/sum;
 temp_prob = rand() % sum;
      if (temp_prob < c)
         return neighbour_1[p];
}

}


////////////////////////////////////////////////////////////////


// give the user and find neighbour to dispersal (with probability)
int choose_neighbour_double_pos(MatrixGraph *graph, Usertype *users, int u)
{
   int  p, n, j, num_of_neighbours, node ;
   int k=0;
   int neighbour_1[graph->numOfVertices];
   //find the node in the graph
   node = users[u].node_id;
   //find all neighbours of node from The adjacency matrix
   for (j=0; j<graph->numOfVertices; j++)
      //if j is neighbour
      if ((graph->matrix[node][j] == 1) || (graph->matrix[j][node] == 1))
      {
```

```
        //put in neighbour matrix
        neighbour_1[k]=j;
        ++k;
        //if it is one of the central then put it again (double time)
        if (one_of_central(j))
        {
           neighbour_1[k]=j;
           ++k;
        }
      }


for (p=0; p<k; ++p)
{
  printf("    %d is neighbour of  %d\n", neighbour_1[p],node);
}


   num_of_neighbours=k;
   //choose a node which is neighbour
   n= node_dispersal(neighbour_1, num_of_neighbours);
   return neighbour_1[n]; //neighbour[n] is the node_id which user u pick to dispersal (with
probability)

}

////////////////////////////////////////////////////////////////
int one_of_central(int j)
{
   int i;
   for (i=0; i < NUM_OF_CENTRAL; ++i)
      if (central_square_25[i] == j)
         return 1;

   return 0;
}


////////////////////////////////////////////////////////////////


// give the user and find neighbour to dispersal (with probability)
int prob_of_dispersal_2(MatrixGraph *graph, Usertype *users, int u)
```

```
{
   int  p, n, j, sum, node, c, prob, temp_prob ;
   int k=0;
   //find the node in the graph
   node = users[u].node_id;
   // find capacity of node
   c = graph->values[node].capacity;
   //find all neighbours of node from The adjacency matrix
   for (j=0; j<graph->numOfVertices; j++)
      //if j is neighbour
      if ((graph->matrix[node][j] == 1) || (graph->matrix[j][node] == 1))
      {
         //add its capacity
        sum+=graph->values[j].capacity;
      }
   //find the fraction
   prob = c/sum;
   //find the propability
   // find a number 0- sum
      temp_prob=rand() % sum;
      if (temp_prob < c)
         return 1;
      else
         return 0;

}

//////////////////////////////////////////////////


int disp_or_not(double possibility)
{
   int prob, temp_prob;
   prob = possibility * 10;
   //find the propability
   // find a number 0- 10
      temp_prob=rand() % 10;
    //  printf("Temp_prob = %d\n", temp_prob);
      if (temp_prob < prob)
         return 1;
      else
         return 0;
```

```
}

//user u dispers from node p to n
int dispersal(MatrixGraph *graph, Usertype *users, int u, int p, int n)
{
      users[u].node_id = n;   //assign user k to node n
      graph->values[n].num_of_users++;
      graph->values[p].num_of_users--;
      graph->values[n].current_load = graph->values[n].current_load + users[u].load;
      graph->values[p].current_load = graph->values[p].current_load - users[u].load;
   return;

}

int prob_of_dispersal(MatrixGraph *graph, int p, int n)
{
   int  probability, temp_prob_2, temp_prob_3;
   double cur_load, neigh_load, temp_prob;
   cur_load = graph->values[p].current_load;
   neigh_load = graph->values[n].current_load;

   // probability of dispersal is 1-Xjb(t)/Xib(t)
      //where Xjb(t) --> current load of resourse j (neighbour node)
      //      Xib(t) --> current load of resourse i  (current resourse/node)
   printf("cur_load: %6.2f\n", cur_load);
    printf("neigh_load: %6.2f\n", neigh_load);
   temp_prob = 1.0 - (neigh_load / cur_load);
   if (temp_prob < 0) //that means Xjb(t) > Xib(t) --> doesn't worth to disperse
      return 0;
   else
   {
   //make a number of temp_prob
      temp_prob_2 = round (temp_prob *10000);

      // find a number 0-10.000
      temp_prob_3=rand() % 10000;
      if (temp_prob_3 < temp_prob_2)
         return 1;
      else
         return 0;
   }
}
```

```
//check whether the network has load balancing
   //which means that in every node load <=capacity
   //if network is in balance return 1
int network_in_balance(MatrixGraph *graph, int algorithm)
{
   int i;
   double cur_load, capacity, temp_prob;

   for (i=0; i < graph->numOfVertices; i++)
   {
      cur_load = graph->values[i].current_load;
      capacity = graph->values[i].capacity;

      if (cur_load > capacity)
      {
         if (algorithm == 1)
         {
         loops_1++;
         printf ("         number of loops for 1st algorithm:  %d\n\n", loops_1);
         return 0;
         }
         else
         {
         loops_2++;
         printf ("         number of loops for 2nd algorithm:  %d\n\n", loops_2);
         return 0;
         }
      }
   }
  // if (i == graph->numOfVertices)
   for (i=0; i < graph->numOfVertices; i++)
   {
      printf("Capacity node %d = %d\n", i, graph->values[i].capacity);
   }
      return 1;
}
```

Παράρτημα Γ

Κώδικας Προγράμματος – Header File

```
#ifndef __MATRIX_GRAPH_H__
#define __MATRIX_GRAPH_H__

#define NUM_OF_CENTRAL 25
#define NUM_OF_NODES 101
#define NUM_OF_ALL 126
extern int loops_1 = 0;
extern int loops_2 = 0;
FILE *f1;


int central_star_101[NUM_OF_CENTRAL] = { 100, 99, 89, 79, 69, 59, 88, 75, 78, 85,
                98, 97, 95, 94, 58, 84, 87, 74, 77, 65,
                62, 67, 55, 57, 68 } ;

int central_star_60[NUM_OF_CENTRAL] = { 9, 19, 29, 39, 49, 50, 51, 52, 53, 54, 55,
                        56, 57, 58, 59, 60} ;

int central_star_31[NUM_OF_CENTRAL] = { 25, 26, 27, 28, 29, 30, 4, 14, 24, 9 } ;



int central_square_100[NUM_OF_CENTRAL] = { 24, 25, 33, 34, 35, 36, 42, 43, 44, 45, 46, 47,
                            52, 53, 54, 55, 56, 57, 63, 64, 65, 66, 73, 74, 75 };

int central_square_49[NUM_OF_CENTRAL] = { 16, 17, 18,  22, 23, 24, 25, 26, 30, 31, 32 };


int central_square_25[NUM_OF_CENTRAL] = { 7, 11, 12, 13, 17 };



int nodes_matrix[NUM_OF_ALL];


#include <stdio.h>
```

```
#include <stdlib.h>
#include <assert.h>
#include <math.h>

/**
 * Represent a Graph as an Adjacent Matrix.
 */
 typedef struct _Nodetype {
   int node_id;
   int num_of_users;
   int capacity;
   int current_load;
   int speed;
}Nodetype;

typedef struct _Usertype {
   int num;              //number_of_user
   int x;               //position x
   int y;               //position y
   int load;             //weight of task
   int node_id;          //which node belongs to
}Usertype;



typedef struct _MatrixGraph {
        int **matrix;                   // The adjacency matrix (pinakas geitniasis )
        int numOfVertices;              // The number of graph nodes
        int *visited;                   // Indicated if  node was visited during a search or not
        Nodetype *values;               // The values for each node
}MatrixGraph;

int network_in_balance(MatrixGraph *graph, int algorithm);
int prob_of_dispersal(MatrixGraph *graph, int p, int n);
int prob_of_dispersal_2(MatrixGraph *graph, Usertype *users, int u);
int disp_or_not(double possibility);
void dispersal_2(MatrixGraph *graph, Usertype *users, int u, int p, int n );
int check_for_dispersal (MatrixGraph *graph, Usertype *users, int n, int u);
int choose_neighbour(MatrixGraph *graph, Usertype *users, int u);
int choose_neighbour_with_probability(graph, users, u);
int choose_neighbour_double_pos(MatrixGraph *graph, Usertype *users, int u);
int dispersal(MatrixGraph *graph, Usertype *users, int k, int i, int n);
int one_of_central(int j);
int check_for_dispersal_2(MatrixGraph *graph, Usertype *users, int n);
```

```
void compute_current_load (MatrixGraph *graph, Usertype users[], int numUsers);
void assign_node(MatrixGraph *graph, Usertype users[], int numUsers);
void assign_node_double_central(MatrixGraph *graph, Usertype users[], int numUsers);

//print coordinates and load of users
void mgPrintUsers (Usertype array_users[], int i);

//initialize nodes of graph with capacity and speed
//void mgInitialize(MatrixGraph *graph, int c);
void mgInitialize(MatrixGraph *graph, int capacity13, int capacity1);

// create and initialize a new graph
MatrixGraph *mgCreate(/*GRAPHTYPE type,*/ int numOfVertices);

// Delete the graph
void mgDelete(MatrixGraph *graph);

// Set an edge with specified weight between vertices v1 and v2
// If the graph is directed, then the edge is from v1 to v2
// If the graph is undirected, the edge is unidirectional
void mgAddEdge(MatrixGraph *graph, int v1, int v2, int weight);

// print the adjacency matrix
void mgPrint(MatrixGraph *graph);


#endif // __MATRIX_GRAPH_H__
```