ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΣΥΓΚΡΙΤΙΚΗ ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ ΕΞΙΣΟΡΡΟΠΗΣΗΣ ΦΟΡΤΟΥ ΜΕ NODEJS ΚΑΙ WEBSOCKETS

Μπόβολος Χριστόφορος

Αθήνα, 2021

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

Τριμελής Εξεταστική Επιτροπή

Τσερπές Κωνσταντίνος

Επίκουρος Καθηγητής, Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο

Μιχαήλ Δημήτριος

Επίκουρος Καθηγητής, Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο

Αναγνωστόπουλος Δημοσθένης

Καθηγητής, Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο

Ο/Η Μπόβολος Χριστόφορος

δηλώνω υπεύθυνα ότι:

1) Είμαι ο κάτοχος των πνευματικών δικαιωμάτων της πρωτότυπης αυτής εργασίας και από όσο γνωρίζω η εργασία μου δε συκοφαντεί πρόσωπα, ούτε προσβάλει τα πνευματικά δικαιώματα τρίτων.

2) Αποδέχομαι ότι η ΒΚΠ μπορεί, χωρίς να αλλάξει το περιεχόμενο της εργασίας μου, να τη διαθέσει σε ηλεκτρονική μορφή μέσα από τη ψηφιακή Βιβλιοθήκη της, να την αντιγράψει σε οποιοδήποτε μέσο ή/και σε οποιοδήποτε μορφότυπο καθώς και να κρατά περισσότερα από ένα αντίγραφα για λόγους συντήρησης και ασφάλειας.

Αφιέρωση

Αφιερώνω την εργασία αυτή στους γονείς μου και στην κοπέλα μου,που με στήριξαν σε αυτή μου την προσπάθεια.

Γνωμικό

Αδικεί πολλάκις ο μη ποιών τι, ου μόνον ο ποιών τι.

- **Πολλές φορές είναι άδικος αυτός που δεν κάνει κάτι, όχι μόνο αυτός που κάνει κάτι.**

Μάρκος Αυρήλιος, 121-180 μ.Χ., Ρωμαίος Αυτοκράτωρ ("Τα εις εαυτόν" θ' 5)

Ευχαριστίες

Η εκπόνηση της συγκεκριμένης πτυχιακής εργασίας δεν θα ήταν δυνατή χωρίς την καθοδήγηση, ενδιαφέρον και την βοήθεια την οποία προσέφερε ο Καθηγητής του Τμήματος Πληροφορικής και Τηλεματικής κ. Κωνσταντίνος Τσερπές. Τον ευχαριστώ ιδιαίτερα για την συνεχή βοήθεια που μου παρείχε, την αμέριστη προσφορά του στην προσπάθεια μου καθώς και για την καθοδήγηση και την εμπιστοσύνη που μου έδειξε κατα την εκπόνηση της πτυχιακής μου εργασίας.

Περιεχόμενα

Περίληψη **8**

Abstract **9**

Κατάλογος Εικόνων **10**

Κατάλογος Πινάκων **11**

1. Εισαγωγή **11**

2. Ανάλυση **13**

3. Προσέγγιση **15**

4. Αποτελέσματα **23**

Βιβλιογραφία **31**

Περίληψη

Οι εξισορροπητές φόρτου είναι προγράμματα ή/και συσκευές οι οποίες διανέμουν τον φόρτο εργασίας είτε προγραμμάτων είτε διαδικτύου σε ένα πλήθος από εξυπηρετητές (server). Ο σκοπός της εργασίας ήταν να μελετήσει διάφορους αλγορίθμους που μπορούν να χρησιμοποιηθούν για διαμοιρασμό φόρτου, καθώς και να τους συγκρίνει μεταξύ τους έτσι ώστε να βρεθεί ο καταλληλότερος αλγόριθμος για κάθε σενάριο.

Για το σκοπό αυτό σχεδιάστηκε και υλοποιήθηκε ένα βασικό σενάριο χωρίς εξισορρόπηση φόρτου για να παρομοιάσω την λειτουργία ενός server που δεν χρησιμοποιεί εξισορροπητές με την χρήση της NodeJs , βρίσκοντας έτσι τον μέγιστο αριθμό αιτημάτων που μπορεί να διαχειριστεί χωρίς να υπάρχουν καθυστερήσεις, και στην συνέχεια αφού συνέδεσα με τον server μας έναν αριθμό clients με την βοήθεια του Socket.io, εφαρμόσαμε τρεις διαφορετικούς αλγορίθμους εξισορρόπησης για να παρατηρήσω ποιός λειτουργεί καλύτερα και σε ποιες προϋποθέσεις. Για να το πετύχουμε αυτό, χρησιμοποιήσαμε το Apache JMeter, στέλνωντας έναν συγκεκριμένο αριθμό GET request στον main server μέσα σε μια περίοδο 10 δευτερολέπτων, αυξομειώνοντας τον αριθμό των αιτημάτων σε κάθε σενάριο.

Οι αλγόριθμοι που μελετήθηκαν ήταν ο Round Robin [10], Least Connections και ο Ant Colony σε 5 διαφορετικά σενάρια, ο Ant Colony μελετήθηκε σε 2 διαφορετικές παραλλαγές. Κατά μέσο όρο ο Round Robin ήταν ο αλγόριθμος που είχε την καλύτερη απόδοση και στα 5 διαφορετικά σενάρια, καθώς οι υπόλοιποι αλγόριθμοι παρουσίασαν μεγάλη χρονική απόκλιση μεταξύ των server, με αποτέλεσμα κάποιοι server να εξυπηρετούν τα αιτήματα τους πολύ γρήγορα, ενώ άλλοι με μεγάλη καθυστέρηση.

Λέξεις κλειδιά: [Εξισορρόπηση φόρτου, αλγόριθμος, Round Robin, Least Connections, Ant Colony]

Abstract

Load Balancers are either programs or hardware devices whose purpose is to distribute network or application traffic to other servers. The purpose of this degree thesis was to implement and compare different algorithms that can be used in a load balancing scenario, in order to find what's the most appropriate one and in which case it would be best to use it.

We created a basic scenario without utilizing any load balancer, in order to simulate how a normal server who does not use load balancer would perform with the help of NodeJs, this way i was able to find out how many requests a server could receive before delays could occur. Afterwards, after creating a load balancer and connecting a number of clients to it, using Socket.io, we implemented 3 different algorithms in order to observe which algorithm would perform better and in which case, to achieve this, we send a varying amount of GET requests in the span of 10 seconds through Apache JMeter, tuning the amount of requests each time, so i can get valuable information.

The algorithms that we studied were Round Robin, Least Connections and Ant Colony in 5 different scenarios, with Ant Colony being studied in 2 different variants. Across all the scenarios, Round Robin was the one with the most consistent results between the 5 scenarios, while the other algorithms presented great discrepancies between each scenario, with some servers resolving requests too fast while others delayed considerably to resolve the requests in their pool.

Keywords: [Load Balancers, algorithms, Round Robin, Ant Colony, Least Connections]

Κατάλογος Εικόνων

Εικ.1.1 Εκκίνηση εξισορροπητή……..……..……σ.20

Εικ.1.2 Εκκίνηση εξυπηρετητή……...……..……σ.20

Εικ.1.3 Σύνδεση εξυπηρετητή στον εξισορροπητή……..……..……σ.20

Εικ.1.4 Αποτέλεσμα μέτρησης……..……σ.22

Εικ.2.1 Ρύθμιση Apache JMeter (μερος 1)……...……..……σ.21

Εικ.2.2 Ρύθμιση Apache JMeter (μερος 2)……....……σ.21

Κατάλογος Πινάκων

Πίν.1: Μετρήσεις χωρίς εξισορρόπηση……..……..σ.24

Πίν.2:Παράδειγμα πίνακα μετρήσεων με 3 κόμβους……..…σ.25

Πίν.3: Πίνακας μετρήσεων με 3 κόμβους……..……..σ.25

Πίν.4:Παράδειγμα πίνακα μετρήσεων με 5 κόμβους……..…σ.27

Πίν.5: Πίνακας μετρήσεων με 5 κόμβους……..……..σ.28

1. Εισαγωγή

Ο σκοπός αυτής της εργασίας είναι να συγκρίνουμε διαφορετικούς αλγορίθμους σε ένα σενάριο εξισορρόπησης φόρτου [9], χρησιμοποιώντας Node.js και Socket.io, καταγράφοντας την απόδοσή τους και έπειτα συγκρίνοντάς τους ώστε να αντλήσουμε συμπεράσματα ως προς την καταλληλότητα τους για κάθε σενάριο.

Ένας εξισορροπητής φόρτου αναλαμβάνει να μοιράσει τον φόρτο που θα αναλάμβανε κανονικά ένα μηχάνημα σε διαφορετικά επιμέρους μηχανήματα, έχει ως σκοπό δηλαδή να χωρίσει μια εργασία σε μικρότερα κομμάτια και να τα μεταβιβάσει την επίλυση τους σε άλλα μηχανήματα, προσπαθώντας πάντοτε να το κάνει όσο πιο αποτελεσματικά μπορεί, δίνοντας ιδιαίτερο βάρος στην ομοιογένεια της κατανομής και τον χρόνο απόκρισης κάθε μηχανήματος που συμμετέχει στην διαδικασία [1]. Οι εξισορροπητές χωρίζονται σε 2 κατηγορίες, στατικούς και δυναμικούς. Οι στατικοί αλγόριθμοι αναλαμβάνουν να χωρίσουν το φορτίο έχοντας γνώση για τους εξισορροπητές πριν ξεκινήσει η διαδικασία και χρησιμοποιώντας ότι γνωρίζουν ήδη, ενώ οι δυναμικοί [2] χρησιμοποιούν και άλλα δεδομένα τα οποία προκύπτουν κατά την διαδικασία εξισορρόπησης. Οι εξισορροπητές φόρτου χρησιμοποιούνται ευρέως από μικρές και μεγάλες εφαρμογές καθώς αποτελούν μία από τις μεθόδους που εξασφαλίζουν σταθερότητα στην λειτουργία τους.

Ο τρόπος που προσεγγίσαμε το συγκεκριμένο πρόβλημα, ήταν να δημιουργήσουμε έναν εξυπηρετητή που θα λαμβάνει, καταγράφει και αποθηκεύει αιτήματα και στην συνέχεια θα υπολογίζει τον χρόνο τον οποίο χρειάστηκε για να τα διεκπαιρεώσει. Η συγκεκριμένη μεθοδολογία χρησιμοποιήθηκε επίσης και για την δημιουργία ενός σεναρίου που δεν χρησιμοποιούμε εξισορροπητή φόρτου, το οποίο αποτελεί το βασικό σενάριο σύγκρισης της εφαρμογής μας.

Αφού έχουμε εγκαθιδρύσει το βασικό μας σενάριο, δημιουργήσαμε έναν εξισορροπητή ο οποίος λαμβάνει τα αιτήματα και τα διανέμει αναλόγως με τον ποιόν αλγόριθμο χρησιμοποιούμε, αυξάνοντας σταδιακά τον αριθμό των αιτημάτων ώστε να μελετήσουμε ακόμη καλύτερα την συμπεριφορά τους

Θεωρούμε πως αυτό το project αποτελεί ένα καλό σημείο αναφοράς για άτομα τα οποία ενδιαφέρονται να μελετήσουν την λειτουργία εξισορροπητών φόρτου. Ο τρόπος ο

οποίος έχει αναπτυχθεί, επιτρέπει την δημιουργία επιπλέον αλγορίθμων πέρα από αυτούς που έχουν υλοποιηθεί, αποτελεί επομένως όχι μόνο μια μελέτη για την συμπεριφορά των συγκεκριμένων αλγορίθμων, αλλά και ως μία βάση πάνω στην οποία μπορούν να δημιουργηθούν περαιτέρω σενάρια καθώς και να ελεγχθούν ακόμη περισσότερες υποθέσεις.

Οι μεγαλύτερες δυσκολίες που αντιμετωπίστηκαν κατά την υλοποίηση, ήταν ο τρόπος λειτουργίας της Node.js και της Javascript γενικότερα, ο παραλληλισμός καθώς και οι περιορισμένες δυνατότητες υπολογισμού τιμών ήταν τα κύρια προβλήματα, συνεπώς οι λύσεις οι οποίες δόθηκαν έχουν σχεδιαστεί με τέτοιο τρόπο ώστε να είτε να αποφύγουν αυτά τα θέματα είτε ακόμη και να τα χρησιμοποιήσουν για μια καλύτερη υλοποίηση.

2. Ανάλυση

Για να μελετήσουμε και να υλοποιήσουμε τα σενάρια μας, χρησιμοποιήσαμε διάφορα εργαλεία ανοιχτού κώδικα τα οποία προμηθευτίκαμε από το Node package manager. Το Node package manager είναι ένα εργαλείο το οποίο χρησιμοποιείται ευρέως από developers για να κάνουν bundle και να διανείμουν τον κώδικά τους σε μορφή πακέτων.

Ο εξυπηρετητής που χρησιμοποιούμε για να δεχτούμε τα αιτήματα, δημιουργήθηκε με την βοήθεια της Node.js και της Express.js. Η Node.js είναι ένα περιβάλλον ανοιχτού κώδικα, το οποίο χρησιμοποιεί Javascript και βασίζεται στην μηχανή V8 που αναπτύχθηκε από την Google, και επιτρέπει την εκτέλεση Javascript έξω από έναν περιηγητή διαδικτύου, ενώ η Express.js είναι ένα σύνολο εργαλείων που καθιστά την δημιουργία εφαρμογών και API στην Node.js πολύ ευκολότερη. Η σύνδεση μεταξύ εξισορροπητών επιτεύχθηκε με την βοήθεια του Socket.io, μια βιβλιοθήκη βασισμένη σε Javascript, η οποία επιτρέπει την ανταλλαγή δεδομένων αμφίδρομα και σε πραγματικό χρόνο.

Για την εξομοίωση της λειτουργίας αποστολής αιτημάτων που δέχεται ο εξισορροπητής μας, επιλέξαμε το εργαλείο Apache JMeter, το οποίο μας επιτρέπει να δημιουργήσουμε διαφορετικά σενάρια με μια σχετική ευκολία. Βρήκαμε το συγκεκριμένο εργαλείο να ταιριάζει απόλυτα στην εφαρμογή μας, καθώς επιτρέπει την δημιουργία

διαφόρων προφίλ καθώς και την αδιάλειπτη εναλλαγή μεταξύ τους για την χρήση σε κάθε σενάριο.

Τα προβλήματα που αντιμετωπίστηκαν με τα συγκεκριμένα εργαλεία, εντοπίζονται κυρίως στον τρόπο λειτουργίας της Node.js. Η Node.js λειτουργεί σε ένα νήμα, καθώς παρόμοια άλλα τέτοια εργαλεία λειτουργούν σε περισσότερα, παρόλα αυτα, η Node.js μας προμηθεύει με άλλες σημαντικές λειτουργίες, όπως ασύγχρονη επεξεργασία εργασιών Input/Output στο μοναδικό νήμα της, εκμεταλλεύοντας την λειτουργία του callback functionality της Javascript καθώς και του Event Loop της Node.js.

Αυτήν την στιγμή υπάρχουν πολλές εφαρμογές οι οποίες παρέχουν λειτουργίες εξισορρόπησης φόρτου σαν service [4] καθώς και προγράμματα[5] τα οποία ασχολούνται με μεγαλύτερη ευλάβεια στο κομμάτι της εξισορρόπησης δίνοντας επιλογές στον χρήστη ώς προς ποιους αλγορίθμους μπορεί να χρησιμοποιήσει, με την πλειοψηφία τους να ασχολείται με εξισορρόπηση φόρτου σε περιβάλλον cloud [1], ενώ φυσικά μπορούμε να αναλύσουμε την εξισορρόπηση φόρτου και σε περιβάλλον intranet, οι διαφορές που παρουσιάζονται ανάλογα με την απόσταση των κόμβων χωράει διαφορετική προσέγγιση σχετικά με την τοποθεσία τον κόμβων, στην υλοποίηση μας έχουμε θεωρήσει ότι η εξισορρόπηση λαμβάνει χώρα σε ένα περιβάλλον cloud και για αυτόν τον λόγο έχουμε δημιουργήσει τεχνητές δυσκολίες όπως ο χρόνος απόκρισης μεταξύ των κόμβων [6].

Σημαντική αναφορά γίνεται και στην πολυπλοκότητα [7] ενός αλγορίθμου που χρησιμοποιείται , καθώς όσο πιο σύνθετος είναι ο αλγόριθμος που χρησιμοποιούμε τόσο μεγαλύτερη απώλεια έχουμε στην αποτελεσματικότητα και μεγαλύτερη πιθανότητα προβλημάτων αντιμετωπίζουμε.

Τα περισσότερα συστήματα εξισορρόπησης φόρτου κατανέμουν το μεγαλύτερο κομμάτι της διαδικασίας σε επιμέρους κομμάτια ώστε να αποφύγουν την πιθανότητα σφάλματος και να μειώσουν τον φόρτο στον κύριο εξισορροπητή, ακολουθούν επομένως ένα σύστημα distributed cloud load balancing όπου ο κύριος εξισορροπητής διατηρεί πληροφορίες για τους εξυπηρετητές σε ένα άλλο μηχάνημα και τις ανακαλεί με την βοήθεια Distributed Hash Tables (DHTs) [8]. Στην συγκεκριμένη υλοποίηση δεν υπάρχει παρόμοια προσέγγιση, καθώς ο κύριος εξισορροπητής μας είναι υπεύθυνος να καταγράφει και να μελετά τους εξυπηρετητές παράλληλα με την διαδικασία εξισορρόπησης.

Αναφορικά με την υλοποίηση, υπάρχουν αρκετοί τρόποι προσέγγισης ενός συστήματος εξισορρόπησης το οποίο αναλύεται εκτενώς τόσο στη βιβλιογραφία [12][13] όσο και σε πραγματικά σενάρια [4] τα οποία αντιμετωπίζουμε καθημερινά. Εντοπίζονται σενάρια στα οποία χρησιμοποιούμε εξισορρόπηση σε διάφορα σημεία του OSI Model, όπως στο επίπεδο 3 του δικτύου [14] καθώς και στο επίπεδο 7 με τη βοήθεια των DNS. Εμείς θα μελετήσουμε την προσέγγιση με τη βοήθεια εξυπηρετητών σε ένα σενάριο παρόμοιο με αυτό ενός χρήστη που χρησιμοποιεί τον παγκόσμιο ιστό. Αυτή η υλοποίηση μας παρέχει την δυνατότητα να διαχειριζόμαστε με σχετική ευκολία τον φόρτο της εργασίας που ανατίθεται στον εξισορροπητή και έπειτα να την διαμοιράσουμε αναλόγως στον χρήστη που επιθυμούμε.

Η κατηγοριοποίηση των αλγορίθμων σε στατικούς και δυναμικούς μας βοηθάει να αναλύσουμε τα θετικά και τα αρνητικά και των δυο αντίστοιχα. Η μικρότερη πολυπλοκότητα και ο μικρός αριθμός υπολογισμών που επιτυγχάνεται με την βοήθεια των στατικών αλγορίθμων μπορεί πολλές φορές να προσδίδει μεγαλύτερο προβάδισμα σε έναν αλγόριθμο που αναλύει πεπερασμένο αριθμό δεδομένων και ασχολείται με μηχανήματα των οποίων οι συνθήκες δεν μεταβάλλονται σε σημαντικό βαθμό, καθώς και η ακρίβεια των δεδομένων που χρησιμοποιούνται αρκετές φορές στην υλοποίηση δυναμικών αλγορίθμων δεν συνάδουν με τα πραγματικά λόγω καθυστέρησης [15][16], δεδομένου της ερευνητικής φύσεως της συγκεκριμένης εργασίας, αποφασίσαμε να προσεγγίσουμε την διαδικασία με την βοήθεια ενός στατικού αλγορίθμου (Round Robin) και δύο δυναμικών (Least Connections, Ant Colony).

3. Προσέγγιση

Η υλοποίηση της εφαρμογής βρίσκεται στο προσωπικό μου github στην διεύθυνση https://github.com/WottatoParrior/Load-Balancer, μπορεί κάποιος να κάνει clone το συγκεκριμένο project και αφού εγκαταστήσει τα απαραίτητα πακέτα με την βοήθεια του npm install να ακολουθήσει τις οδηγίες που θα παρουσιαστούν στο τέλος του κεφαλαίου.

Το πρώτο και πιο σημαντικό πράγμα που έπρεπε να υλοποιήσουμε ήταν η δημιουργία ενός server που θα αποτελούσε το βασικό μας σενάριο. Όπως αναφέρθηκε και πρωτύτερα, χρησιμοποιήσαμε την Node.js και Express.js για να να δημιουργήσουμε τον server .

Χρησιμοποιώντας την Node.js , δημιουργούμε τον server της εφαρμογής μας με την Express και τον βάλαμε να ακούει στην πόρτα 4000. Επιπλέον χρησιμοποιούμε και βοηθητικές μεθόδους οι οποίες χρησιμοποιούνται και σε περισσότερες υλοποιήσεις, ώστε να μειώσουμε την επαναληψιμότητα και να αυξήσουμε την ορατότητα.

Δημιουργούμε επίσης δύο μεταβλητές, pool και freshstart. Στην pool αποθηκεύουμε τα αιτήματα τα οποία δεν επεξεργαζόμαστε εκείνη την στιγμή ενώ το freshstart είναι μια μεταβλητή τύπου boolean η οποία τίθεται true by default κάθε φορά που ξεκινάει ένα instance του server μας. Ο κύριος λόγος της freshstart είναι να ξεκινάει τον χρονομετρητή μας, έτσι ώστε κάθε φορά που ο server μας λαμβάνει το πρώτο του αίτημα γίνεται false και ξεκινάει ο μετρητής μας με την εντολή console.time("time").

Μόλις ο server μας λάβει ένα αίτημα τύπου GET στο μονοπάτι / μέσω του Apache JMeter, η συνάρτηση ξεκινάει, στέλνωντας μια απάντηση ότι έλαβε το αίτημα και ότι ο server μας λειτουργεί, στη συνέχεια προσθέτουμε ένα αίτημα στην λίστα pool δίνοντας μαζί τον χρόνο δημιουργίας του, καθώς και ένα υποτυπώδες string για να παρομοιάσουμε ένα μήνυμα που θα λάμβανε ο server μας, προφανώς στην συγκεκριμένη περίπτωση δεν είναι αναγκαίο να κάνουμε κάτι τέτοιο καθώς δεν διαχειριζόμαστε το περιεχόμενο των δεδομένων, ωστόσο επέλεξα να το εμπεριέχω.

Έως τώρα λοιπόν έχουμε δημιουργήσει έναν server που δέχεται αιτήματα και τα αποθηκεύει, το επόμενο βήμα είναι να εξομοιώσουμε την λειτουργία επίλυσης των αιτημάτων και αφαίρεσης τους από την λίστα. Επιλέξαμε ο χρόνος επίλυσης να κυμαίνεται μεταξύ 400 και 700 ms, δεδομένου ότι η μελέτη μας δεν χρειάζεται ακριβή δεδομένα για να λειτουργήσει καθώς μελετά σχεσιακά την απόδοση κάθε αλγορίθμου, θα μπορούσαμε να χρησιμοποιήσουμε διαφορετικούς αριθμούς, σε περίπτωση που χρειαστεί να μελετήσουμε αλγορίθμους που λαμβάνουν υπόψη τον χρόνο απόκρισης, θα πρέπει να δείξουμε μεγαλύτερη προσοχή στους συγκεκριμένους αριθμούς.

Με την βοήθεια της συνάρτησης getRandomInRange , παίρνουμε έναν αριθμό στο εύρος τιμών που το προμηθεύουμε στη συνέχεια στην removeElementFromPool.

Εδώ αντιμετωπίζουμε την ιδιοσυγκρασία της Node.js σε σχέση με τα νήματα, έπρεπε να βρεθεί ένας τρόπος να λαμβάνουμε και να αφαιρούμε αιτήματα ταυτόχρονα κρατώντας υπόψιν ότι η ίδια διαδικασία θα ακολουθηθεί και στους εξισορροπητές, η λύση δόθηκε με την

βοήθεια της αναδρομής και του setTimeout. Η συνάρτηση ξεκινάει θέτοντας ένα timeout, η setTimeout είναι μια μέθοδος η οποία επιτρέπει σε συναρτήσεις να εκτελούνται μετά από ένα προκαθορισμένο χρονικό διάστημα , επομένως εάν το μέγεθος του pool μας είναι μεγαλύτερο του μηδέν, αφαιρούμε το πρώτο στοιχείο από την λίστα μας, το οποίο αντιπροσωπεύει το πρώτο αίτημα, έπειτα δημιουργούμε ένα νέο τυχαίο χρόνο μεταξυ 400 και 700 ms και τρέχουμε την removeElementFromPool, χάρη στην λειτουργία του Event Loop της Node.js, τοποθετείται στην κορυφή της λίστας προς εκτέλεση και με την βοήθεια των callback διεκπεραιώνεται στον αναμενόμενο χρόνο, η συγκεκριμένη διαδικασία επαναλαμβάνεται όσο η λίστα μας έχει μέγεθος μεγαλύτερο του μηδέν. Εάν η λίστα έχει μέγεθος μηδέν, αυτό αποδεικνύει ότι τα αιτήματα μας διεκπαιρεώνονται γρηγορότερα απο τον ρυθμό που τα δεχομαστε ή δεν έχουμε αλλα αιτηματα να διαχειριστούμε και καλούμε την console.timeEnd για να σταματήσουμε τον χρονομετρητή και να βρούμε το χρόνο που πέρασε.

Η συγκεκριμένη προσέγγιση χρησιμοποιείται κάθε φορά που κάποιος εξισορροπητής θέλει να επιλύσει ένα αίτημα. Μπορούμε τώρα να προχωρήσουμε στο επόμενο βήμα της υλοποίησης, το οποίο είναι η σύνδεση των εξισορροπητών και η μεταβίβαση των αιτημάτων σε αυτούς.

Όπως μπορούμε να παρατηρήσουμε, έχουμε ενσωματώσει την βιβλιοθήκη Socket.io, για να επιτρέψουμε την αμφίδρομη επικοινωνία σε πραγματικό χρόνο μεταξύ των εξισορροπητών μας, έχουμε θεωρήσει ότι ο συγκεκριμένος server θα αναλάβει την εξισορρόπηση των αιτημάτων και οι υπόλοιποι server θα δέχονται τα αιτήματα που θα τους προμηθεύει.

Όσον αφορά τις μεταβλητές, έχουμε προσθέση τις activeNode, pheromoneNode, nodeWithLeastConnections, leastConnections καθώς και μία λίστα clients, στην οποία προσθέτουμε κάθε server που συνδέεται στον εξισορροπητή μας, επιτρέποντας μας έτσι να έχουμε όσους clients χρειαζόμαστε και το πρόγραμμα μας να λειτουργεί χωρίς να γνωρίζει τον ακριβή αριθμό client επιτρέποντας του να λειτουργεί με εναν, δέκα ακόμη και εκατό ταυτόχρονες συνδέσεις χωρίς να αλλάζει ο τρόπος λειτουργίας του.

Έχουμε ρυθμίσει τον κύριο εξισορροπητή μας να ακούει σε ένα γεγονός connection, έτσι κάθε φορά που κάποιος client συνδέεται, η συνάρτηση ενεργοποιείται. Όταν συνδεθεί ο

client, θα τον προσθέσουμε στην λίστα με τους client μαζί με άλλες πληροφορίες όπως τον αριθμό των συνδέσεων του, που θα χρησιμοποιήσουμε στην συνέχεια.

Κάθε φορά που συνδέεται ένας client, εμφανίζουμε στην κονσόλα ένα μήνυμα με το αναγνωριστικό του και όταν αποσυνδεθεί εμφανίζουμε ένα μήνυμα αποσύνδεσης.

Είναι πολύ σημαντικό για τον κεντρικό εξισορροπητή μας να παρακολουθεί συνεχώς την κατάσταση των υπολοίπων client και για αυτόν τον λόγο έχουμε υλοποιήσει μια συνάρτηση η οποία τρέχει κάθε 500 ms η οποία στέλνει ένα σήμα ελέγχου από τον κεντρικό εξισορροπητή στους clients. Θα ήταν βοηθητικό σε αυτό το σημείο να εξηγήσουμε τον τρόπο λειτουργίας του κάθε client καθώς και να επισημάνουμε πως όλοι οι clients είναι υλοποιημένοι με τον ίδιο ακριβώς τρόπο, με καμία διαφορά μεταξυ τους.

Όπως είδαμε και νωρίτερα, ο τρόπος με τον οποίο είναι δομημένος ο client μας είναι ο ίδιος με τον οποίο είχαμε υλοποιήσει όταν είχαμε έναν server χωρίς την διαδικασία της εξισορρόπησης, είναι επίσης σημαντικό να παρατηρήσουμε ότι η πόρτα στην οποία συνδεόμαστε είναι η ίδια η οποία έχουμε ορίσει και στην διαδικασία δημιουργίας του server μας.

Μερικές βασικές συναρτήσεις ενημέρωσης έχουν δημιουργηθεί, οι οποίες μας ενημερώνουν πότε συνδέεται και πότε αποσυνδέεται ένας client. Όταν ένας client λάβει ένα γεγονός τύπου message από το βασικό εξισορροπητή, τοποθετεί αυτό το αίτημα στην λίστα pool και σε περίπτωση που είναι το πρώτο αίτημα ξεκινάμε τον χρονομετρητή μας μέχρι η λίστα να μείνει άδεια. Η μεγαλύτερη διαφορά υλοποίησης σε σχέση με το σενάριο που δεν έχουμε εξισορροπητή είναι η προσθήκη ενός event listener, του make_passive_check, έτσι μόλις ένας client λάβει ένα αίτημα, απαντάει με ένα γεγονός response_of_check μαζί με κάποιες επιπλέον πληροφορίες, όπως το μέγεθος που έχει εκείνη την στιγμή, το οποίο λαμβάνει ο βασικός εξισορροπητής μας για να ενημερώσει τις πληροφορίες του συγκεκριμένου client στην λίστα pool του.

Δεδομένου ότι ο βασικός κορμός του προγράμματός μας είναι έτοιμος, το μονο που μένει είναι η υλοποίηση, εφαρμογή και σύγκριση των αλγορίθμων.

Ο πρώτος αλγόριθμος που υλοποιήθηκε ήταν ο Round Robin, κατά τον οποίο ξεκινάμε από τον πρώτο κόμβο και κάθε φορά που λαμβάνουμε ένα request μετακινούμαστε στον επόμενο. Μόλις φτάσουμε στον τελευταίο κόμβο, η διαδικασία ξεκινάει από την αρχή.

Όπως μπορεί κανείς να παρατηρήσει και στην υλοποίηση, η συνάρτηση δέχεται έναν κόμβο, στην αρχή αυτός ο κόμβος παίρνει την θέση 0, έπειτα αυξάνει την θέση κατά 1 και ελέγχει εάν ο κόμβος ο οποίος βρισκόμαστε είναι ο τελευταίος, και σε αυτην την περίπτωση ξεκινάει πάλι από την αρχή, τέλος επιστρέφει έναν κόμβο ο οποίος θα ανατεθεί στην global μεταβλητή activeNode, βάση της οποίας επιλέγεται ο επόμενος κόμβος που θα είναι υπεύθυνος να δεχθεί το επόμενο request.

Ο επόμενος αλγόριθμος ο οποίος εφαρμόστηκε, ήταν ο Least Connections. Βάση του Least Connections, ο κόμβος ο οποίος ανατίθεται στην μεταβλητή activeNode κάθε φορά, είναι αυτός με τον μικρότερο αριθμό αντικειμένων στο pool του. Χρησιμοποιώντας την updateSocketInfo, έχουμε συνεχή ενημέρωση για την κατάσταση των αιτημάτων στους κόμβους μας.

Στη συγκεκριμένη υλοποίηση ελέγχουμε τους κόμβους μας έναν-έναν με την σειρά και βρίσκουμε τον κόμβο με τις λιγότερες συνδέσεις συγκρίνοντας κάθε φορά τον αριθμό συνδέσεων κάθε κόμβου με τον κόμβο με τις λιγότερες. Η διαδικασια αυτη περιγράφεται στην παραπάνω υλοποίηση.

Ο τελευταίος αλγόριθμος που χρησιμοποιήθηκε στις συγκρίσεις μας, είναι ο Ant Colony Optimization.

Στο συγκεκριμένο αλγόριθμο, ο διαμοιρασμός φόρτου ξεκινούσε πάντα στον πρώτο κόμβο, μόλις ο κόμβος έφτανε σε ένα όριο αιτημάτων και ήταν υπερφορτωμένος, τότε το επόμενο αίτημα το διαχειρίζεται ο επόμενος κομβος. Συνεχώς κρατάμε την θέση του κόμβου στην οποία βρισκόμαστε και με αυτόν τον τρόπο δεν χρειάζεται να ελεγχουμε από την αρχή όλους τους κόμβους την επόμενη φορά που θα δεχτούμε αίτημα. Υπάρχει φυσικά ένα σενάριο στο οποίο όλοι οι κόμβοι μας έχουν τον πλήρη αριθμό αιτημάτων που τους επιτρέπεται, συνεπώς εδώ επιλέξαμε να ακολουθήσουμε την υλοποίηση του Least Connections, ώστε να τηρήσουμε τις αρχές του Ant Colony.

Είναι σημαντικό να προσθέσουμε εδώ ότι έχουν υλοποιηθεί 2 διαφορετικές παραλλαγες του συγκεκριμένου αλγορίθμου, η πρώτη παραλλαγή υποθέτει ότι ο κάθε κόμβος υπερφορτώνεται όταν έχει να διαχειριστεί 20 αιτήματα και ο δεύτερος 15, ο συγκεκριμένος διαχωρισμός έγινε, ώστε να παρατηρήσουμε την απόδοση του συγκεκριμένου αλγορίθμου σε 2 διαφορετικά σενάρια και να παρατηρήσουμε σε ποιό σενάριο θα είχαμε τις καλύτερες τιμές. Ο τρόπος που το πετυχαίνουμε αυτό, είναι περνώντας σαν μεταβλητή στην συνάρτηση έναν αριθμό, η μεταβλητή ON (Overload Node) δίνει στην συνάρτηση τον μέγιστο αριθμό συνδέσεων που μπορεί να δεχτεί πριν θεωρηθεί overloaded.

Ολοκληρώνεται με αυτόν τον τρόπο η υλοποίηση για τον έλεγχο του κάθε αλγορίθμου, συνεπώς το μόνο που χρειάζεται για να ελέγξουμε τους αλγορίθμους μας, είναι να κάνουμε comment in την κατάλληλη γραμμή ώστε να ενεργοποιήσουμε τον εκάστοτε αλγόριθμο. Αφού έχουμε επιλέξει ποιόν αλγόριθμο θέλουμε να χρησιμοποιήσουμε, χρησιμοποιούμε την εντολή της Node.js node για να τρέξουμε το κύριο script, το οποίο ονομάζεται Balancer.js. Μόλις τρέξουμε το script, θα εμφανιστεί ένα μήνυμα το οποίο θα μας ενημερώσει αν έχουμε ενεργοποιήσει την λειτουργία Node Balancing, για παράδειγμα στην Εικ. 1.

```
PS C:\Users\Christopher Bovo\Desktop\Σχολη\Ptuixakh> node .\Balancer.js
Load Balancing is turned true
```

(Εικ. 1.1)

Στην συνέχεια, θα τρέξουμε τον αριθμό των server που θέλουμε, για να αποκτήσουμε πληθώρα αποτελεσμάτων, χρησιμοποιήσαμε σε μια εκτέλεση 3 server και σε άλλην 5, συνεπώς αφού ανοίξουμε 3 ή 5 διαφορετικά terminal αναλόγως, τρέχουμε την εντολή node .\Server.js.

```
PS C:\Users\Christopher Bovo\Desktop\Σχολη\Ptuixakh> node .\Server1.js
Connected server 1
```

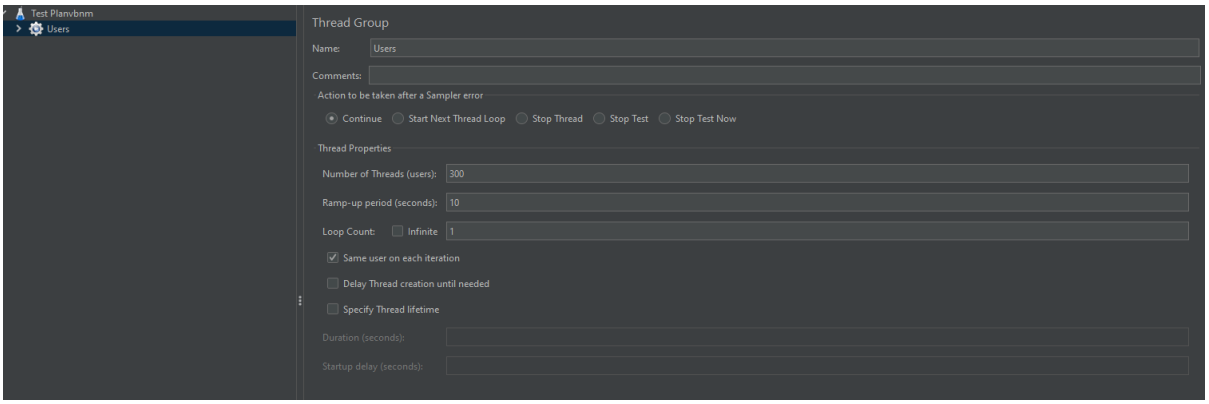
(Εικ. 1.2)

Παράλληλα στο terminal στο οποίο έχουμε ξεκινήσει τον Node Balancer, εμφανίζεται το παρακάτω μήνυμα, το οποίο μας ενημερώνει πως ένας σέρβερ έχει συνδεθεί στον Node Balancer καθώς και το αναγνωριστικό του

```
PS C:\Users\Christopher Bovo\Desktop\Σχολη\Ptuixakh> node .\Balancer.js
Load Balancing is turned true
Connected with client TgHcfRf36lKmXeD5AAAB
```

(Εικ. 1.3)

Αφού έχουμε συνδέσει τον αριθμό server που επιθυμούμε να χρησιμοποιήσουμε, θα χρησιμοποιήσουμε το Apache JMeter για να προσομοιώσουμε την λειτουργία των request, παρακάτω βλέπουμε πως θα καταφέρουμε αυτήν την διαδικασία



(Εικ. 2.1)

Το Number of threads , ορίζει τον αριθμό τον αιτημάτων που θα προσομοιώσει το Apache JMeter, ενώ το Ramp-up Period φανερώνει τον αριθμό των δευτερολέπτων μέσα στα οποία αυτά τα αιτήματα θα δημιουργηθούν, επομένως απο την παραπάνω εικόνα μπορούμε να συμπεράνουμε ότι θα δημιουργηθούν 300 αιτήματα μέσα σε μια περίοδο 10 δευτερολέπτων.

Στο επόμενο βήμα θα ορίσουμε το μέρος που θα κατευθύνουμε τα αιτήματα μας, αυτή η διαδικασία φανερώνεται στην παρακάτω οθόνη

(Εικ. 2.2)

Βλέπουμε ότι μπορούμε να ορίσουμε τον τύπο πρωτοκόλλου, την IP που θα χρησιμοποιήσουμε καθώς και το port στο οποίο θα συνδεθούμε. Επιπλέον θα πρέπει να θέσουμε τον τύπο αιτήματος που θα δημιουργήσουμε καθώς και σε ποιο path θα κατευθυνθεί. Στην υλοποίηση που ακολούθησα έχω θέσει το port 4000 σαν το ενεργό όταν όρισα τον web server μας και αναγνωρίζουμε τα αιτήματα στο Path “/”, επίσης όλες οι μετρήσεις έγιναν στο τοπικό μας μηχάνημα και για αυτόν τον λόγο η διεύθυνση του server μας είναι 127.0.0.1.

Έχοντας ορίσει όλες τις παραμέτρους μας και έχουμε σιγουρευτεί ότι έχουμε ξεκινήσει τον Balancer, τότε το μόνο που απομένει είναι να ξεκινήσουμε τον αριθμό σέρβερ που

επιθυμούμε και να ξεκινήσουμε την διαδικασία των αιτημάτων μέσω του Apache. Μόλις η λίστα των αιτημάτων του κάθε εξυπηρετητή αδειάσει, θα μας ενημερώσει στην κονσόλα με τον χρόνο τον οποίο χρειάστηκε για να διευθετήσει όλα τα αιτήματα.

```
PS C:\Users\Christopher Bovo\Desktop\Σχολη\Ptuixakh> node .\Server1.js
Connected server 1
time: 150728.837ms
```

(Εικ. 1.4)

Χρησιμοποιώντας την μεθοδολογία που περιγράφηκε έως τώρα, θα μετρήσουμε την απόδοση κάθε διαφορετικού αλγορίθμου σε διαφορετικά σενάρια. Εκτελέσαμε τον αλγόριθμο 3 φορές και κρατήσαμε μετρήσεις για κάθε κόμβο ξεχωριστά καθώς θέλουμε να γνωρίζουμε πόση ώρα χρειάζεται ο κάθε κόμβος για να διεκπαιρεώσει τα αιτήματα.

4. Αποτελέσματα

Συνολικά, τρέξαμε οκτώ διαφορετικά σενάρια με διαφορετικό αριθμό αιτημάτων για κάθε αλγόριθμο τρεις φορές το καθένα, τρία από τα οποία έγιναν με τρεις εξυπηρετητές και τα υπόλοιπα πέντε με πέντε εξυπηρετητές. Σε όλα τα σενάρια, η πρώτη δοκιμή έγινε χωρίς την βοήθεια εξισορροπητή φόρτου, ώστε να συμπαιράνουμε πως θα λειτουργούσε ένα σύστημα χωρίς αυτόν, συνεπώς λόγω αποφυγής επαναληψιμότητας τρέξαμε το σενάριο χωρίς εξισορροπητή πέντε φορές, καθώς δεν υπάρχει διαφοροποίηση μεταξύ του αριθμού των εξισορροπητών.

Τα αποτελέσματα για την διαδικασία ελέγχου χωρίς εξισορροπητή φαίνονται παρακάτω στον Πιν. 1. Σε κάθε στήλη παρουσιάζεται ο αριθμός των αιτημάτων που δεχόμαστε μέσα στην περίοδο των 10 sec που έχουμε θεσπίσει στο Apache JMeter

Πιν. 1, Single Node

	20 (requests in 10 seconds)	50 (requests in 10 seconds)	75 (requests in 10 seconds)	150 (requests in 10 seconds)	300 (requests in 10 seconds)
Χρόνος διεκπεραίωσης αιτημάτων στην 1η,2η και 3η επανάληψη (seconds)	10.48 10.79 10.67	25.37 25.88 25.92	38.97 37.67 38.92	75.36 77.32 76.74	153.21 152.18 152.35

Όπως μπορεί να παρατηρήσει κανείς ο χρόνος διεκπεραίωσης αιτημάτων είναι αναλογικός με τον αριθμό τον αιτημάτων που δέχεται ο εξισορροπητής μας στην περίπτωση που δεν χρησιμοποιεί κάποιον αλγόριθμο. Οι μικρές διαφοροποιήσεις δημιουργούνται λόγω του τυχαίου παράγοντα επιλογής χρόνου από την συνάρτηση getRandomInRange.

Είναι σημαντικό να επισημάνουμε εδώ, ότι στα 10 αιτήματα παρατηρείται ότι ένας μοναδικός εξυπηρετητής τα διεκπεραιώνει οριακά στο χρονικό διάστημα στο οποίο εμφανίζονται, συνεπώς μετά απο τα 10 αιτήματα παρατηρείται δυσκολία στην αντιμετώπιση των αιτημάτων έγκαιρα. Εδώ είναι και το σημείο στο οποίο παρουσιάζεται η χρησιμότητα των εξισορροπητών που θα δούμε παρακάτω.

Θα μελετήσουμε τώρα την απόδοση των αλγορίθμων σε σενάριο με τρεις εξυπηρετητές, οι μετρήσεις διατίθενται στον Πιν. 3. Σε περίπτωση που ο αλγόριθμος έχει

διεκπεραιώσει όλα τα αιτήματα στο pool του πριν το πέρας των 10 δευτερολέπτων, επισημαίνεται με τα γράμματα OT, τα οποία σημαίνουν On Time. Στην περίπτωση του αλγορίθμου Ant Colony, εμφανίζουμε με τους χαρακτήρες OV20 και OV15 την υλοποίηση στην οποία θεωρούμε έναν εξυπηρετητή υπερφορτωμένο στα 20 αιτήματα με OV20 (Overload at 20) και αντίστοιχα στα 15 αιτήματα με OV15 (Overload at 15).

Ο Πιν. 3 έχει την μορφή που παρουσιάζεται στον Πιν. 2

Πιν. 2

	Αριθμός request σε περίοδο 10 seconds
Όνομα αλγορίθμου	• Χρόνος διεκπεραίωσης του κόμβου 1 στην 1η επανάληψη/κομβου 2 στην 1η επανάληψη/κομβου 3 στην 1η επανάληψη • Χρόνος διεκπεραίωσης του κόμβου 1 στην 2η επανάληψη/κομβου 2 στην 2η επανάληψη/κομβου 3 στην 2η επανάληψη • Χρόνος διεκπεραίωσης του κόμβου 1 στην 3η επανάληψη/κομβου 2 στην 3η επανάληψη/κομβου 3 στην 3η επανάληψη

Πιν. 3

	20 αιτήματα	**50 αιτήματα**	**75 αιτήματα**
Round Robin	OT/OT/OT OT/OT/OT OT/OT/OT	OT/OT/OT OT/OT/OT OT/OT/OT	13.02/13.18/13.35 13.17/13.23/13.49 13.34/13.61/13.22
Least Connections	OT/OT/OT OT/OT/OT OT/OT/OT	11.23/OT/OT OT/OT/OT 10.68/OT/OT	12.69/13.23/13.41 14.61/13.51/12.12 13.10/13.24/12.93

Ant Colony OV20	10.72/OT/OT 10.48/OT/OT 10.72/OT/OT	17.79/OT/OT 16.89/OT/OT 17.18//OT/OT	OT/15.64/14.23 15.25/14.90/OT 13.74/11.40/14.78
Ant Colony OV15	10.80/OT/OT 10.83/OT/OT 10.79/OT/OT	11.83/13.81/OT 13.12/12.73/OT 13.41/13.15/OT	17.18/11.97/10.59 17.02/11.98/10.43 15.85/10.84/12.4 1

Θα ξεκινήσουμε τις παρατηρήσεις μας με το γεγονός ότι η διαφορά μεταξύ των σεναρίων χωρίς εξισορροπητή και με εξισορροπητή είναι πολύ μεγάλη, καθώς ακόμη και με 3 μόνο εξυπηρετητές στα 75 αιτήματα έχουμε σχεδόν την ίδια απόδοση με αυτην στο σενάριο χωρις εξισορροπητή στα 20 αιτήματα με τη χρήση αλγορίθμου Round Robin, όσον αφορά τον αλγόριθμο Round Robin, όπως φαίνεται και στον πίνακα, ο αλγόριθμος συμπεριφέρεται πολύ αποδοτικά στα σενάρια με 20 και 50 αιτήματα και ο κάθε κόμβος έχει διεκπαιρεώσει τα αιτήματα του πριν το πέρας των 10 δευτερολέπτων, το οποίο σημαίνει ότι τα αιτήματα διεκπαιρεώνονται την στιγμή που έρχονται. Στο σενάριο των 75 αιτημάτων παρατηρείται ένας παρόμοιος χρόνος απόκρισης μεταξύ όλων των κόμβων ο οποίος κυμαίνεται μεταξύ 13.02 και 13.61 δευτερολέπτων, συνεπώς υπάρχει μία μέγιστη απόκλιση περίπου 0.59 δευτερολέπτου μεταξύ των 3 κόμβων και στις 3 επαναλήψεις.

Στην συνέχεια έχουμε τον αλγόριθμο Least Connections, ενώ παρουσιάζει καλά αποτελέσματα στο σενάριο των 20 αιτημάτων, εμφανίζονται μικρές καθυστερήσεις στα 50 αιτήματα σε μερικούς κόμβους, ενώ στο σενάριο των 75 αιτημάτων παρόλο που πετυχαίνουμε χρόνους μικρότερους απο τον Round Robin, υπάρχει μεγαλύτερη απόκλιση μεταξύ των ακραίων τιμών μας 12.69 και 14.61, το οποίο δεν συνιστά μια καλή υλοποίηση ενός εξισορροπητή.

Στον Ant Colony με overload στα 20 αιτήματα ανά κόμβο, παρατηρούμε μικρές καθυστερήσεις ακόμη και στο αρχικό σενάριο των 20 αιτημάτων, καθώς ουσιαστικά ο μόνος κόμβος που χρησιμοποιείται είναι ο πρώτος μιας και τα αιτήματα που δεχόμαστε είναι όλα και όλα 20, στα 50 αιτήματα παρατηρείται ακόμη μεγαλύτερος χρόνος απόκρισης στον 1ο κόμβο

που δέχεται τα αιτήματα, ενώ στην περίπτωση των 75 αιτημάτων τα αποτελέσματα δεν παρουσιάζουν κάποιο μοτίβο αλλά αντιθέτως έχουνε μεγάλες αποκλίσεις μεταξύ τους, καθώς κόμβοι έχουν διεκπαιρεώσει τα αιτήματα τους και άλλοι βρίσκονται στο όριο τους.

Παρόμοια συμπεριφορά παρατηρείται και στον Ant Colony με overload στα 15 αιτήματα, το σενάριο των 20 αιτημάτων είναι φαινομενικά ίδιο με αυτό του Ant Colony OV20, ενώ στα 50 αιτήματα ο Ant Colony OV15 έχει καλύτερη απόδοση από τον Ant Colony OV 20 καθώς υπερφορτώνεται γρηγορότερα και διαμοιράζει το φορτίου του καλύτερα στους κόμβους. Δυστυχώς στο τελευταίο σενάριο με τα 75 αιτήματα ο Ant Colony OV15 παρουσιάζει μεγάλο χρόνο διεκπαιρέωσης στον 1ο του κόμβο και πολύ μικρό στους υπόλοιπους 2, γεγονός το οποίο τον καθιστά μη επαρκή σε ένα σενάριο εξισορρόπησης φόρτου.

Εν κατακλείδι, οι αλγόριθμοι με την καλύτερη απόδοση στα σενάρια των 15,20 και 75 αιτημάτων με 3 κόμβους ήταν ο Round Robin και ο Least Connections, παρόλα αυτά είναι σημαντικό να επισημάνουμε την μεγαλύτερη απόκλιση στα αποτελέσματα του Least Connection, καθώς χρησιμοποιούμε μικρούς αριθμούς αιτημάτων και κόμβων στο συγκεκριμένο παράδειγμα, το πρόβλημα μπορεί να είναι μεγαλύτερο όταν αυξηθούν και οι απαιτήσεις μας. Είναι ενδιαφέρον λοιπόν να αναλύσουμε περαιτέρω τους αλγόριθμους σε περισσότερο ακραία σενάρια καθώς και τον τρόπο που θα τα αντιμετωπίσουν, για αυτό το λόγο επαναλάβαμε τις μετρήσεις μας αλλα αυτήν την φορά με 5 κόμβους σε κάθε σενάριο και προσθέσαμε 2 ακόμη σενάρια το ένα με 150 αιτήματα και το δεύτερο με 300 αιτήματα, τα αποτελέσματα των μετρήσεων αυτών εμφανίζονται στον παρακάτω Πιν. 5, ο οποίος ακολουθεί την μορφή του Πιν.4.

Πιν. 4

	Αριθμός request σε περίοδο 10 seconds

Όνομα αλγορί-θμου	● Χρόνος διεκπεραίωσης του κόμβου 1 στην 1η επανάληψη/κομβου 2 στην 1η επανάληψη/κομβου 3 στην 1η επανάληψη/κόμβου 4 στην 1η επανάληψη/κόμβου 5 στην 1η επανάληψη ● Χρόνος διεκπεραίωσης του κόμβου 1 στην 2η επανάληψη/κομβου 2 στην 2η επανάληψη/κομβου 3 στην 2η επανάληψη/κόμβου 4 στην 1η επανάληψη/κόμβου 5 στην 1η επανάληψη ● Χρόνος διεκπεραίωσης του κόμβου 1 στην 3η επανάληψη/κομβου 2 στην 3η επανάληψη/κομβου 3 στην 3η επανάληψη/κόμβου 4 στην 1η επανάληψη/κόμβου 5 στην 1η επανάληψη

Πιν. 5

	20 αιτήματα	**50 αιτήματα**	**75 αιτήματα**	**150 αιτήματα**	**300 αιτήματα**
Round Robin	ΟΤ/ΟΤ/ΟΤ/ΟΤ/ΟΤ ΟΤ/ΟΤ/ΟΤ/ΟΤ/ΟΤ ΟΤ/ΟΤ/ΟΤ/ΟΤ/ΟΤ	ΟΤ/ΟΤ/ΟΤ/ΟΤ/ΟΤ ΟΤ/ΟΤ/ΟΤ/ΟΤ/ΟΤ ΟΤ/ΟΤ/ΟΤ/ΟΤ/ΟΤ	ΟΤ/ΟΤ/ΟΤ/ΟΤ/ΟΤ ΟΤ/ΟΤ/ΟΤ/ΟΤ/ΟΤ ΟΤ/ΟΤ/ΟΤ/ΟΤ/ΟΤ	15.95/15.94/15.62/15.89/15.92 15.75/15.37/16.07/15.83/15.46 15.79/15.63/15.43/16.01/15.82	31.20/31.37/31.68/30.42/31.50 31.79/30.69/30.58/30.99/31.07 31.12/31.55/30.93/30.49/31.19
Least Connections	ΟΤ/ΟΤ/ΟΤ/ΟΤ/ΟΤ ΟΤ/ΟΤ/ΟΤ/ΟΤ/ΟΤ ΟΤ/ΟΤ/ΟΤ/ΟΤ/ΟΤ	ΟΤ/ΟΤ/ΟΤ/ΟΤ/ΟΤ ΟΤ/ΟΤ/ΟΤ/ΟΤ/ΟΤ ΟΤ/ΟΤ/ΟΤ/ΟΤ/ΟΤ	ΟΤ/ΟΤ/ΟΤ/ΟΤ/ΟΤ ΟΤ/ΟΤ/ΟΤ/ΟΤ/ΟΤ ΟΤ/ΟΤ/ΟΤ/ΟΤ/ΟΤ	14.63/13.54/16.65/16.46/16.75 15.77/15.41/16.12/15.85/15.39 15.71/15.62/14.60/15.28/16.74	34.24/31.66/30.06/30.69/28.79 31.60/28.95/33.01/29.02/32.25 29.25/30.47/33.5/30.17/31.16
Ant Colony OV20	10.52/ΟΤ/ΟΤ/ΟΤ/ΟΤ	16.53//ΟΤ/ΟΤ/ΟΤ/ΟΤ	15.49/15.07/ΟΤ/ΟΤ/ΟΤ	21.41/15.84/13.30/15.05/12.46	28.30/35.88/31.57/31.83/26.88

	10.54/OT/OT/OT/OT 10.59/OT/OT/OT/OT	17.18/OT/OT/OT/OT 17.12/OT/OT/OT/OT	14.51/15.67/OT/OT/OT 15.20/15.81/OT/OT/OT	15.39/15.60/13.15/20.68/12.56 14.10/13.68/15.27/22.88/12.65	33.79/26.44/29.76/34.71/30.73 35.76/28.15/32.98/28.91/28.68
Ant Colony OV15	10.01/OT/OT/OT/OT 10.79/OT/OT/OT/OT 10.80/OT/OT/OT/OT	12.66/12.84/OT/OT/OT 13.33/13.10/OT/OT/OT 12.78/12.89/OT/OT/OT	11.10/12.31/11.50/OT/OT 12.31/10.81/11.76/12.30/OT 12.88/10.64/11.18/OT/OT	15.50/11.73/16.92/14.24/17.33 12.69/17.93/16.17/14.79/15.99 12.35/18.04/15.5/15.99/16.22	30.08/29.54/30.90/33.07/30.85 29.37/28.94/29.10/29.65/36.77 30.27/27.90/28.70/32.25/33.97

Όπως μπορεί να διαπιστώσει κανείς και απο τον παραπάνω πίνακα τα αποτελέσματα σε μεγαλύτερο αριθμό αιτημάτων δεν αναιρούν τα προηγούμενα μας συμπεράσματα, αντιθέτως τα επιβεβαιώνουν ακόμη περισσότερο, καθώς οι χρονικές αποκλίσεις μεταξύ του ελαχίστου και μεγίστου χρόνου απόκρισης του κάθε εξυπηρετητή είναι ακόμη μεγαλύτερες.

Ο Round Robin παρουσιάζει αποτελέσματα σε ένα εύρος τιμών 30.42 με 31.79, γεγονός που αποδεικνύει μια δίκαιη κατανομή φόρτου σε όλους τους εξυπηρετητές, ενώ ο Least Connections παρουσιάζει ακόμη μεγαλύτερη διαφορά στις ακραίες τιμές του 34.24 και 28.79.Όσον αφορά τις παραλλαγές του Ant Colony, παρόμοιο μοτίβο με τις προηγούμενες μετρήσεις συναντάται και εδώ, με καθυστερήσεις ακόμη και σε πολύ μικρό αριθμό αιτημάτων και ακόμη μεγαλύτερες αποκλίσεις στις ακραίες τιμές του όταν ο αριθμός τον αιτημάτων αυξάνεται.

Καταφέραμε επομένως με την εφαρμογή που παρουσιάζεται να προσομοιώσουμε πλήρως την λειτουργία ενός εξισορροπητή που δέχεται αιτήματα και τα διαμοιράζει σε πολλαπλούς εξυπηρετητές ανάλογα με τον αλγόριθμο που έχει επιλέξει. Έχοντας ολοκληρώσει την διαδικασία των συγκρίσεων και έχοντας όλα τα αποτελέσματα διαθέσιμα μπορούμε να καταλήξουμε σε ορισμένα συμπεράσματα. Η βασική παρατήρηση που προκύπτει είναι ότι η απόδοση του Round Robin είναι καλύτερη από τους υπόλοιπους αλγορίθμους σε οποιονδήποτε αριθμό αιτημάτων επιλέξουμε, ανεξαρτήτως του αριθμού εξυπηρετητών που χρησιμοποιούμε, σε αντίθεση με τον αλγόριθμο Ant Colony, ο οποίος παρουσιάζει μια αδυναμία όσον αφορά την εξισορρόπηση φόρτου και στις 2 παραλλαγές που επιλέξαμε να ακολουθήσουμε, τέλος, ο Least Connections αν και παράγει αποτελέσματα

που δεν απέχουν πολύ από αυτά του Round Robin, παρουσιάζει μια τάση να αποδίδει χειρότερα όταν ο αριθμός των αιτημάτων αυξάνεται, γεγονός που τον καθιστά χειρότερο από τον Round Robin σε όλα τα σενάρια που επιλέξαμε να αναλύσουμε.

Όπως αναφέρθηκε και στην αρχή, η παρούσα εφαρμογή είναι μια βασική υλοποίηση η οποία μπορεί να λειτουργήσει ως κορμός και πάνω της να αναπτυχθούν ακόμη περισσότερες λειτουργίες. Ο ενδιαφερόμενος αφού αποκτήσει πρόσβαση στον κώδικα θα μπορέσει να υλοποιήσει διαφορετικούς αλγορίθμους και να τους συγκρίνει με τους ήδη υπάρχων, ένα παράδειγμα το οποίο θα παρουσίαζε ενδιαφέρον προς μελέτη, είναι ο αλγόριθμος του Least Response Time. Με τον Least Response Time, θα μπορούσαμε να ορίσουμε εμείς χειροκίνητα τον χρόνο απόκρισης κάθε εξυπηρετητή, γεγονός το οποίο θα άνοιγε την πόρτα σε μία πληθώρα ακόμη συγκρίσεων, όπως τον ιδανικότερο αλγόριθμο σε ένα σενάριο όπου έχουμε μεγάλες αποκλίσεις σε χρόνο απόκρισης εξυπηρετητών, ορίζοντας τον χρόνο απόκρισης μπορούμε να απαντήσουμε σε περισσότερα ερωτήματα όπως ποιο είναι το καταλληλότερο πλήθος εξυπηρετητών και ποιος ο αλγόριθμος που θα ήταν προτιμότερο να χρησιμοποιηθεί, υπάρχει κάποιο σημείο απόκρισης το οποίο να δικαιολογούσε αλλαγή αλγορίθμου καθώς και άλλα.

Ενδιαφέρον θα έδειχνε, να εξετάσει κάποιος την υλοποίηση του συγκεκριμένου προγράμματος σε ένα περιβάλλον με Docker, όπου αναλόγως του πλήθους των αιτημάτων μπορούμε να δημιουργήσουμε containers για να καλύψουμε την αυξημένη ζήτηση καθώς και να μειώσουμε τον αριθμό τους σε περίπτωση που έχουμε μικρότερη.

Τέλος, για την ευκολότερη πρόσβαση στον συγκεκριμένο κώδικα το μόνο που απομένει είναι η δημιουργία ενός πακέτου που τον εμπεριέχει για καλύτερη διανομή του και την διάθεση του μέσω npm έτσι ώστε οποιοσδήποτε επιθυμεί να μπορέσει να επαναλάβει τα πειράματα τα οποία πραγματοποιήσαμε, δίνοντας την δυνατότητα στον οποιοδήποτε με πρόσβαση σε ένα μηχάνημα να ολοκληρώσει τα πειράματα και να επιβεβαιώσει τα συμπεράσματα στα οποία προκύψαμε.

Βιβλιογραφία

1. Randles, M., D. Lamb and A. Taleb-Bendiab, “A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing,” in Proc. IEEE 24th International Conference on

Advanced Information Networking and Applications Workshops (WAINA), Perth, Australia, April 2010.

2. Rimal, B. Prasad, E. Choi and I. Lumb, "A taxonomy and survey of cloud computing systems." In proc. 5th International Joint Conference on INC, IMS and IDC, IEEE, 2009.
3. Zhang, Z. and X. Zhang, "A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation." In proc. 2nd International Conference on. Industrial Mechatronics and Automation (ICIMA), IEEE, Vol. 2, May 2010.
4. Arvindhan, M and Anand, Abhineet, Scheming an Proficient Auto Scaling Technique for Minimizing Response Time in Load Balancing on Amazon AWS Cloud (March 15, 2019). International Conference on Advances in Engineering Science Management & Technology (ICAESMT) - 2019, Uttaranchal University, Dehradun, India
5. X. Chi, B. Liu, Q. Niu and Q. Wu, "Web Load Balance and Cache Optimization Design Based Nginx under High-Concurrency Environment," 2012 Third International Conference on Digital Manufacturing & Automation, 2012, pp. 1029-1032, doi: 10.1109/ICDMA.2012.241.
6. Buyya R., R. Ranjan and RN. Calheiros, “InterCloud: Utility-oriented federation of cloud computing environments for scaling of application services,” in proc. 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP), Busan, South Korea, 2010.
7. Grosu, D., A.T. Chronopoulos and M. Leung, "Cooperative load balancing in distributed systems," in Concurrency and Computation: Practice and Experience, Vol. 20, No. 16, pp: 1953-1976, 2008.
8. Ranjan R., Zhao L., Wu X., Liu A., Quiroz A., Parashar M. (2010) Peer-to-Peer Cloud Provisioning: Service Discovery and Load-Balancing. In: Antonopoulos N., Gillam L. (eds) Cloud Computing. Computer Communications and Networks. Springer, London.
9. M. Armbrust,A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report No. UCB/EECS-2009-28, University of California at Berkley, USA, Feb. 10, 2009.
10. Zhong Xu, Rong Huang, "Performance Study of Load Balancing Algorithms in Distributed Web Server Systems", CS213 Parallel and Distributed Processing Project Report.
11. P. L. McEntire, J. G. O'Reilly, and R. E. Larson, Distributed Computing: Concepts and Implementations. New York: IEEE Press, 1984.

12. V. Cardellini, M. Colajanni and P. S. Yu, "Dynamic load balancing on Web-server systems," in *IEEE Internet Computing*, vol. 3, no. 3, pp. 28-39, May-June 1999, doi: 10.1109/4236.769420.

13. T.Bourke, “Server Load Balancing”, 1st Edition,Book ISBN : 0-596-00050-2, 2001.

14. D. Puccinelli and M. Haenggi, "Arbutus: Network-Layer Load Balancing for Wireless Sensor Networks," 2008 IEEE Wireless Communications and Networking Conference, 2008, pp. 2063-2068, doi: 10.1109/WCNC.2008.366.

15. Y. Zhang, H. Kameda, S.L. Hung, Comparison of dynamic and static load-balancingstrategiesinheterogeneousdistributedsystems,IEEProc.Comput.Digit.Tech.144 (2)(1997)100–106

16. Satish Penmatsa, Anthony T. Chronopoulos, Game-theoretic static load balancing for distributed systems, Journal of Parallel and Distributed Computing, Volume 71, Issue 4,