Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Ανάκτηση Πληροφορίας και Εξισορρόπηση Φορτίου σε Υπολογιστικά Νέφη**
Ονοματεπώνυμο Φοιτητή	**Χρήστος Κοψιδάς**
Πατρώνυμο	**Γεράσιμος**
Αριθμός Μητρώου	**ΜΠΣΠ/13056**
Επιβλέπων	**Χρήστος Δουληγέρης, Καθηγητής**

Ημερομηνία Παράδοσης **Οκτώβριος 2016**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Χρήστος Δουληγέρης Καθηγητής	Δέσποινα Πολέμη Αναπληρωτής Καθηγητής	Παναγιώτης Κοτζανικολάου Επίκουρος Καθηγητής

Ευχαριστίες

Πάρα πολλές ευχαριστίες στον επιβλέποντα καθηγητή κ. Χρήστο Δουληγέρη και στον υποψήφιο διδάκτορα κ. Δημήτρη Καλλέργη για τις σημαντικές υποδείξεις και οδηγίες καθώς και για την άριστη συνεργασία που είχαμε κατά τη διάρκεια της εκπόνησης της μεταπτυχιακής διατριβής.

Θα ήθελα τέλος να ευχαριστήσω όλους όσοι συνέβαλαν, με ανέχτηκαν και με βοήθησαν όλα αυτά τα χρόνια ώστε να πετύχω αυτά που έχω κατορθώσει έως τώρα.

Περιεχόμενα

Ευρετήριο Εικόνων

Ευρετήριο Πινάκων

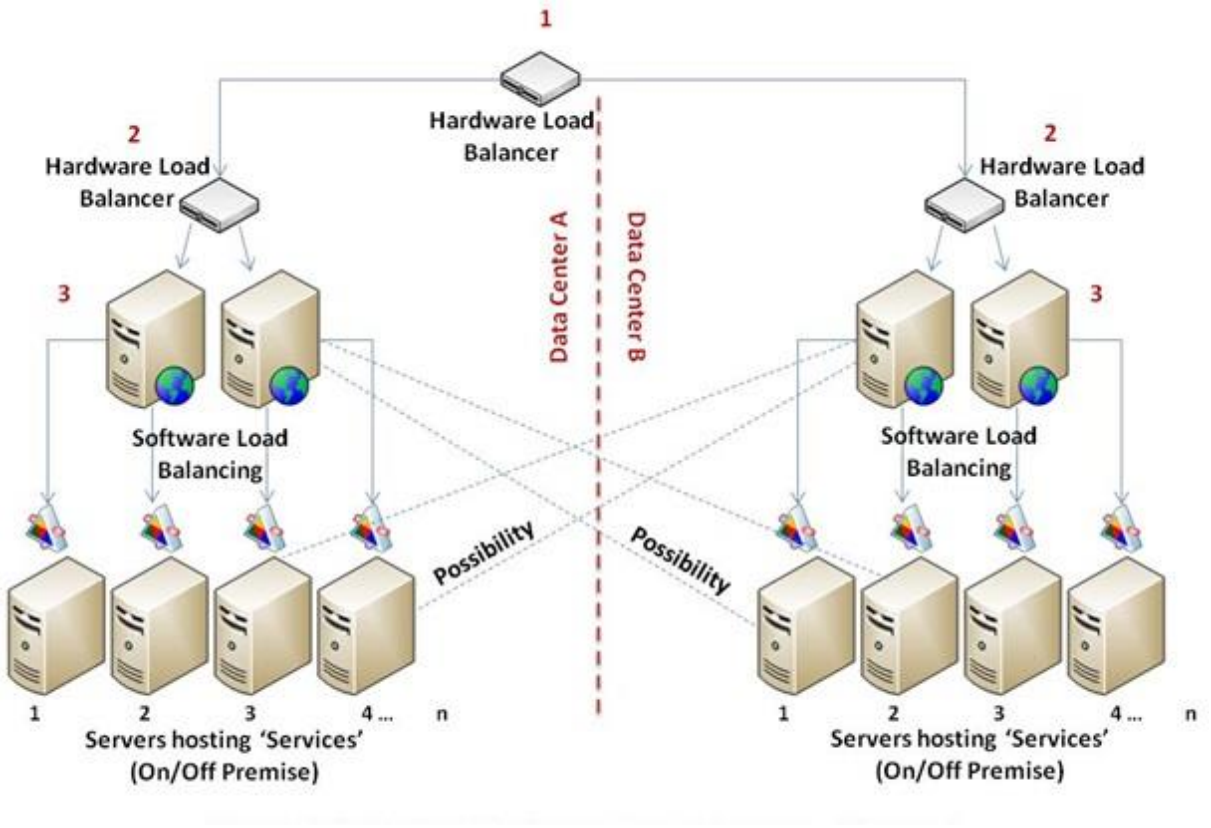
1. ΕΙΣΑΓΩΓΗ

1.1 Πρόλογος

Η παρούσα εργασία εντάσσεται στο πεδίο των υπηρεσιοκεντρικών αρχιτεκτονικών και πιο συγκεκριμένα αφορά την *Υπολογιστική Νέφους* ή, διαφορετικά, τα *Υπολογιστικά Νέφη* (Cloud Computing). Κίνητρα για τη δημιουργία και την ανάπτυξη αυτού του τομέα της Πληροφορικής είναι, μεταξύ άλλων, η μείωση του κόστους χρήσης για τον χρήστη, καθώς και η δημιουργία εφαρμογών που χρησιμοποιούνται από πολλά εκατομμύρια χρήστες σε ολόκληρο τον κόσμο, οι οποίες δε θα μπορούσαν να είναι λειτουργικές σε τόσο μεγάλη κλίμακα με τις ήδη υπάρχουσες τεχνικές. Κύριο χαρακτηριστικό είναι η χρήση πόρων και υπηρεσιών ανάλογα με τη ζήτηση, έτσι ώστε κάθε στιγμή να μη χρησιμοποιούνται παραπάνω πόροι από αυτούς που πραγματικά είναι απαραίτητοι.

Η Υπολογιστική Νέφους βασίζεται στην τεχνική της εικονικοποίησης (Virtualization). Δηλαδή, στην απόκρυψη όλης της δομής και του υλικού ενός συστήματος υπολογιστικού νέφους και εμφάνιση στο χρήστη μόνο εικονικών πόρων που μπορεί να χρησιμοποιήσει κατά βούληση. Ανάλογα με τη χρήση για την οποία προορίζεται το κάθε σύστημα, δεσμεύονται οι κατάλληλοι πόροι οι οποίοι γίνονται διαθέσιμοι στον χρήστη ή την εφαρμογή που τις αιτείται. Η αποδοτική κατανομή και διαχείριση αυτών των εικονικών πόρων είναι αντικείμενο της *εξισορρόπησης φορτίου* (Load Balancing).

Η εξισορρόπηση φορτίου, είναι ένας από τους βασικότερους τομείς του υπολογιστικού νέφους. Στόχος του τομέα αυτού είναι να δημιουργεί και να απελευθερώνει εικονικούς πόρους καθώς και να μοιράζει αποδοτικά σε αυτούς τις εργασίες προς εκτέλεση. Ένας καλός μηχανισμός για εξισορρόπηση φορτίου είναι απαραίτητος για να μπορεί να λειτουργεί αποδοτικά, απρόσκοπτα και με όσο δυνατόν χαμηλότερο κόστος μια υποδομή υπολογιστικού νέφους.



Εικόνα 1.1 Διαφορετικά επίπεδα Load Balancing (*Πηγή:* msdn.microsoft.com)

Η εξισορρόπηση φορτίου είναι μια σημαντική διαδικασία η οποία γίνεται σε αρκετά σημεία μιας υποδομής υπολογιστικού νέφους, όπως φαίνεται και στην παραπάνω εικόνα. Ένας load balancer μπορεί να χρειάζεται να μοιράσει αποδοτικά το φορτίο ανάμεσα σε datacenters, ανάμεσα σε φυσικά μηχανήματα, ανάμεσα σε εικονικούς πόρους, ακόμα και ανάμεσα σε πυρήνες του ίδιου CPU. Για κάθε μία από τις παραπάνω περιπτώσεις απαιτούνται διαφορετικές τεχνικές και πρέπει να αντιμετωπιστούν διαφορετικές προκλήσεις ανάλογα με τις ανάγκες και τις ιδιαιτερότητες που υπάρχουν σε κάθε μία. Κοινός παρονομαστής σε κάθε περίπτωση είναι το ότι προσπαθούμε να εκμεταλλευτούμε όσο καλύτερα γίνεται τους πόρους που έχουμε στη διάθεσή μας για να επιτευχθεί το μέγιστο δυνατό όφελος. Φαίνεται λοιπόν καθαρά το πόσο σημαντικό είναι το πεδίο της εξισορρόπησης φορτίου σε μια

υποδομή υπολογιστικού νέφους ώστε όλα τα κομμάτια της να λειτουργούν σωστά και αποδοτικά.

1.2 Σκοπός και Αντικείμενο Μελέτης

Σκοπός της μεταπτυχιακής διατριβής είναι η κατανόηση των υπαρχόντων μεθόδων και η βελτίωση μιας ή περισσότερων από αυτές σε ταχύτητα, απόδοση ή αξιοπιστία. Ιδανικά θα προταθεί μια εξ ολοκλήρου νέα μέθοδος εξισορρόπησης φορτίου η οποία θα μπορούσε να προτιμηθεί στην πράξη έναντι κάποιας από τις ήδη υπάρχουσες.

Αντικείμενο μελέτης είναι η καταγραφή των επιδόσεων των υπαρχόντων μεθόδων για εξισορρόπηση φορτίου. Στη συνέχεια θα αναπτυχθεί το πειραματικό περιβάλλον στο οποίο θα δοκιμαστούν εξαντλητικά, θα γίνουν οι κατάλληλες μετρήσεις και θα μπορεί κάποιος εύκολα αλλάζοντας κάποιες παραμέτρους να εκτελέσει εκ νέου το πείραμα χωρίς πολλές αλλαγές στην παραμετροποίηση.

Αρχικά θα γίνει η βιβλιογραφική επισκόπηση επάνω στην εξισορρόπηση φορτίου. Θα γίνει σαφές το πρόβλημα και οι παράμετροί του και θα γίνει μια λεπτομερής καταγραφή των πλεονεκτημάτων και των μειονεκτημάτων των σημαντικότερων μεθόδων που χρησιμοποιούνται σήμερα. Στη συνέχεια θα αποφασιστούν οι λεπτομέρειες του πειράματος και θα οριστικοποιηθεί η διαδικασία και τα εργαλεία που θα χρησιμοποιηθούν.

Σκοπός:

- Δημιουργία και παραμετροποίηση της υποδομής για πείραμα που θα συγκρίνει μεθόδους εξισορόπησης φορτίου, η οποία θα παρέχει εύκολη και άμεση προσαρμογή σε νέα δεδομένα και μεθόδους.
- Καταγραφή αποτελεσμάτων της σύγκρισης των σημαντικότερων μεθόδων για εξισορρόπηση φορτίου που χρησιμοποιούνται σήμερα μέσω του πειράματος που θα διενεργηθεί για αυτό το σκοπό.
- Παρατήρηση και ερμηνεία των αποτελεσμάτων. Μέσα από αυτή τη διαδικασία θα προταθεί βελτίωση μίας ή περισσότερων μεθόδων και θα γίνει επανάληψη του πειράματος για να καταγραφούν τα νέα αποτελέσματα.

- Ανάλογα με τις παρατηρήσεις θα προταθεί για εξισορρόπηση φορτίου σε υπολογιστικά νέφη η οποία θα αξιολογηθεί περνώντας τα τεστ με τα οποία δοκιμάστηκαν οι προηγούμενες και με τις ίδιες συνθήκες ώστε να είναι αντικειμενική η σύγκρισή της με τις προηγούμενες.

Αντικείμενο:

- Κατανόηση και ανάλυση υπαρχόντων μεθόδων εξισορρόπησης φορτίου σε συστήματα υπολογιστικού νέφους.
- Έρευνα και επιλογή υποδομής υπολογιστικού νέφους ανοιχτής αρχιτεκτονικής που θα φιλοξενήσει το πείραμα.
- Μελέτη υπαρχόντων λύσεων για γεννήτριες φορτίου και προσαρμογή στις απαιτήσεις του δικού μας πειράματος αν αυτό είναι απαραίτητο.
- Καταγραφή πειραματικών διατάξεων για αντίστοιχες δοκιμές και ανάλυση πλεονεκτημάτων και μειονεκτημάτων τους.
- Εντοπισμός σημείων βελτίωσης των λύσεων για το πρόβλημα του load balancing και πειραματική δοκιμή ώστε να εξακριβωθεί αν η βελτίωση ισχύει και σε πραγματικές συνθήκες.

Προσπάθεια για δημιουργία νέας μεθόδου με τη βοήθεια των παρατηρήσεων και των δεδομένων που έχουν συλλεχθεί.

1.3 Δομή

Η δομή της παρούσας εργασίας είναι η εξής:

- **Κεφάλαιο 1:** Εισαγωγή που περιλαμβάνει σύντομες αναφορές στο θεωρητικό υπόβαθρο καθώς και τους στόχους της παρούσας διπλωματικής εργασίας.
- **Κεφάλαιο 2:** Αναλυτική περιγραφή για την υπολογιστική νέφους γενικά και ειδικότερα την εξισορρόπηση φορτίου και τους τρόπους δοκιμής cloud υποδομών

- **Κεφάλαιο 3:** Περιγραφή του πειραματικού μέρους της εργασίας. Αναφορά στα εργαλεία που χρησιμοποιήθηκαν, τους λόγους που αυτά επιλέχθηκαν και αναλυτική παρουσίαση της πειραματικής διάταξης. Εκτεταμένη αναφορά στην υλοποίηση των ήδη υπαρχόντων αλγορίθμων, καθώς και στις διαφοροποιημένες μεθόδους που υλοποιήθηκαν και μετρήθηκαν.

- **Κεφάλαιο 4:** Εδώ παρουσιάζονται τα αποτελέσματα των μετρήσεων που έγιναν για τους υπάρχοντες αλγορίθμους καθώς και για τις παραλλαγές που υλοποιήθηκαν και μετρήθηκαν. Τέλος υπάρχει και μια σύγκριση της υλοποίησης που φτιάχτηκε για τα πειράματα με τον load balancer του nginx

- **Κεφάλαιο 5:** Μια συνολική αποτίμηση των εργασιών που έγιναν και όλων των συμπερασμάτων που εξάγονται από την παρούσα εργασία, καθώς και μια αναφορά για τις μελλοντικές εργασίες για τις οποίες η παρούσα διπλωματική μπορεί να αποτελέσει τη βάση.

2. ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ

2.1 Cloud Computing

Τα υπολογιστικά νέφη έχουν ως κεντρική ιδέα την εκτέλεση εφαρμογών και γενικότερα τη χρησιμοποίηση υπολογιστικών πόρων σε απομακρυσμένους υπολογιστές ώστε να πραγματοποιηθεί μια εργασία η οποία μέχρι πρότινος θα γινόταν από ένα εγκατεστημένο πρόγραμμα στον προσωπικό υπολογιστή του χρήστη. Η λέξη cloud χρησιμοποιήθηκε λόγω της συχνής σχηματικής απεικόνισης του Διαδικτύου ως ένα σύννεφο μέσα στο οποίο υπάρχουν οι εξυπηρετητές των εφαρμογών και οι διάφοροι υπολογιστές συνδέονται σε αυτό [1].

Κίνητρα για τη δημιουργία και την ανάπτυξη αυτού του τομέα της πληροφορικής είναι, μεταξύ άλλων, η μείωση του κόστους χρήσης για το χρήστη, η ανάγκη του να είναι προσβάσιμα τα αρχεία του από πολλά τερματικά, καθώς και η δημιουργία εφαρμογών που χρησιμοποιούνται από πολλά εκατομμύρια χρήστες σε ολόκληρο τον κόσμο, οι οποίες δε θα μπορούσαν να είναι λειτουργικές σε τόσο μεγάλη κλίμακα με τις ήδη υπάρχουσες τεχνικές[2]. Για παράδειγμα, όσον αφορά το κόστος, η χρησιμοποίηση δημοφιλών προγραμμάτων στο cloud (όπως το photoshop[1] της Adobe η οποία το προσφέρει και ως εφαρμογή στο cloud) ελαφρύνει το χρήστη από τα έξοδα απόκτησής τους αλλά και από τα έξοδα απόκτησης ενός ισχυρού μηχανήματος, εάν η εφαρμογή είναι απαιτητική, αφού όλη η επεξεργαστική ισχύ που απαιτείται αντλείται από τους εξυπηρετητές του νέφους και στον υπολογιστή του χρήστη φτάνει μόνο το επεξεργασμένο αποτέλεσμα.

Στον τομέα της μείωσης του κόστους υπάρχει ένας ακόμα λόγος για τον οποίο η στροφή στο υπολογιστικό νέφος είναι κάτι παραπάνω από οικονομική. Η αγορά, χρήση και συντήρηση εξυπηρετητών από παντός είδους εταιρείες είναι ένα τεράστιο βάρος στον προϋπολογισμό αφού απαιτούν ειδικό χώρο, τακτικές εργασίες συντήρησης και αναβάθμισης και φυσικά προσωπικό το οποίο είναι υπεύθυνο για την εύρυθμη λειτουργία τους. Νοικιάζοντας ένα

[1] http://www.photoshop.com/tools

εικονικό μηχάνημα να παίζει το ρόλο του εξυπηρετητή πολλά από αυτά τα κόστη εκμηδενίζονται και άλλα μειώνονται σε πολύ μεγάλο βαθμό.

Επίσης, πολύ διαδεδομένες είναι και οι υπηρεσίες αποθήκευσης αρχείων σε εξυπηρετητές στο cloud ώστε να είναι προσβάσιμα από οποιοδήποτε μέρος του κόσμου και σε οποιαδήποτε συσκευή θελήσει ο χρήστης [3]. Η χρήση υπηρεσιών αποθήκευσης αρχείων στο Διαδίκτυο απαλλάσσει από την ανάγκη κατοχής πολυάριθμων φορητών, και μη, μέσων αποθήκευσης, διευκολύνει την ανταλλαγή και την κοινή χρήση αρχείων και μπορεί να χρησιμοποιηθεί και ως backup σημαντικών αρχείων σε περίπτωση απώλειας ή βλάβης της φυσικής συσκευής στην οποία βρίσκονται.

Ένας ακόμα τομέας χάρη στον οποίο αναπτύχθηκε ο κλάδος της υπολογιστικής νέφους είναι η εξάπλωση του Διαδικτύου σε συνδυασμό με την ύπαρξη ιστοσελίδων και υπηρεσιών που χρησιμοποιούνται καθημερινά από πάρα πολλά εκατομμύρια χρήστες σε όλον τον πλανήτη, όπως το facebook, το youtube κλπ. Τέτοιες υπηρεσίες έχουν τη δομή cloud εφαρμογής, δηλαδή τρέχουν σε ένα μεγάλο σύνολο εξυπηρετητών. Ανάλογα με τη γεωγραφική περιοχή του χρήστη που κάνει το αίτημα, η σύνδεση γίνεται στον κατάλληλο εξυπηρετητή ο οποίος πρέπει να κάνει τις ζητούμενες ενέργειες γρήγορα, αξιόπιστα, καθώς και (ανάλογα την εφαρμογή) να κάνει διαθέσιμο σε όλη την υποδομή της υπηρεσίας το όποιο αποτέλεσμα της αλληλεπίδρασης του χρήστη με αυτή ή με άλλους χρήστες [4].

2.1.1 Κατηγοριοποίηση του cloud [5]

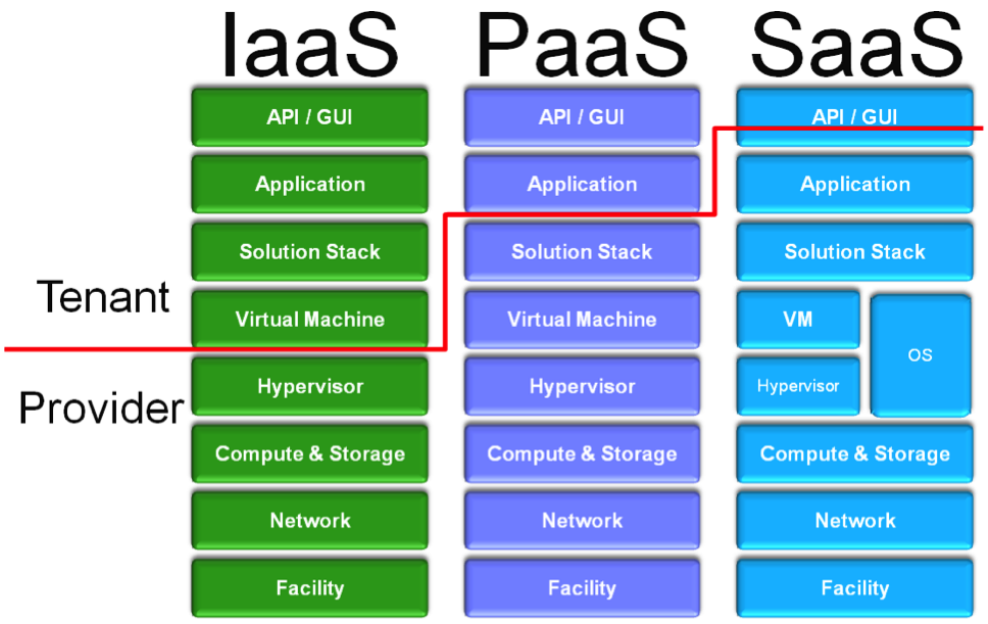
Το cloud, όπως αναφέρθηκε και παραπάνω, χρησιμοποιείται με αρκετούς διαφορετικούς σκοπούς, άρα σύμφωνα με το είδος των προσφερόμενων υπηρεσιών υπάρχει η εξής κατηγοριοποίηση:

- **IaaS (Infrastructure as a Service)** ή αλλιώς υποδομή (hardware) ως υπηρεσία. Σε αυτόν τον τύπο παροχής υπηρεσιών, ο πελάτης νοικιάζει φυσικά μηχανήματα από τον πάροχο υπηρεσιών υπολογιστικού νέφους και χρεώνεται με βάση τους υπολογιστικούς πόρους που χρησιμοποιεί. Δεν παρέχεται κάποιο είδος λογισμικού στον πελάτη και αυτός είναι ελεύθερος να χρησιμοποιήσει το μηχάνημα με όποιον τρόπο επιθυμεί, να εγκαταστήσει το λογισμικό που τον ενδιαφέρει και να το παραμετροποιήσει ώστε να ανταποκρίνεται

στις ανάγκες του. Εδώ εμπίπτει το παράδειγμα της χρησιμοποίησης εξυπηρετητών του cloud έναντι της αγοράς φυσικών μηχανημάτων στο χώρο μιας εταιρείας ή οργανισμού.

- **PaaS (Platform as a Service)** ή αλλιώς πλατφόρμα ως υπηρεσία. Αυτός ο τύπος παροχής υπηρεσιών, είναι ένα επίπεδο πιο πάνω από την προηγούμενη κατηγορία, αφού εδώ ο πελάτης νοικιάζει ένα εικονικό μηχάνημα το οποίο συνήθως προσφέρει και εγκατεστημένο κάποιο λογισμικό που βοηθάει τον πελάτη στην ανάπτυξη της εφαρμογής του. Φυσικά και εδώ οι δυνατότητες του εικονικού μηχανήματος καθορίζονται εξ' αρχής από το χρήστη και λόγω της παραμετροποίησης, που αποτελεί θεμελιώδες χαρακτηριστικό των υπηρεσιών υπολογιστικού νέφους, μπορούν να αλλάξουν κατά παραγγελία ανάλογα με τις ανάγκες του. Και εδώ ο πελάτης έχει την ελευθερία να εγκαταστήσει λογισμικό που τον εξυπηρετεί, έχοντας όμως ήδη ένα πλήρως λειτουργικό μηχάνημα.

- **SaaS (Software as a Service)** ή λογισμικό ως υπηρεσία. Μια πολύ σημαντική κατηγορία υπηρεσιών, που κάποιες φορές προσφέρεται και δωρεάν. Όταν μια εταιρεία παρέχει λογισμικό ως υπηρεσία στους χρήστες της, τους δίνει πρόσβαση να τρέξουν μέσω Διαδικτύου συγκεκριμένες εφαρμογές τις οποίες πλέον δε χρειάζεται να εγκαταστήσουν στον υπολογιστή τους. Είναι το ίδιο με την εκτέλεση της αντίστοιχης εφαρμογής στον τοπικό υπολογιστή του χρήστη χωρίς όμως να απαιτείται η εγκατάστασή της και η κατανάλωση υπολογιστικής ισχύος από το τοπικό μηχάνημα. Σε αυτή την κατηγορία υπηρεσιών ανήκει η σουίτα γραφείου που η Google παρέχει στους χρήστες της ώστε να επεξεργάζονται τα αρχεία τους χωρίς να χρειάζεται να τα κατεβάζουν και η προσφορά του Photoshop της Adobe σε online εκδοχή όπως αναφέρθηκε παραπάνω.



Εικόνα 2.1 Κάθε κατηγορία υπηρεσιών αφήνει διαφορετικά περιθώρια ελέγχου της εικονικής υποδομής στο χρήστη. Η πραγματική υποδομή είναι πάντα υπό τη διαχείριση του παρόχου (*Πηγή*: https://pen-testing.sans.org/blog/2012/07/05/pen-testing-in-the-cloud)

2.1.2 Μοντέλα ανάπτυξης νέφους

Το υπολογιστικό σύννεφο μπορεί επίσης να ταξινομηθεί με βάση το υποκείμενο μοντέλο ανάπτυξης των υποδομών, σε Δημόσιο, Ιδιωτικό, Κοινοτικό και Υβριδικό. Τα διαφορετικά μοντέλα ανάπτυξης των υποδομών διακρίνονται από την αρχιτεκτονική τους, την τοποθεσία που έχουν εγκατεστημένο το κέντρο δεδομένων (data center) που υποστηρίζει το υπολογιστικό νέφος και τις ανάγκες του τελικού πελάτη τους.

- **Public Cloud - Δημόσιο Υπολογιστικό Σύννεφο:** Το Δημόσιο υπολογιστικό σύννεφο είναι ιδιοκτησία του παρόχου των υπηρεσιών Υπολογιστικού Νέφους. Μια τέτοια υπηρεσία υποστηρίζει διάφορες εφαρμογές τις οποίες διάφοροι πελάτες χρησιμοποιούν και ως αντάλλαγμα για την χρήση τους καταβάλλουν ενοίκιο ανάλογο προς την χρήση που έκαναν.

- **Private Cloud - Ιδιωτικό υπολογιστικό σύννεφο:** Ιδιωτικό υπολογιστικό σύννεφο ονομάζεται το εκείνο που είναι φτιαγμένο για την αποκλειστική χρήση ενός και μόνο πελάτη. Υπάρχουν διάφορες παραλλαγές ανάλογα με το καθεστώς ιδιοκτησίας του, τα

λειτουργικά χαρακτηριστικά του κλπ. Ιδιαίτερο πάντως χαρακτηριστικό του είναι η αποκλειστική χρήση του από ένα και μόνο πελάτη. Ένα ιδιωτικό υπολογιστικό νέφος επίσης είναι δυνατό να αποτελεί ιδιοκτησία του πελάτη. Η εγκατάσταση, η λειτουργία και η συντήρηση του ωστόσο δεν γίνεται από αυτόν. Οι φυσικές υποδομές (πχ εξυπηρετητές) είναι δυνατόν να βρίσκονται είτε στις εγκαταστάσεις του πελάτη είτε στις εγκαταστάσεις του παρόχου της υπηρεσίας. Πρόσφατα θεσπίστηκε ο όρος του «εικονικού ιδιωτικού νέφους» (virtual private cloud). Σε ένα τέτοιο εικονικό ιδιωτικό νέφος δεσμεύονται φυσικές υποδομές από ένα δημόσιο υπολογιστικό σύννεφο για την αποκλειστική χρήση ενός και μόνο πελάτη. Εξαιτίας της δέσμευσης των πόρων ο πελάτης μπορεί να είναι βέβαιος ότι τα δεδομένα του αποθηκεύονται και η επεξεργασία τους γίνεται μονάχα σε παραχωρημένους σε αυτόν διακομιστές στους οποίους βεβαίως έχει το αποκλειστικό δικαίωμα χρήσης.

- **Community Cloud - Κοινοτικό υπολογιστικό σύννεφο:** Στην περίπτωση που κάποιοι πελάτες με παρόμοιες ανάγκες αποφασίζουν για λόγους οικονομίας κλίμακας ή άλλους να μοιραστούν κοινές υλικοτεχνικές υποδομές, εγκαταστάσεις, παραμετροποιήσεις αλλά και την ευθύνη της διαχείρισης του Cloud, τότε έχουμε το κοινοτικό υπολογιστικό σύννεφο. Η διαχείριση του υπολογιστικού νέφους στην περίπτωση αυτή μπορεί να γίνεται είτε από τα μέλη της κοινότητας είτε από εξουσιοδοτημένο τρίτο μέλος εκτός κοινότητας.

- **Hybrid Cloud - Υβριδικό υπολογιστικό σύννεφο:** Τέλος οποιοσδήποτε συνδυασμός των παραπάνω μοντέλων ανάπτυξης των υπολογιστικών σύννεφων, μπορεί να οριστεί ως υβριδικό μοντέλο.

Public Cloud	**Private Cloud**	**Hybrid Cloud**
Επεκτάσιμο	Επεκτάσιμο	Επεκτάσιμο
Αξιόπιστο	Ασφαλές	Ασφαλές

Προσιτό οικονομικά	Ευέλικτο	Ευέλικτο
Ανεξαρτησία από την τοποθεσία του χρήστη	Μεγαλύτερος Έλεγχος	Έξυπνη τιμολογιακή πολιτική

Πίνακας 2.1 Τύποι Cloud και χαρακτηριστικά με βάση το υποκείμενο μοντέλο ανάπτυξης των υποδομών

2.1.3 **Χαρακτηριστικά του Cloud** Error! Reference source not found.

Το υπολογιστικό νέφος είναι ένας τομέας της πληροφορικής ταχέως αναπτυσσόμενος και μπορεί να καλύψει ανάγκες που έχουν δημιουργηθεί στους χρήστες από την τεράστια εξάπλωση του Διαδικτύου. Προσπαθώντας να εξακριβωθεί ποια είναι ακριβώς τα κύρια χαρακτηριστικά και άρα οι τομείς μελέτης για το cloud computing, προχωράμε στην εξής κατηγοριοποίησή τους:

I. **Scalability of Infrastructure.** Είναι η ικανότητα να προσθαφαιρούνται εξυπηρετητές στην ήδη υπάρχουσα υποδομή του υπολογιστικού νέφους, το οποίο μπορεί να σημαίνει από προσθήκη μηχανημάτων σε ένα τοπικό κέντρο δεδομένων μέχρι και την προσθήκη στο δίκτυο ολόκληρων κέντρων δεδομένων από άλλα μέρη του πλανήτη. Η ένταξη των νέων εξυπηρετητών/μηχανημάτων και η προσαρμογή του συστήματος πρέπει να γίνεται άμεσα ώστε να υπάρχει πάντα η ιδανική ισορροπία μεταξύ του κόστους συντήρησης και της ικανότητας κάλυψης των αναγκών των χρηστών που εξυπηρετεί το υπολογιστικό νέφος.

II. **Flexibility/Elasticity.** Είναι η δυνατότητα που δίνεται στους χρήστες να τροποποιήσουν, σύμφωνα με τις ανάγκες τους, τους υπολογιστικούς πόρους που θα χρησιμοποιήσουν και κατ' επέκταση το πόσο θα χρεωθούν για αυτή τους τη χρήση. Αυτό πρέπει να γίνεται αυτόματα χωρίς να χρειάζεται να υπάρχει προσωπικό το οποίο θα είναι επιφορτισμένο με τη διαδικασία της χειροκίνητης ρύθμισης τυχόν αλλαγών που επιθυμεί ο κάθε πελάτης.

III. **Virtualization.** Εαν ο χρήστης έχει νοικιάσει ένα μηχάνημα με συγκεκριμένες δυνατότητες στο υπολογιστικό νέφος για να εξυπηρετήσει κάποιους σκοπούς, τότε

πρέπει και να “βλέπει” αυτό το μηχάνημα. Το πιθανότερο είναι πως αυτό το μηχάνημα δεν υπάρχει αυτούσιο στην πραγματικότητα αλλά είναι ένα εικονικό μηχάνημα το οποίο τρέχει επάνω σε τελείως διαφορετικό υλικό. Οι πραγματικές υποδομές ελάχιστα ενδιαφέρουν το χρήστη ο οποίος απλά θέλει να έχει στη διάθεσή του το μηχάνημα για το οποίο πληρώνει και να μην ασχολείται με λεπτομέρειες υλοποίησης.

IV. **Broad network access.** Η συνεχής σύνδεση των εξυπηρετητών, που είναι υπεύθυνοι για τις υπηρεσίες υπολογιστικού νέφους, στο Internet πρέπει να είναι κάτι παραπάνω από δεδομένη. Η μη διαθεσιμότητα κάποιων υπηρεσιών για οποιοδήποτε χρονικό διάστημα μπορεί να προξενήσει στους χρήστες από ασήμαντα προβλήματα (προσωρινή μη διαθεσιμότητα εφεδρικών αρχείων) μέχρι μεγάλη οικονομική ζημιά (σε περιπτώσεις οργανισμών ή εταιριών που η προσφορά υπηρεσιών ή προϊόντων τους βασίζεται στη χρήση του υπολογιστικού νέφους).

V. **Reliability.** Συμπληρωματική της αδιάκοπης σύνδεσης στο Διαδίκτυο είναι η αξιοπιστία των μηχανημάτων για τη διαρκή παροχή υπηρεσιών. Για τους λόγους που αναφέρθηκαν προηγουμένως συχνά οι πάροχοι διαθέτουν παραπάνω κέντρα δεδομένων τα οποία λειτουργούν σε περιπτώσεις disaster recovery και για λόγους επιχειρησιακής συνέχειας.

VI. **Location independence.** Πολύ σημαντικό χαρακτηριστικό του υπολογιστικού νέφους είναι η απρόσκοπτη λειτουργία των εφαρμογών από οποιοδήποτε σημείο του κόσμου. Εάν, για παράδειγμα, ένας χρήστης αποθηκεύσει ένα αρχείο στο λογαριασμό του σε μια υπηρεσία αποθήκευσης αρχείων βασισμένης στο υπολογιστικό νέφος, αυτό το αρχείο πρέπει να είναι διαθέσιμο σε οποιοδήποτε σημείο του κόσμου αυτός ο χρήστης αποκτά πρόσβαση (login) στο λογαριασμό του. Ομοίως, οι αλλαγές στο προφίλ μιας ιστοσελίδας κοινωνικής δικτύωσης ενός χρήστη πρέπει να είναι εμφανείς στους φίλους του σε οποιαδήποτε χώρα και αν κατοικούν.

2.2 Load balancing

Πολύ σημαντικός τομέας έρευνας για το cloud computing είναι η εξισορρόπηση φορτίου. Η αποδοτική διαχείριση των φυσικών και εικονικών πόρων και η ομαλή λειτουργία της υποδομής είναι οι κύριες ευθύνες της εξισορρόπησης φορτίου. Τα κέντρα δεδομένων

υπολογιστικού νέφους αποτελούνται από μεγάλους αριθμούς φυσικών μηχανημάτων τα οποία εκτελούν όλες τις εργασίες για την εξυπηρέτηση των χρηστών αλλά και αυτές που σχετίζονται με τη λειτουργία της υποδομής. Αρκετές φορές μια πολύπλοκη εργασία μοιράζεται σε παραπάνω από ένα μηχάνημα ώστε να εκτελεστεί γρηγορότερα και η απόφαση του ποιο ή ποια θα είναι αυτά τα μηχανήματα λαμβάνεται από τον load balancer. Κίνητρο είναι η υψηλή απόδοση, άρα πρέπει να επιλεγούν μηχανήματα τα οποία είναι όσο το δυνατό λιγότερο φορτωμένα [6]. Αυτό ισχύει είτε αναφερόμαστε σε πραγματικά είτε σε εικονικά μηχανήματα.

2.2.1 Η σημασία του Load Balancing στο cloud Error! Reference source not found.

Η εξισορρόπηση φορτίου στο υπολογιστικό νέφος είναι ένας μηχανισμός που μοιράζει τον πολύ μεγάλο φόρτο ισόποσα σε όλους τους τοπικούς πόρους. Χάρη σ αυτό μπορούμε να πετύχουμε μεγαλύτερη ικανοποίηση στον τελικό χρήστη και ικανοποιητικό ποσοστό χρήσης των πόρων πετυχαίνοντας να μην έχουμε πόρους που υπερχρησιμοποιούνται (και επηρεάζεται αρνητικά η απόδοση) αλλά και να μην υπάρχουν πόροι ανεκμετάλλευτοι (το οποίο μεταφράζεται σε σπατάλη χρημάτων). Με τη σωστή εξισορρόπηση φορτίου πετυχαίνουμε το ιδανικό ποσοστό χρήσης των μηχανημάτων ελαχιστοποιώντας την κατανάλωση πόρων. Στο τέλος αυτά μεταφράζονται, μεταξύ άλλων, σε ελαχιστοποίηση των λαθών, υλοποίηση επεκτασιμότητας, αποφυγή των bottlenecks και του υπερβολικού φόρτου για εξαντλητική και λεπτομερή παρακολούθηση των πόρων.

Η υπολογιστική νέφους προσφέρει διαδικτυακές υπηρεσίες οι οποίες εγγυόνται στον πελάτη μια ελάχιστη απόδοση. Μέχρι τώρα, οι διαδικτυακές υπηρεσίες οι οποίες παρέχονται από αρχιτεκτονικές προσανατολισμένες στην παροχή υπηρεσιών χρησιμοποιούνται μόνο περιορισμένα και εντός συμβολαίων σε business to business (B2B) επίπεδο. Αυτό το μοντέλο λοιπόν, μέχρι πρόσφατα, δεν είχε χρησιμοποιηθεί σε ευρεία κλίμακα.

Κάποιοι λόγοι για τους οποίους συνέβη αυτό είναι μεταξύ άλλων η πολύ υψηλή πολυπλοκότητα και οι τεχνικές γνώσεις που απαιτούνται, τα μεγάλα κόστη για δημιουργία και συντήρηση των υποδομών καθώς και η έλλειψη ύπαρξης τυποποιημένων διαδικασιών

και προτύπων για τον ορισμό της συνεργασίας μεταξύ υπηρεσιών, την αναγνώριση και τη δομή των υπηρεσιών. Αυτά τα προβλήματα είναι συνέπεια της δύσκολης διαχείρισης της αρχιτεκτονικής των υπηρεσιών και των πόρων. Το μέγεθος και η πολυπλοκότητα των συστημάτων που χρειάζονται κάνει την κεντρική διαχείριση συγκεκριμένων μηχανημάτων πολύ δύσκολη, κάτι που σημαίνει πως απαιτούνται αποκεντροποιημένες λύσεις.

Η διαχείριση πόρων ανάλογα με τις τοπικές παραμέτρους και συνθήκες είναι ζωτικής σημασίας και για να γίνει αυτό πρέπει να υιοθετηθεί η έννοια Internet of Things (IoT) όσον αφορά τη διαχείριση των πόρων και των υπηρεσιών. Αυτό σημαίνει πως τα μηχανήματα σε ένα τέτοιο περιβάλλον δημιουργούν μεταξύ τους οικοσυστήματα τα οποία παρέχουν πληροφορίες για τη δομή, την επικοινωνία και τις απαραίτητες οδηγίες για τη λειτουργία και παροχή υπηρεσιών σύμφωνα με τις απαιτήσεις και τους στόχους που έχουν τεθεί.

Για να λειτουργήσει αποδοτικά αυτό το μοντέλο διαχείρισης που στηρίζεται στο Internet of Things, η αυτοοργάνωση και η διαχείριση των πόρων πρέπει να είναι λειτουργίες που θα είναι υπαρκτές σε κάθε επίπεδο αυτών των ψηφιακών οικοσυστημάτων. Έτσι τελικά, η συνολική διαχείριση γίνεται από τα συστήματα σε τοπικές δομές και σε μικρότερη κλίμακα, κάτι που οδηγεί στη συνολική διαχείριση των πόρων.

Εδώ έρχεται η εξισορρόπηση φορτίου για να βοηθήσει τα συστήματα υπολογιστικού νέφους να προσαρμοστούν στις αυξήσεις της ζήτησης για αυτού τους είδους τις υπηρεσίες. Χρειάζεται διότι δεν είναι καθόλου πρακτικό και αποδοτικό, όσον αφορά το κόστος, να διατηρούνται πόροι και υπηρεσίες που δε χρησιμοποιούνται από πελάτες κάποια συγκεκριμένη χρονική περίοδο μόνο και μόνο για να εξασφαλίζεται πως κάποια στιγμή που θα χρειαστούν είναι διαθέσιμες [7]. Επιπλέον, είναι αδύνατο να μοντελοποιηθούν και να αναλυθούν όλες οι πιθανές μελλοντικές καταστάσεις των συστημάτων με πλήρη λεπτομέρεια. Γι’ αυτό είναι απαραίτητο να αποφασιστεί ποιοι πόροι χρειάζονται σε κάθε σύστημα τοπικά, εφαρμόζοντας τους κατάλληλους αλγορίθμους στο σύστημα λαμβάνοντας υπόψη την τωρινή κατάσταση.

Ο αποδοτικός διαμοιρασμός του φόρτου και των πόρων δε μπορεί να λειτουργήσει μοιράζοντας στατικά δουλειές προς επεξεργασία σε προκαθορισμένους εξυπηρετητές, αφού

τα υστήματα υπολογιστικού νέφους μπορούν ανά πάσα στιγμή να αλλάξουν μέγεθος είτε ενσωματώνοντας νέους πόρους, είτε το αντίθετο. Οι δουλειές λοιπόν πρέπει να μοιραστούν ανάλογα με τις πληροφορίες και τα χαρακτηριστικά των μηχανημάτων και των πόρων που συλλέγονται συνεχώς από τους αλγορίθμους που υλοποιούν την εξισορρόπηση φορτίου [8]**Error! Reference source not found.**.

2.2.2 Αλγόριθμοι για Load Balancing

Έχουν αναπτυχθεί αρκετοί αλγόριθμοι για εξισορρόπηση φορτίου, από απλοί και προφανείς μέχρι σύνθετοι, πολύπλοκοι και στοχευμένοι σε συγκεκριμένες εργασίες [9]. Ακολουθεί μια σύντομη παρουσίαση των ειδών και των κατηγοριών των αλγορίθμων για εξισορρόπηση φορτίου**. Οι αλγόριθμοι χωρίζονται σε δύο κατηγορίες:**

α) Σε αυτούς που εξαρτώνται από το ποιος εκκίνησε την εργασία προς εκτέλεση και

β) Σε αυτούς που εξαρτώνται από την κατάσταση του συστήματος πριν και μετά την εκτέλεση της εργασίας, οι οποίοι είναι και οι πιο διαδεδομένοι.

Η δεύτερη κατηγορία χωρίζεται στις 3 εξής υποκατηγορίες Error! Reference source not found.Error! Reference source not found.**:**

1. **Static.** Εδώ ο χρόνος εκτέλεσης της εργασίας είναι κατά ένα μεγάλο βαθμό προβλέψιμος με μεγάλη ακρίβεια πριν την εκτέλεση της εργασίας, οπότε η αποφάσεις για το που θα σταλεί η εργασία είναι σχετικά εύκολες.

2. **Semi-static.** Ο χρόνος εκτέλεσης είναι γνωστός είτε στην αρχή είτε σε καλά ορισμένα σημεία κατά τη διάρκεια της εκτέλεσης πράγμα που επιτρέπει να χρησιμοποιηθούν προσεγγίσεις που είναι κατάλληλες για static scheduling.

3. **Dynamic.** Ο χρόνος εκτέλεσης σε αυτήν την περίπτωση γίνεται γνωστός εφ' όσον έχει ήδη γίνει ένα μεγάλο μέρος της επεξεργασίας του αιτήματος. Σε αυτήν την περίπτωση υπάρχουν αλγόριθμοι που προσαρμόζονται συνέχεια στα δεδομένα και οι κόμβοι προσπαθούν να λύσουν το δύσκολο πρόβλημα της εξισορρόπησης φορτίου είτε συνεργατικά είτε ο καθένας ξεχωριστά.

Μερικοί σημαντικοί αλγόριθμοι για load balancing είναι οι ακόλουθοι [1]Error! Reference source not found.**[11][12]:**

- **Round Robin.** Σε αυτή τη διαδικασία έχουμε ένα σύνολο κόμβων και επιλέγουμε έναν τυχαία. Σε αυτόν θα ανατεθεί η πρώτη εργασία που πρέπει να γίνει. Η επόμενη εργασία θα ανατεθεί στον επόμενο κ.ο.κ. Η ανάθεση των εργασιών γίνεται έτσι κυκλικά, ανεξάρτητα από το φόρτο των κόμβων και το χρόνο διεκπεραίωσης της εργασίας. Είναι η πιο απλή μέθοδος και θα δουλέψει σε περιπτώσεις που οι κόμβοι και οι χρόνοι εκτέλεσης των εργασιών είναι όμοιοι μεταξύ τους. Σε αντίθετη περίπτωση είναι βέβαιο ότι κάποιοι κόμβοι του συστήματος θα είναι σημαντικά γεμάτοι ενώ άλλοι θα χρησιμοποιούνται λιγότερο από όσο πρέπει. Μια απλή παραλλαγή του συγκεκριμένου αλγορίθμου είναι η αντιστοίχιση μιας τιμής “βάρους” στον κάθε κόμβο ανάλογα με την υπολογιστική του δύναμη πράγμα το οποίο επιτρέπει την ανάθεση εργασιών σε κάθε κόμβο ανάλογα με την υπολογιστική του ικανότητα [12]. Άλλη προσπάθεια βελτίωσης του round robin είναι η δουλειά να ανατίθεται σε ένα κόμβο και αν αυτός δεν την εκτελέσει μέχρι ένα προκαθορισμένο χρονικό διάστημα τότε η δουλειά αυτή ανατίθεται στον επόμενο κόμβο.

- **Ip-hash / DNS based load balancing.** Βασικό χαρακτηριστικό αυτών των μεθόδων είναι η προσπάθεια να μοιράσουν το φόρτο ανάλογα με την τοποθεσία του χρήστη. Έτσι ένα αίτημα θα πάει στην κοντινότερη γεωγραφικά υποδομή σε ένα χρήστη με συνέπεια να εξοικονομούνται δικτυακοί πόροι και η απάντηση να είναι πιο άμεση. Συνήθως αυτές οι μέθοδοι αφορούν τα πιο υψηλά επίπεδα των cloud υποδομών ώστε να επιλεχθεί το κατάλληλο υπολογιστικό κέντρο και στη συνέχεια χρησιμοποιούνται και άλλες μέθοδοι για την εσωτερική κατανομή του αιτήματος.

- **Equally Spread Current Execution.** Εδώ οι εργασίες που έρχονται προς εκτέλεση από τους clients καταλήγουν σε μια ουρά. Η δομή που διαχειρίζεται το υπολογιστικό νέφος εκτιμά το μέγεθος της κάθε εργασίας και ελέγχει για τη διαθεσιμότητα του κάθε εικονικού μηχανήματος καθώς και τη χωρητικότητά του [14]**Error! Reference source not found.**. Μόλις γίνει ο υπολογισμός, η εργασία στέλνεται προς εκτέλεση στον κόμβο που

αποφασίστηκε. Αντίθετα με τον round robin δεν υπάρχει το overhead της διαχείρισης των time slots περιοδικά. Πλεονεκτήματα αυτού του αλγορίθμου είναι η βελτίωση τόσο σε χρόνο απόκρισης όσο και σε χρόνο επεξεργασίας. Οι εργασίες μοιράζονται εξίσου στις πηγές και όλο το σύστημα έρχεται σε μια ισορροπία χωρίς να υπάρχουν κόμβοι που δουλεύουν λιγότερο από άλλους. Έτσι έχουμε μείωση στο κόστος των εικονικών μηχανημάτων και της μεταφοράς δεδομένων.

- **Load Balancing of VM Resources.** Χρησιμοποιεί δεδομένα από το παρελθόν για να αποφασίσει ποιόν κόμβο θα στείλει το αίτημα. Στον υπολογισμό συμπεριλαμβάνεται φυσικά και η τωρινή κατάσταση του συστήματος. Οι περισσότερες τεχνικές για εξισορρόπηση φορτίου λαμβάνουν υπ' όψη τους μόνο την τωρινή κατάσταση του συστήματος και σπάνια τα δεδομένα από το παρελθόν που οδηγεί σε μη σωστά κατανεμημένο φόρτο. Αυτή η τεχνική είναι βασισμένη σε γενετικό αλγόριθμο. Αναλύοντας τα δεδομένα από το παρελθόν προσπαθεί να προβλέψει τη συμπεριφορά του συστήματος εάν αλλάξει η κατανομή του φόρτου στα εικονικά μηχανήματα και έτσι αποφεύγονται επιλογές που οδηγούν σε λάθος κατανεμημένο φόρτο.

- **Honeybee Foraging.** Βασίζεται στη συμπεριφορά των μελισσών όταν ψάχνουν για τροφή και γι' αυτό έχει αυτό το όνομα. Ένας μικρός αριθμός από μέλισσες ψάχνει μέρος με τροφή (στην περίπτωσή μας κόμβο διαθέσιμο για εκτέλεση εργασίας) και μόλις το βρει επιστρέφει στη φωλιά (load balancer) για να ενημερώσει για την ποσότητα διαθέσιμου φαγητού **Error! Reference source not found.**Ανάλογα με τους διαθέσιμους πόρους στέλνεται ο κατάλληλος φόρτος σε αυτόν τον κόμβο και στη συνέχεια ξαναενημερώνεται ο load balancer για τους διαθέσιμους πόρους του κόμβου.

- **Biased Random Sampling.** Εδώ, όλοι οι κόμβοι της υποδομής υπολογιστικού νέφους θεωρούνται κορυφές ενός γράφου. Ένα σημείο του γράφου θεωρείται σημείο εισόδου και ένα άλλο σημείο εξόδου. Ένα αίτημα που πρέπει να εκτελεστεί εισέρχεται στο γράφο και βάσει των στατιστικών των κόμβων και των συνδέσεων μεταξύ τους, βρίσκεται το συντομότερο μονοπάτι προς την έξοδο. Μπορεί να χρησιμοποιηθεί σε

οποιουδήποτε μεγέθους γράφους οπότε μπορεί να εφαρμοστεί από ένα μικρό σύνολο κόμβων μέχρι ολόκληρα κέντρα δεδομένων υπολογιστικού νέφους.

- **Active Clustering.** Η μέθοδος αυτή έχει ως βασική ιδέα την ομαδοποίηση των κόμβων ανάλογα με τις εργασίες που μπορεί να εκτελέσει γρηγορότερα ο καθένας. Οι συνδέσεις μεταξύ των κόμβων είναι δυναμικές και ανάλογα με το είδος της εργασίας που πρέπει να εκτελεστεί προστίθενται ή αφαιρούνται κόμβοι από κάθε ομάδα.

- **Compare and Balance.** Χαρακτηριστικό αυτής της τεχνικής είναι η συνεχής προσπάθεια για την επίτευξη της τέλειας ισορροπίας μεταξύ των κόμβων του συστήματος. Ανά προκαθορισμένα χρονικά διαστήματα ο κάθε κόμβος του συστήματος διαλέγει τυχαία έναν άλλο κόμβο και συγκρίνει μαζί του το φόρτο του. Εάν ο άλλος κόμβος είναι λιγότερο φορτωμένος, στέλνει κάποιες εργασίες σε εκείνον. Αυτή η διαδικασία εκτελείται από όλους τους κόμβους του συστήματος.

2.3 Cloud testing and Load generators

2.3.1 Η χρησιμότητα των δοκιμών σε περιβάλλοντα υπολογιστικών νεφών (Cloud testing) [16]

Οι δοκιμές (testing) των υποδομών cloud που δημιουργούμε είναι ένας πολύ σημαντικός τομέας. Ειδικά στο υπολογιστικό νέφος το testing πρέπει να είναι εξαντλητικό και να καλύπτει κάθε πιθανό σενάριο. Λόγω της πολυπλοκότητας των υποδομών το testing είναι απαραίτητο αφενός για να διασφαλιστεί ότι όλα δουλεύουν ομαλά αφετέρου για να βεβαιωθούμε ότι η απόδοση του cloud είναι τέτοια ώστε να μπορεί να υποστηρίξει όλα τα σενάρια χρήσης και να καλύψει στο ακέραιο τις δεσμεύσεις που έχει δώσει ο πάροχος στον πελάτη **Error! Reference source not found.** Επιπλέον, το testing βοηθάει να ανακαλύψουμε τι γίνεται σε περιπτώσεις που ξαφνικά το cloud δέχεται πολύ περισσότερα αιτήματα από τα αναμενόμενα, κάτι που παρατηρείται συχνά και για πολλούς λόγους. Έτσι μπορεί να μελετηθεί η συμπεριφορά της υποδομής σε τέτοιες περιπτώσεις και να αποφασιστεί εάν και πόσοι πόροι χρειάζεται να προστεθούν ώστε να μπορούν να διαχειριστούν τέτοια σενάρια.

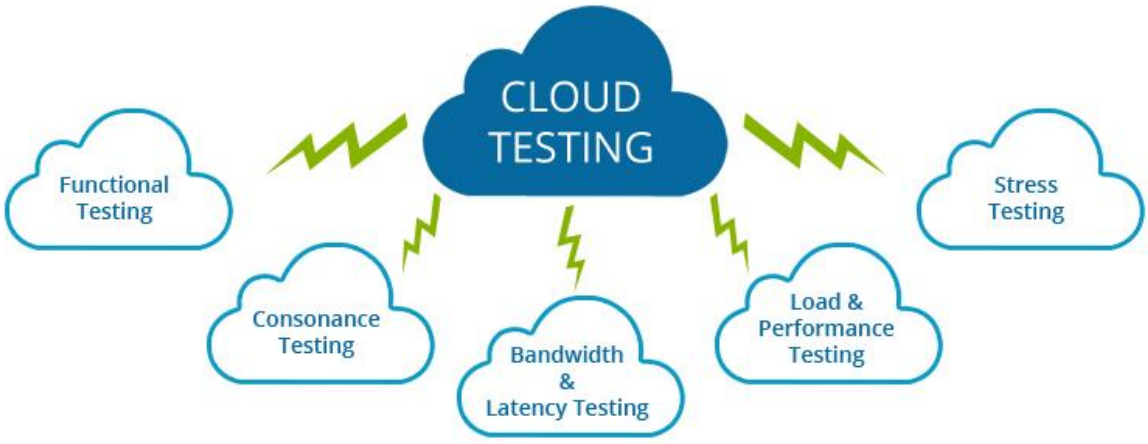
2.3.2 Τρόποι testing Cloud υποδομών

Συνήθως στα συστήματα υπολογιστικού νέφους το testing και οι μετρήσεις γίνονται είτε με τη βοήθεια εργαλείων που προσφέρουν οι πάροχοι υπηρεσιών υπολογιστικού νέφους όταν πρόκειται για απλά στατιστικά είτε στήνοντας testing υποδομές σχεδιασμένες για να ελέγχουν λεπτομερώς την απόδοση ενός ή περισσότερων υποσυστημάτων [11]. Στην πρώτη περίπτωση τα εργαλεία είναι ικανοποιητικά για να έχουμε μια γενική εικόνα του cloud συστήματος σε αριθμούς αλλά δε δίνουν λεπτομερή στατιστικά τα οποία χρειάζονται σε περιπτώσεις όπως αυτή που θα εξετάσουμε στο πλαίσιο της μεταπτυχιακής διατριβής.

Αρκετές φορές χρησιμοποιούνται custom πειραματικά περιβάλλοντα που είναι στοχευμένα σε κάποιο συγκεκριμένο κομμάτι του cloud. Σε τέτοιες περιπτώσεις τα τεστ γίνονται είτε χειροκίνητα είτε αυτοματοποιημένα και υπάρχουν ακόμα και υπηρεσίες testing-as-a-service [[18]] που προσφέρουν μεγάλη παραμετροποίηση τόσο σε επίπεδο του συστήματος που πραγματοποιεί τον έλεγχο όσο και στην υποδομή η οποία ελέγχεται.

Κάποια από τα είδη testing που είναι χρήσιμα στο cloud είναι τα εξής (Εικόνα 2.2):

- **Τεστ λειτουργικότητας (functional testing):** Ελέγχουν πως η υποδομή λειτουργεί και βγάζει σωστά αποτελέσματα.
- **Τεστ συμφωνίας (consonance testing):** Ελέγχεται κατά πόσο οι λειτουργίες και οι διαδικασίες ελέγχου είναι διακριτές και έχουν υλοποιηθεί όπως έχει σχεδιαστεί.
- **Τεστ ελέγχου εύρους ζώνης και καθυστέρησης δικτύου (Bandwidth & Latency testing):** Ελέγχονται οι παράμετροι και η απόδοση του δικτύου ανάμεσα στις μονάδες που αποτελούν την cloud υποδομή
- **Τεστ φόρτου και απόδοσης (Load & Performance testing):** Ελέγχουν αν η απόδοση και ο φόρτος των μηχανημάτων είναι ο αναμενόμενος σε πολλά σενάρια και συνθήκες λειτουργίας
- **Τεστ προσομοίωσης ακραίων καταστάσεων (Stress testing):** Ελέγχουν την απόδοση και συμπεριφορά της cloud υποδομής σε ακραία και μη αναμενόμενα σενάρια χρήσης



Εικόνα 2.2 Πολλά είναι τα test στα οποία μπορούμε να υποβάλουμε τις cloud εφαρμογές (*Πηγή*: https://mycrowd.com/)

Όσο το cloud computing ωριμάζει σιγά σιγά, όλο και μεγαλύτερο μέρος του λογισμικού που προσφέρουν οι εταιρίες προσφέρεται μέσω υπηρεσιών υπολογιατικού νέφους. Φυσικά και η μέτρηση απόδοσης μαζί με τον έλεγχο της συμπεριφοράς μια υποδομής κάτω από συγκεκριμένο φόρτο δε θα μπορούσαν να αποτελούν εξαίρεση. Το να μεταφερθούν αυτές οι υπηρεσίες στο cloud σημαίνει πως πλέον διέπονται από όλα τα πλεονεκτήματά του όπως μικρότερο κόστος και υποστήριξη άμεσης αλλαγής των χρησιμοποιούμενων πόρων **Error! Reference source not found.**Επιπλέον αυτών, υπάρχουν θετικές αλλαγές γενικά στον τρόπο που γίνονται πλέον τα τεστ. Αυτές οι αλλαγές έρχονται ακριβώς πάνω στη στιγμή που όλο και περισσότερες εταιρίες στηρίζονται στο λογισμικό για να αποκτήσουν συγκριτικό πλεονέκτημα στην αγορά.

«*Κάθε εταιρεία είναι εταιρεία λογισμικού ανεξάρτητα από το ποιο είναι το αντικείμενό της. Πολλές από αυτές παράγουν κάθε χρόνο περισσότερες γραμμές κώδικα απ' ό,τι οι μεγαλύτερες εταιρείες λογισμικού. Το λογισμικό κάνει τη διαφορά στον ανταγωνισμό τη σημερινή εποχή*», όπως αναφέρει η Theresa Lanowitz[2].

Μία από τις μεγαλύτερες προκλήσεις στη διαχείριση του κύκλου ζωής του λογισμικού, σύμφωνα με την Lanowitz είναι η απόδοση. «*Η απόδοση θα καθορίσει εάν τελικά κάποιος*

[2] ιδρυτής και αναλυτής του Voke.

θα χρησιμοποιήσει την εφαρμογή σας ή όχι. Εάν σκεφτείτε τον τύπο των εφαρμογών που χρησιμοποιείτε – εταιρικές ή προσωπικές – η απόδοση είναι ο καθοριστικός παράγοντας, γι αυτό να είστε σίγουροι ότι η εφαρμογή σας είναι αποδοτική και μπορείτε να τεστάρετε την απόδοσή της κατάλληλα».

Αυτό ισχύει ακόμα περισσότερο για διαδικτυακές εφαρμογές και εφαρμογές που χρησιμοποιούμε σε κινητές συσκευές. Με τα εργαλεία μέτρησης απόδοσης που είναι βασισμένα στο υπολογιστικό νέφος είναι πιο εύκολο πλέον να διαπιστωθεί εάν η εφαρμογή, είτε μέσα στην εταιρεία είτε για τους πελάτες μπορεί να αντέξει τη ζήτηση των χρηστών.

Ακολουθούν 3 τρόποι με τους οποίους αλλάζει ο τρόπος που τεστάρονται οι διαδικτυακές εφαρμογές λόγω της ύπαρξης υποδομών cloud Error! Reference source not found..

1. **Testing at scale.** Το testing που βασίζεται στο υπολογιστικό νέφος προσφέρει μια συμφέρουσα οικονομικά λύση για τη δοκιμή εφαρμογών σε κλίμακα. Χρησιμοποιώντας μόνο την υποδομή του εργαστηρίου μπορούμε μόνο να εξομοιώσουμε μόνο ένα μέρος της πραγματικής κίνησης. Αυτό σημαίνει πως αντί να τεστάρουμε μια εφαρμογή χρησιμοποιώντας μόνο ένα μέρος των χρηστών και να προσπαθούμε να βγάλουμε συμπέρασμα για τη συμπεριφορά της εφαρμογής σε production περιβάλλον μπορούμε πλέον να τεστάρουμε απ ευθείας με τον πραγματικό αριθμό χρηστών. Αυτό επιτυγχάνεται χάρη στα πολυάριθμα data centers που έχει ο κάθε πάροχος υπηρεσιών υπολογιστικού νέφους σε όλον τον κόσμο. Λόγω αυτής της δομής μπορεί εύκολα να εξομοιωθεί ο αριθμός χρηστών που θα υπάρχει και σε πραγματικές συνθήκες.

2. **Testing globally.** Λόγω της δομής που αναφέραμε πριν το τεστ μπορεί να πραγματοποιηθεί από αρκετές διαφορετικές γεωγραφικές περιοχές, εξομοιώνοντας χρήστες από διαφορετικά μέρη του κόσμου. Αυτό είναι πολύ χρήσιμο σε ένα μεγάλο αριθμό εφαρμογών και υπηρεσιών που η γεωγραφική τοποθεσία του χρήστη παίζει σημαντικό ρόλο για πολλούς λόγους [**Error! Reference source not found.**]. Πάροχοι υπηρεσιών υπολογιστικού νέφους κάνουν ακόμα και συμφωνίες μεταξύ τους ώστε ο ένας να μπορεί να τεστάρει την υποδομή του άλλου έχοντας αμοιβαίο όφελος.

3. **Testing production apps.** Εκτός από να δοκιμάζονται οι εφαρμογές κατά τη διαδικασία ανάπτυξης (development) και τα πρώτα στάδιά τους, το testing που βασίζεται στο υπολογιστικό νέφος μπορεί να χρησιμοποιηθεί και για να ελεγχθεί η απόδοση εφαρμογών που είναι ήδη σε στάδιο παραγωγής (production) και χρησιμοποιούνται ήδη από πολλούς πελάτες. Σε αυτό το σημείο που όλα τα κομμάτια της εφαρμογής έχουν συμπληρωθεί και λειτουργούν στις πραγματικές συνθήκες το testing πρέπει να είναι ακόμα πιο άμεσο και να μπορεί να ανακαλύψει με μεγάλη ταχύτητα τα όποια προβλήματα. Έτσι προλαμβάνονται τα προβλήματα που θα μπορούσαν να έχουν εμφανιστεί στο μέλλον σε πραγματικούς χρήστες πράγμα καθόλου καλό για τη φήμη μιας εφαρμογής ή υπηρεσίας.

2.3.3 Χρήση λογισμικού Testing as a Service

Όταν πρόκειται κάποιος να χρησιμοποιήσει cloud testing εργαλεία, οι επιλογές είναι πλέον πάρα πολλές. Σημαντικό στοιχείο που πρέπει να προσέξει ο κάθε ενδιαφερόμενος είναι το μοντέλο με το οποίο χρεώνει ο κάθε πάροχος τέτοια εργαλεία το οποίο πρέπει να ταιριάζει στις ανάγκες του και τον προϋπολογισμό του. Όσο ανεβαίνει ο αριθμός των χρηστών που θέλουμε να εξομοιώσουμε, συνήθως τόσο ανεβαίνει και το κόστος οπότε χρειάζεται προσοχή**Error! Reference source not found.Error! Reference source not found.**.

Επιπλέον, σημαντικό είναι να εξεταστεί τι στοιχεία είναι διαθέσιμα από τα τεστ ώστε να ερμηνευθούν σωστά και να βγουν ασφαλή συμπεράσματα για τη συμπεριφορά της εφαρμογής.

Πρέπει ο κάθε οργανισμός που θέλει να κάνει χρήση τέτοιων υπηρεσιών να έχει κατανοήσει πλήρως τις δυνατότητες της υπηρεσίας συγκριτικά με τις δικές του απαιτήσεις. Αρκετές φορές το πώς είναι κατασκευασμένη η υποδομή μπορεί να παίζει σημαντικό ρόλο στον τρόπο ερμηνείας των αποτελεσμάτων και επομένως είναι πολύ σημαντικό ο κάθε προγραμματιστής που θα χρησιμοποιήσει τέτοια εργαλεία να γνωρίζει καλά τις λεπτομέρειες υλοποίησης [9].

Σχεδόν σίγουρο είναι ότι θα γνωρίζουμε τα τεχνικά χαρακτηριστικά (πλήθος CPUs, μνήμη, κλπ) των εικονικών μηχανημάτων που διαχειριζόμαστε. Για παράδειγμα, στις υπηρεσίες που

προσφέρει η εταιρεία Amazon (AWS[3]) μπορεί κάποιος να διαλέξει ανάμεσα σε αρκετούς τύπους και μεγέθη μηχανημάτων. Κάποιες λεπτομέρειες σχετικές με την υποδομή και τα μηχανήματα πάνω στα οποία τρέχει ο κάθε τύπος εικονικού μηχανήματος παρέχονται από την Amazon ώστε ο διαχειριστής να έχει αυτή τη γνώση. Όπως και σε ένα περιβάλλον δοκιμών εκτός υπολογιστικού νέφους πρέπει να είναι γνωστά αυτά τα χαρακτηριστικά τα οποία θα χρησιμοποιούνται σε περιβάλλον παραγωγικής λειτουργίας (production mode). Συμπερασματικά, το testing πρέπει να είναι προσαρμοσμένο στον τύπο μηχανήματος που έχουμε επιλέξει και τα τεχνικά χαρακτηριστικά της υποδομής πάνω στην οποία η εφαρμογή θα τρέχει όταν είναι σε production mode [17]. Διαφορετική προσέγγιση θα οδηγήσει σε λάθος συμπεράσματα όπως ακριβώς, θα γινόταν και σε παραδοσιακές αρχιτεκτονικές bare metal.

Λογισμικό virtualization

Ένα ακόμα σημαντικό κομμάτι της αρχιτεκτονικής του cloud είναι το λογισμικό που χρησιμοποιείται για το virtualization των πόρων το οποίο τρέχει πάνω στα πραγματικά μηχανήματα. Για τον τελικό χρήστη δεν είναι τόσο σημαντικό ποιο ακριβώς είναι αυτό το λογισμικό, όσο να ξέρει ότι ο πάροχος υπολογιστικού νέφους χρησιμοποιεί αυτό και μόνο το λογισμικό σε όλες του τις πραγματικές υποδομές με τις οποίες μπορεί να αλληλεπιδράσουν κάποια κομμάτια της εικονικής μας υποδομής.

Διαμοιραζόμενοι πόροι

Κάτι επίσης πολύ σημαντικό που πρέπει να αναλογιστεί ο υπεύθυνος της cloud υποδομής είναι πως το υλικό το οποίο χρησιμοποιούν οι δικοί του πόροι το μοιράζεται επίσης με άλλους πελάτες του παρόχου. Αυτός πρακτικά είναι και ο σκοπός του virtualization, να μπορέσει δηλαδή ο πάροχος να «νοικιάζει» το ίδιο υλικό σε πολλούς πελάτες. Είναι προφανές πως αυτή η παράμετρος είναι πάρα πολύ σημαντική.

Βέβαια, η χρήση των ίδιων πόρων από πολλούς χρήστες σίγουρα επηρεάζει την απόδοση. Υπάρχουν τρόποι να περιορίσει κανείς το πόσο επηρεάζεται από τη χρήση πόρων από

[3] https://aws.amazon.com/

άλλους πελάτες. Αρκετοί πάροχοι δίνουν τη δυνατότητα να αγοράσει κάποιος πακέτα που εγγυόνται κάποιο ελάχιστο ή σταθερό εύρος ζώνης (bandwidth) για παράδειγμα, είτε σε δικτυακούς πόρους είτε σε ταχύτητα δίσκων. Ακόμα και να είναι μοιρασμένοι οι φυσικοί πόροι λοιπόν με άλλους χρήστες είμαστε σίγουροι πως σε κάθε περίπτωση έχουμε μια ελάχιστη εγγυημένη ταχύτητα. Επιπλέον υπάρχουν επιλογές να περιορίσει τον αριθμό των χρηστών με τους οποίους μοιράζεται τους ίδιους πόρους ακόμα και έχοντας την επιλογή να κάνει αποκλειστική χρήση τους. Αυτά τα πακέτα σίγουρα κοστίζουν παραπάνω αλλά σε κάποιες εφαρμογές (π.χ. εφαρμογές ευαίσθητες σε θέματα ασφάλειας [[21]] ή εφαρμογές με πολύ αυστηρά κριτήρια απόδοσης) μπορεί να είναι μονόδρομος [4].

Longevity tests

Η μεγάλη συλλογή δεδομένων από τα τεστ για την απόδοση της εφαρμογής/υπηρεσίας μπορεί να αποτελέσει ένα ισχυρό μέσο για την πρόβλεψη και ελαχιστοποίηση της επιρροής άλλων χρηστών του cloud προς την υπηρεσία. Το ποσοστό χρήσης των φυσικών μηχανημάτων από άλλους χρήστες αποτελεί έναν αστάθμητο παράγοντα ο οποίος φαινομενικά είναι δύσκολο να ελεγχθεί. Ένα πανίσχυρο εργαλείο που μπορεί να χρησιμοποιηθεί για ποσοτικοποίηση αυτής της επιρροής είναι τα **«τεστ μακροζωίας» (longevity tests)** [[22]]. Αυτά είναι μια σειρά από τεστ που γίνονται με σταθερό φόρτο στο σύστημά μας για παρατεταμένες χρονικές περιόδους, για παράδειγμα 8 ώρες ή και περισσότερο. Έτσι λαμβάνουμε πολλά χρήσιμα δεδομένα για την απόδοση και τη σταθερότητα του συστήματος. Παρ’ όλο που μετράμε την απόδοση της δικής μας εφαρμογής, έμμεσα μετράμε και τη συμπεριφορά άλλων χρηστών που μοιράζονται το ίδιο σύστημα με εμάς. Για παράδειγμα, εάν έχουμε μετρήσει για αρκετή ώρα την εφαρμογή μας και έχουμε μια σταθερή τιμή στις μετρήσεις και ξαφνικά παρατηρήσουμε μια μείωση στην απόδοση για ένα σημαντικό χρονικό διάστημα η οποία τελικά θα επανέλθει στα προηγούμενα επίπεδα, σημαίνει πως εκείνη τη χρονική περίοδο κάποιος άλλος συνδρομητής έκανε χρήση των κοινών πόρων. Μπορούμε να ποσοτικοποιήσουμε έτσι σε μεγάλο βαθμό την επιρροή των άλλων χρηστών στην απόδοση της εφαρμογής μας. Εάν οι μετρήσεις δείχνουν πως η απόδοση της εφαρμογής, παρ’ όλο που επηρεάζεται εμφανώς, δεν πέφτει (κάτω από τα όρια που έχουμε θέσει) κατά το longevity test, τότε μπορούμε να

συμπεράνουμε με σχετική ασφάλεια ότι η απόδοση του συστήματός μας είναι εξασφαλισμένη και δεν θα προκύψουν δυσάρεστες εκπλήξεις **Error! Reference source not found.**.

Δεν μπορούμε να είμαστε απόλυτοι ότι κάποιες περίοδοι μειωμένης απόδοσης της εφαρμογής -ενόσω αυτή διαχειρίζεται φόρτο- οφείλονται σε ταυτόχρονη αυξημένη χρήση απ' την πλευρά άλλων χρηστών. Για τον παραπάνω λόγο, τα longevity tests πρέπει να τρέξουν συνολικά για δεκάδες ώρες έτσι ώστε να έχουμε ένα μεγάλο αριθμό δεδομένων για τη συμπεριφορά της απόδοσης και το πώς αυτή επηρεάζεται από πολλούς παράγοντες. Με όλα αυτά τα δεδομένα πρέπει να είμαστε ικανοί να φροντίσουμε για την ελάχιστη δυνατή επιρροή στην απόδοση της εφαρμογής μας από τη δραστηριότητα άλλων χρηστών στους κοινούς πόρους. Για να γίνει αυτό, απαιτεί πολύ καλή ανάγνωση και ερμηνεία όλων των δεδομένων των μετρήσεων [[22]].

Είδη εικονικών μηχανημάτων

Κάθε πάροχος θα προσφέρει διαφορετικούς τύπους εικονικών μηχανημάτων από τα οποία μπορεί να επιλέξει ο χρήστης. Κάποιοι πάροχοι επιπλέον κατηγοριοποιούν αυτούς τους τύπους σε γκρουπ με διάφορες ιδιότητες, οπότε ο χρήστης έχει ακόμα περισσότερες επιλογές. Αυτή η τακτική βοηθάει στη στοχευμένη επιλογή συγκεκριμένων τύπων για τα εικονικά μηχανήματα που έχουν τέτοιες ιδιότητες ώστε να μεγιστοποιούν το λόγο απόδοσης προς κόστος [7]. Αυτό δίνει στον πελάτη το πλεονέκτημα να πληρώσει ακριβώς για όσους πόρους χρειάζεται. Η απόδοση λοιπόν διαφέρει ανάμεσα στα γκρουπ και τους τύπους μηχανημάτων και το ίδιο τεστ θα έχει διαφορετικά αποτελέσματα σε κάθε ένα από αυτά **Error! Reference source not found.**.

Ένα από τα πλεονεκτήματα του υπολογιστικού νέφους είναι η ευκολία αλλαγής των χαρακτηριστικών της υποδομής. Αυτό κάνει εξίσου εύκολη την πραγματοποίηση τεστ σε ένα σύνολο από συνδυασμούς εικονικών μηχανημάτων για να αποφασιστεί ο πιο οικονομικός συνδυασμός εικονικών μηχανημάτων που θα φιλοξενήσει την εφαρμογή. Είναι έτσι προφανές πως τα τεστ δε βοηθάνε μόνο στην απόδοση, αλλά και στο κόστος, δίνοντας περιθώρια σε μια εταιρία για μεγαλύτερο κέρδος και ανταγωνιστικότερες τιμές προς τους

πελάτες, πράγμα που καταδεικνύει την αναγκαιότητα για λεπτομερή μελέτη και σχεδιασμό του testing στην εφαρμογή μας [4]

Παραπάνω μελετήθηκαν μόνο κάποια σημαντικά και κύρια στοιχεία που αφορούν το testing σε cloud υποδομές. Εξαρτάται από τον πάροχο και την υποδομή του τι άλλες παράμετροι θα πρέπει να ληφθούν υπ' όψη για το σχεδιασμό και την υλοποίηση των τεστ.

2.3.4 Συμπεράσματα

Το testing είναι ένα πάρα πολύ σημαντικό στάδιο ειδικά όσον αφορά εφαρμογές σε περιβάλλον υπολογιστικού νέφους. Η διαφορά του υπολογιστικού νέφους είναι ότι πλέον υπάρχει και το στρώμα του virtualization.

Σχετικά με το λογισμικό virtualization πρέπει να σημειωθεί ότι προσθέτει ένα overhead, ώστε να μπορέσει να αντιστοιχίσει τους εικονικούς πόρους με τους πραγματικούς, πράγμα το οποίο δε γίνεται σε αρχιτεκτονικές εκτός cloud συστημάτων. Επομένως, το ίδιο φυσικό μηχάνημα θα χάσει ένα ποσοστό της ωφέλιμης υπολογιστικής του ισχύος όταν χρησιμοποιηθεί σε cloud υποδομή. Έχει μετρηθεί από πειράματα και πραγματικές συνθήκες πως το ποσοστό των πόρων που καταναλώνονται για αυτόν το σκοπό είναι περίπου 10%-15% [[22]].

Επίσης, είναι σημαντικό ο πάροχος να χρησιμοποιεί ένα συγκεκριμένο λογισμικό για virtualization [**Error! Reference source not found.**]. Σε αντίθετη περίπτωση κάποιες παράμετροι των αποτελεσμάτων των μετρήσεων θα μπορούσαν να ερμηνευτούν τελείως λανθασμένα λόγω διαφορετικών παραμέτρων και προσέγγισης του κάθε λογισμικού virtualization [[22]].

Ένα ακόμα συμπέρασμα που προκύπτει είναι ότι θα ήταν πάρα πολύ χρήσιμο να έχουμε την πληροφορία με πόσους και πώς μοιραζόμαστε τους ίδιους πόρους. Αυτό εξαρτάται πολύ από τον πάροχο και τη δομή των data center του, οπότε αυτό θέλει αρκετή διερεύνηση. Για παράδειγμα στο AWS της Amazon οι πόροι που μοιράζονται είναι μεταξύ άλλων το δίκτυο και οι δίσκοι αποθήκευσης. Αυτό σημαίνει πως πρέπει να δοθεί ιδιαίτερη προσοχή στα αποτελέσματα των μετρήσεων απόδοσης αυτών των υποσυστημάτων γιατί αυτά μπορεί να έχουν επηρεαστεί εκείνη την ώρα από την ταυτόχρονη χρήση τους με άλλους που μπορεί να

κάνουν αυξημένη χρήση τους εκείνη τη στιγμή [17]**Error! Reference source not found.**. Ακόμα και αυτά τα αποτελέσματα είναι πολύ σημαντικά γιατί μπορεί να μετρηθεί ποσοτικά σε ποιο βαθμό μπορεί να επηρεαστεί η απόδοση της εφαρμογή μας συγκριτικά με το φόρτο των πραγματικών μηχανημάτων **Error! Reference source not found.**.

Το σημαντικότερο όλων είναι να γνωρίζουμε ότι το cloud είναι μια μοιραζόμενη πλατφόρμα οπότε η απόδοση της εφαρμογής μας θα επηρεαστεί από τη χρήση των υπόλοιπων συνδρομητών που μοιράζονται το ίδιο υλικό (hardware) με εμάς. Η ζήτηση είναι κατά κύριο λόγο μη προβλέψιμη, οπότε αποτελεί ένα υπαρκτό ρίσκο που πρέπει οπωσδήποτε να συνυπολογιστεί.

3. ΕΡΓΑΛΕΙΑ ΚΑΙ ΜΕΘΟΔΟΙ

3.1 Υποδομή

Για τα πειράματα είναι απαραίτητο ένα περιβάλλον υπολογιστικού νέφους στο οποίο θα γίνει η υλοποίηση και όλες οι μετρήσεις. Εξετάστηκαν περιπτώσεις όπως η χρήση του AWS της Amazon ή της υπηρεσίας Azure της Microsoft που προσφέρουν κάποια δωρεάν χρήση πόρων αλλά αυτοί οι πόροι δεν ήταν αρκετοί για να φτιαχτεί ένα πείραμα το οποίο θα κάνει μετρήσεις που θα προσομοιώνουν ικανοποιητικά τις πραγματικές συνθήκες. Έτσι χρησιμοποιήθηκε ο Okeanos λόγω του ότι μπορούσαμε να έχουμε πρόσβαση σε λογαριασμούς οι οποίοι μας επιτρέπουν να έχουμε ένα ικανοποιητικό αριθμό μηχανημάτων.

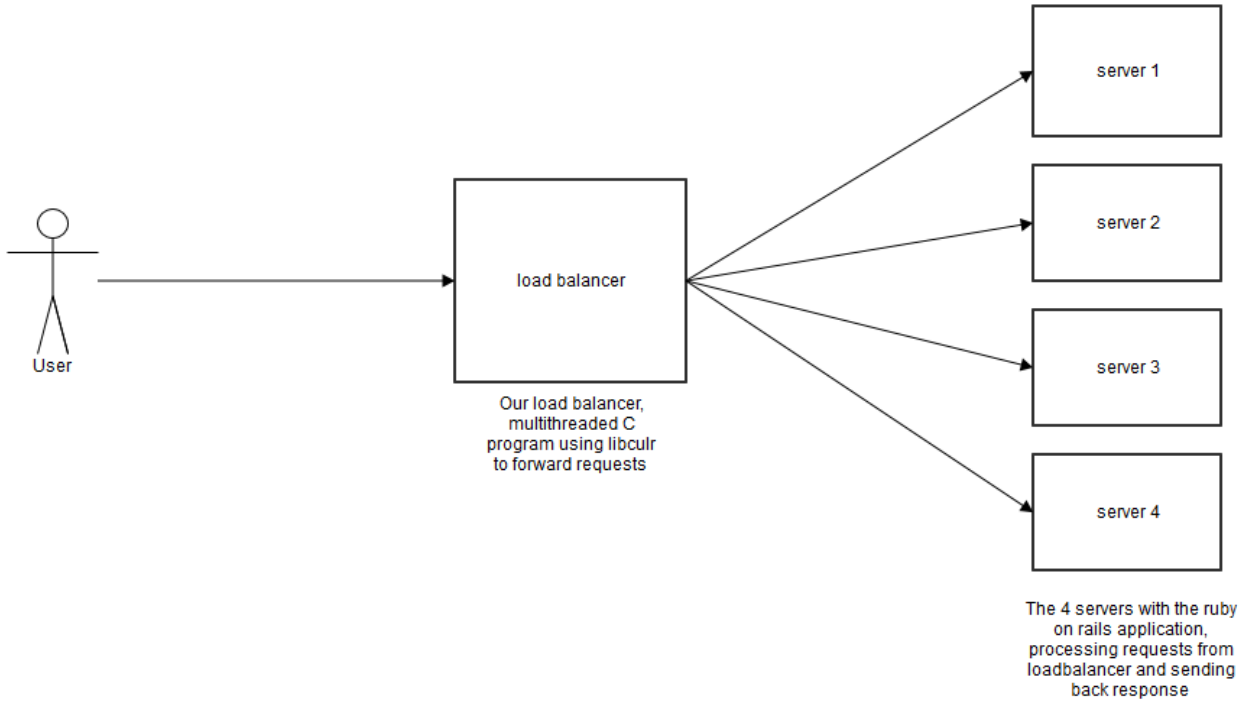
Συγκεκριμένα η υποδομή μας αποτελείται από 5 εικονικά μηχανήματα το καθένα από τα οποία έχει 2 εικονικούς επεξεργαστές και 4GB μνήμης. Το πείραμα περιγράφεται στην επόμενη παράγραφο.

Σενάριο

Αυτό που θα ελεγχθεί και θα μετρηθεί ως προς την απόδοση είναι μια απλή web εφαρμογή. Η εφαρμογή αυτή φτιάχτηκε σε ruby on rails[4]. Αυτό που κάνει είναι να διαβάζει κάποιες εγγραφές από τη βάση δεδομένων, να τις ταξινομεί και να επιστρέφει το αποτέλεσμα σε μια web σελίδα. Αυτό που θα μετρηθεί είναι η ταχύτητα με την οποία θα παίρνει ο χρήστης την απάντηση. Η εφαρμογή αυτή τρέχει με τα ίδια ακριβώς δεδομένα στα 4 εικονικά μηχανήματα. Η δουλειά του 5^{ου} εικονικού μηχανήματος είναι η κρίσιμη και η πιο σημαντική. Ο τελικός χρήστης γνωρίζει την ip μόνο αυτού του εικονικού μηχανήματος και απευθύνει τα request μόνο σε αυτό. Σε αυτό το μηχάνημα δεν τρέχει η εφαρμογή αλλά μόνο ο load balancer που υλοποιεί τους αλγορίθμους που θα παρουσιαστούν στην επόμενη ενότητα. Ανάλογα με τον αλγόριθμο και τα στατιστικά που έχει στη διάθεσή του το request θα προωθηθεί στο πιο κατάλληλο εικονικό μηχάνημα από τα 4. Στη συνέχεια αφού η εφαρμογή τελειώσει με την εξυπηρέτηση αυτού του αιτήματος θα επιστρέψει το αποτέλεσμα πίσω

[4] http://rubyonrails.org/

στον load balancer ο οποίος θα το προωθήσει πίσω στον τελικό χρήστη. Στην επόμενη εικόνα έχουμε τη σχηματική αναπαράσταση της πειραματικής διάταξης.



Εικόνα 3.1 Το βασικό σενάριο χρήσης της υποδομής

3.1.1 Εφαρμογή που χρησιμοποιήθηκε ως benchmark

Ο καθένας από τους 4 εξυπηρετητές τρέχει τη ruby on rails εφαρμογή που χρησιμοποιήθηκε ως benchmark. Η εφαρμογή είναι ακριβώς η ίδια και στους 4 εξυπηρετητές. Διαθέτει μια βάση δεδομένων με χρήστες. Σε κάθε αίτημα, αυτό που κάνει είναι η ανάγνωση αυτών των εγγραφών από το δίσκο, η ταξινόμησή τους και η παρουσίασή τους σε μια ιστοσελίδα. Η αναπαράσταση των χρηστών και ο αριθμός τους είναι ο ίδιος σε όλους τους server, η εφαρμογή λειτουργεί με τον ίδιο ακριβώς τρόπο και χρησιμοποιώντας το ίδια εργαλεία.

Σε αντίθετη περίπτωση θα δίνονταν είτε πλεονέκτημα είτε μειονέκτημα μόνιμα σε κάποιους εξυπηρετητές. Έτσι, υπό κάποιες συνθήκες θα μπορούσαν να ευνοηθούν κάποιοι αλγόριθμοι έναντι κάποιων άλλων.

Ένα αρνητικό βέβαια που προκύπτει από αυτή τη συμμετρία είναι πως τελικά η χρήση αλγορίθμων εξισορρόπησης φορτίου είναι περιττή αφού έχουμε ίδια μηχανήματα που τρέχουν την ίδια εφαρμογή, οπότε αρκεί τα αιτήματα να μοιράζονται εξ ίσου, δηλαδή φτάνει η χρήση της τεχνικής round robin. Πράγματι όταν μετρήθηκαν οι αλγόριθμοι με αυτές τις συνθήκες ο round robin ήταν ο πιο γρήγορος αφού έχει το μικρότερο overhead από όλους, αφού όλοι οι υπολογισμοί και οι μετρήσεις που κάνουν οι υπόλοιποι αλγόριθμοι καταλήγουν στο να μοιράζουν εξ ίσου τα requests στους 4 εξυπηρετητές. Δηλαδή κάνουν το ίδιο με τον round robin κάνοντας παραπάνω υπολογισμούς.

Για να παρακαμφθεί αυτό το πρόβλημα έπρεπε να δημιουργήσουμε τεχνητό φόρτο στα μηχανήματα για να εξαλειφθεί η συμμετρία, η οποία πολύ σπάνια υπάρχει σε πραγματικές συνθήκες. Έτσι έγινε χρήση του εργαλείου stresslinux[5]. Με αυτό το εργαλείο δημιουργείται περιοδικά και με κάποια πιθανότητα φόρτος στα μηχανήματα, κάτι το οποίο προσομοιώνει πραγματικές περιπτώσεις που ένα εικονικό μηχάνημα είναι πιο απασχολημένο από τα υπόλοιπα. Έτσι έχει διαφορετική απόδοση και κάνει περισσότερη ώρα να επεξεργαστεί το αίτημα του χρήστη. Ο load balancer λοιπόν είναι επιφορτισμένος με το να εντοπίζει αυτές τις καθυστερήσεις και να προωθεί το αίτημα με βάση αυτές και την εκάστοτε τεχνική.

Περισσότερες λεπτομέρειες για τη χρήση του stresslinux και τον τρόπο που χρησιμοποιήθηκε θα παρουσιαστούν στο κεφάλαιο 5.

3.1.2 Ο Load balancer

Όσον αφορά το load balancer αρχικά είχε επιλεγεί η χρήση του nginx[6] και η δοκιμή με τις ήδη υλοποιημένες τεχνικές. Ο nginx προσφέρει τις τεχνικές Round Robin, Least Connections και Ip Hash ως λύσεις για εξισορρόπηση φορτίου. Ενώ αρχικά τα πειράματα έγιναν με τον nginx και έγιναν κάποιες μετρήσεις τελικά εγκαταλείφθηκε ως προσέγγιση. Προτιμήθηκε ένας νέος server που υλοποιήθηκε από την αρχή. Ο λόγος είναι πως ο φόρτος του να μελετηθεί λεπτομερώς ο κώδικας του nginx ώστε να μπορέσουν να γίνουν πιθανές βελτιώσεις πάνω σε αυτόν είναι πολύ μεγάλος και ξεπερνάει τα πλαίσια της παρούσας

[5] https://www.stresslinux.org/sl/

[6] https://nginx.org/

διπλωματικής εργασίας. Έτσι λοιπόν ο load balancer είναι ένας εξυπηρετητής γραμμένος σε C ο οποίος έχει υλοποιημένες τις τεχνικές εξισορρόπησης φορτίου που θα μετρήσουμε. Κάνοντας αυτήν την επιλογή η τροποποίηση και η υλοποίηση νέων μεθόδων είναι πολύ ευκολότερη.

Ο load balancer βρίσκεται σε δικό του εικονικό μηχάνημα στην υποδομή του ωκεανού και επικοινωνεί και με τα 4 εικονικά μηχανήματα που τρέχουν την εφαρμογή και όπως είναι φυσικό γνωρίζει όλες τις ip διευθύνσεις τους. Κάθε φορά που λαμβάνει ένα νέο αίτημα, δημιουργείται ένα νέο thread το οποίο θα το διαχειριστεί. Εκεί επιλέγεται ο κατάλληλος εξυπηρετητής, ενημερώνονται οι δομές και το αίτημα προωθείται στον εξυπηρετητή που αποφασίστηκε. Το thread περιμένει μέχρι να λάβει την απάντηση

Για την υλοποίηση του μηχανισμού προώθησης και διαχείρισης αιτημάτων χρησιμοποιήθηκε η βιβλιοθήκη libcurl[7]. Είναι μια βιβλιοθήκη της γλώσσας C για δημιουργία αιτημάτων προς εξυπηρετητές. Έχει απλό interface και δεν απαιτεί ιδιαίτερη προσπάθεια για εκμάθηση από την πλευρά του προγραμματιστή. Αρχικά πρέπει να αρχικοποιηθεί η βιβλιοθήκη σε global scope χρησιμοποιώντας τη συνάρτηση `curl_global_init`. Στη συνέχεια, πριν από κάθε αίτημα πρέπει να γίνεται ένα ακόμα initialize με τη συνάρτηση `curl_easy_init` και να τεθούν κάποιες μεταβλητές με τη χρήση του `curl_easy_setopt`. Μετά το αίτημα στέλνεται στο χρήστη χρησιμοποιώντας τη συνάρτηση `curl_easy_perform`.

Η βιβλιοθήκη αυτή μπορεί να λειτουργήσει είτε σύγχρονα είτε ασύγχρονα. Στη δική μας περίπτωση είναι χρήσιμη η σύγχρονη λειτουργία της Libcurl αφού το thread περιμένει την απάντηση του server για να τη στείλει πίσω στο χρήστη.

[7] https://curl.haxx.se/libcurl/

Ακολουθεί ένα απλοποιημένο παράδειγμα (Εικόνα 3.2) χρήσης της από τον load balancer που υλοποιήθηκε στο πλαίσιο της εργασίας. Αρκετές λεπτομέρειες υλοποίησης, όπως αρχικοποιήσεις μεταβλητών, κώδικας για χρονομέτρηση του request και κάποια μηνύματα για αποσφαλμάτωση (debugging), έχουν αφαιρεθεί για ευκολία στην ανάγνωση του κειμένου. Ολοκληρωμένος ο κώδικας βρίσκεται στο παραδοτέο CD το οποίο περιλαμβάνεται στη μεταπτυχιακή διατριβή.

```
int serve_request(int client_socket, char* server_ip){
   CURL *curl;
   CURLcode res;
   char response_str[5000];
   curl = curl_easy_init();
   strcpy(selected_ip, choose_and_fetch_ip(&served_by_idx));
   if (curl) {
      // setting the server ip
      curl_easy_setopt(curl, CURLOPT_URL, server_ip);
      // setting the function that copies data
      curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, append_html);
      // data copied in response_str
      curl_easy_setopt(curl, CURLOPT_WRITEDATA,
(void*)response_str);
      // actually performing the request
      res = curl_easy_perform(curl);
      // response_str is written to client_socket that client will
read
      write(client_socket, response_str, strlen(response_str));
      curl_easy_cleanup(curl);
   }
}
```

Εικόνα 3.2 Παράδειγμα χρήσης βιβλιοθήκης libcurl για ένα http request

3.2 Αλγόριθμοι που υλοποιήθηκαν

Στο πλαίσιο της παρούσας εργασίας υλοποιήθηκαν 3 αλγόριθμοι οι οποίοι έχουν ήδη μελετηθεί και προϋπάρχουν και 3 παραλλαγές που έχουν στόχο τη σύγκριση με τις ήδη υπάρχουσες καθώς και τη μελέτη επιμέρους χαρακτηριστικών και λεπτομερειών υλοποίησης. Αρχικά θα εξετάσουμε τους 3 ήδη γνωστούς αλγορίθμους:

3.2.1 Round Robin

Ο πιο απλός αλγόριθμος. Δεν έχει κάποιο ιδιαίτερο μηχανισμό ή πολύπλοκο χαρακτηριστικό, απλώς θυμάται ποιος από τους 4 εξυπηρετητές διαχειρίστηκε το προηγούμενο αίτημα και προωθεί το επόμενο στον επόμενο κατά σειρά εξυπηρετητή. Ένας από τους λόγους για τους οποίους επιλέχθηκε ο round robin ήταν επίσης για να έχουμε ένα μέτρο σύγκρισης της απόδοσης των υπόλοιπων αλγορίθμων με την πιο απλή περίπτωση. Οι χρόνοι και τα αποτελέσματα των μετρήσεων του round robin προσφέρονται και ως ένα benchmark για την απόδοση των υπόλοιπων πιο πολύπλοκων τεχνικών. Επιπλέον λειτουργεί και σαν <<proof of concept>> για την πειραματική διάταξη που έχουμε υλοποιήσει εφόσον τα αποτελέσματα των μετρήσεων για τους ήδη υπάρχοντες αλγορίθμους είναι βελτιωμένα σε σχέση με τον round robin, κάτι που σε μια σωστή πειραματική διάταξη είναι αναμενόμενο.

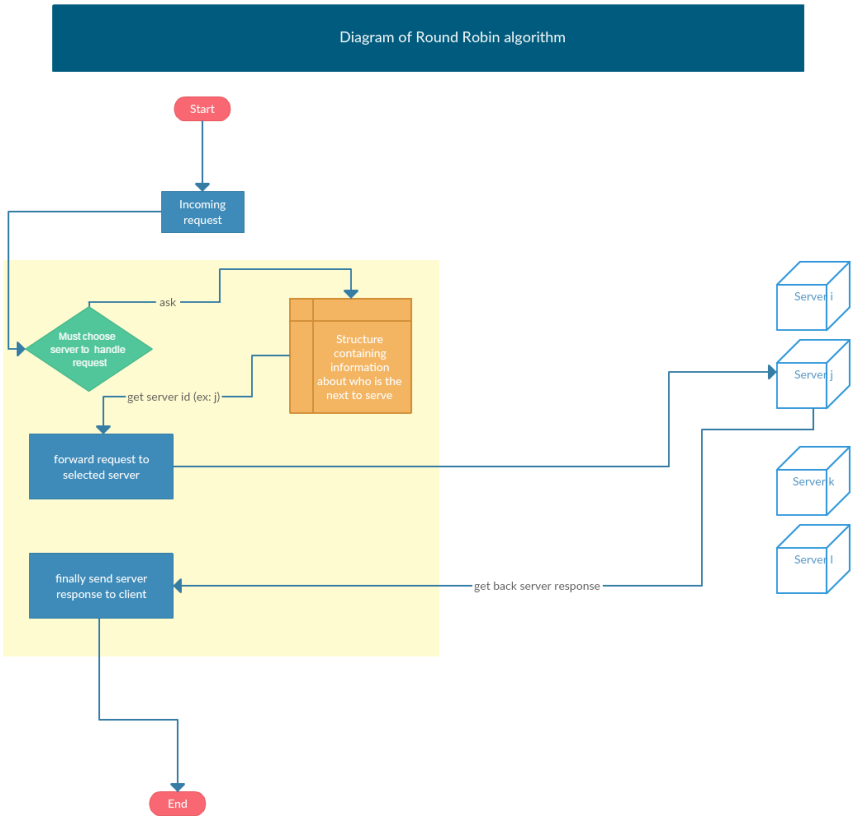
Τη στιγμή που ο load balancer παραλαμβάνει το αίτημα, αμέσως εξετάζεται η δομή που περιέχει τις απαραίτητες πληροφορίες για τη λειτουργία του αλγορίθμου. Ελέγχεται ποιος είναι ο εξυπηρετητής που διαχειρίζεται το τρέχον αίτημα και αμέσως παίρνουμε την την ip του επόμενου στη σειρά εικονικού μηχανήματος στο οποίο θα προωθήσουμε το αίτημα. Το αίτημα προωθείται και πλέον αυτός είναι ο εξυπηρετητής που διαχειρίζεται το τελευταίο αίτημα [[23]]. Το επόμενο αίτημα προωθείται με την ίδια διαδικασία στον επόμενο κατά σειρά και ο κύκλος γυρνάει συνέχεια πηγαίνοντας από εξυπηρετητή σε εξυπηρετητή.

```
servers_container->last_served_index++;
if (servers_container->last_served_index == 4) {
    servers_container->last_served_index=0;
}
return servers_container->servers[servers_container->
last_served_index].ipaddress;
```

Εικόνα 3.3 Επιλογή του επόμενου server στη λίστα και επιστροφή της ip του

Η παραπάνω διαδικασία είναι δυνατό να επαναλαμβάνεται αρκετές φορές πριν προλάβει κάποιος εξυπηρετητής να απαντήσει στο αίτημα που του έγινε στην αρχή. Αυτό σημαίνει πως κάποια χρονική περίοδο ένας server μπορεί να διαχειρίζεται ταυτόχρονα παραπάνω από ένα αίτημα. Αυτός είναι ο στόχος της πειραματικής διάταξης, να προσπαθήσουμε να φορτώσουμε την υποδομή κοντά στα όριά της για να δούμε πως συμπεριφέρονται οι αλγόριθμοι σε απαιτητικές συνθήκες.

Ακολουθεί μια σχηματική αναπαράσταση του αλγορίθμου.



Εικόνα 3.4 Διάγραμμα λειτουργίας round robin

3.2.2 Least Connections

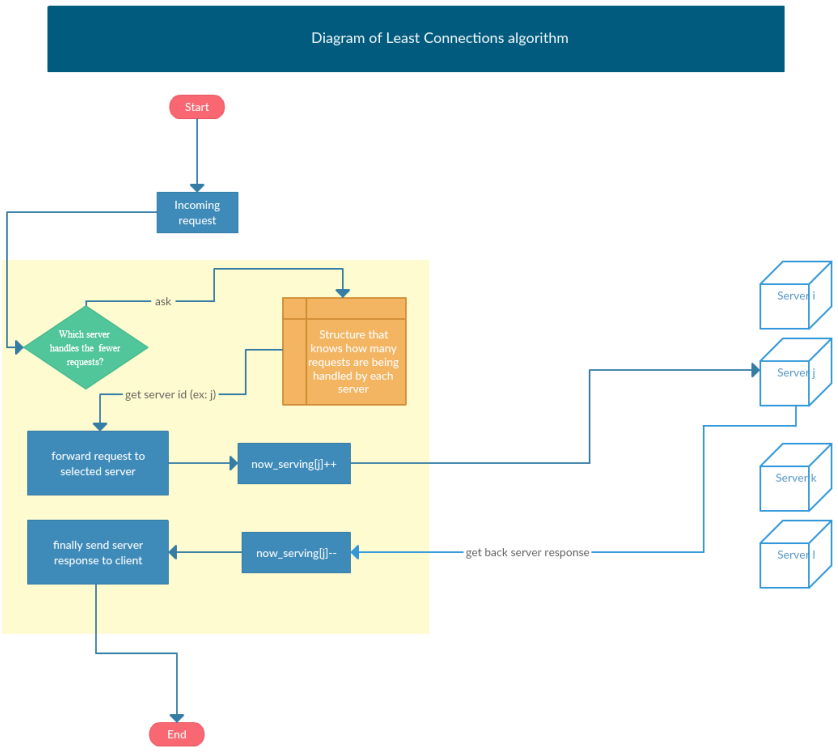
Ο συγκεκριμένος αλγόριθμος έχει ως βασική αρχή την προώθηση του κάθε αιτήματος στο server που διαχειρίζεται τα λιγότερα αιτήματα εκείνη τη στιγμή. Κάθε φορά που έρχεται ένα νέο αίτημα, χρησιμοποιώντας τις κατάλληλες συναρτήσεις βρίσκουμε ποιος από τους 4 εξυπηρετητές διαχειρίζεται τα λιγότερα αιτήματα τη συγκεκριμένη στιγμή. Το αίτημα προωθείται σε αυτόν και φυσικά αυτή η πληροφορία καταγράφεται.

Κάθε φορά που ένα αίτημα εξυπηρετείται από το καθένα από τα 4 εικονικά μηχανήματα και η απάντηση επαναπροωθείται στο load balancer ενημερώνεται και η πληροφορία για το πόσα αιτήματα διαχειρίζεται εκείνη τη στιγμή το VM, τα οποία είναι πλέον ένα λιγότερο.

```
int least_conn_value=servers_container->now_serving[0];
for (i=1; i<=3; i++){
    if (servers_container->now_serving[i] < least_conn_value){
       least_conn_index = i;
       least_conn_value = servers_container->
now_serving[i];
    }
}
servers_container->now_serving[least_conn_index]++;
*served_by_idx=least_conn_index; //κραταμε για να δουμε ποιος
ειναι και να εχει ο απο πανω την πληροφορια
return servers_container->
servers[least_conn_index].ipaddress;
```

Εικόνα 3.5 Εδώ ο αλγόριθμος βρίσκει το server με τις ελάχιστες συνδέσεις, αυξάνει το μετρητή συνδέσεών του και επιστρέφει την ip του

Ο συγκεκριμένος αλγόριθμος δεν κρατάει απολύτως καμία άλλη πληροφορία όπως στατιστικά των εικονικών μηχανημάτων ή το πόσο χρόνο έκανε το κάθε αίτημα επεξεργαστεί από το κάθε εικονικό μηχάνημα. Η βασική ιδέα του αλγορίθμου αυτού είναι πως όσο λιγότερα αιτήματα διαχειρίζεται ένας εξυπηρετητής τόσο λιγότερο φορτωμένος είναι. Οπότε η επεξεργασία του κάθε αιτήματος κρατάει λιγότερη ώρα σε σχέση με τους άλλους. Αν για κάποιο λόγο (λόγω φόρτου της εφαρμογής ή άλλων διεργασιών) ο εξυπηρετητής υπερφορτωθεί, τα αιτήματα θα αρχίσουν να συσσωρεύονται με αποτέλεσμα ο αριθμός τους να αυξηθεί τόσο ώστε να μην προτιμάται πλέον ο συγκεκριμένος εξυπηρετητής μέχρι να μειωθεί ο φόρτος του.



Εικόνα 3.6 Διάγραμμα λειτουργίας Least Connections

3.2.3 Least Latency

Ο αλγόριθμος αυτός εστιάζει στην απόδοση των εξυπηρετητών και τους κατατάσσει με βάση το πόσο καθυστερεί ο κάθε ένας να απαντήσει σε ένα δοκιμαστικό αίτημα. Εδώ, αντίθετα από τους δύο προηγούμενους αλγορίθμους η κατάταξη των servers και ο υπολογισμός του βασικού χαρακτηριστικού προς σύγκριση του κάθε server γίνεται περιοδικά και δεν έχει σχέση με τα αιτήματα που καταφθάνουν. Η βασική ιδέα είναι η εξής. Κάθε 15 δευτερόλεπτα ο load balancer κάνει από 1 αίτημα στον κάθε έναν από τους 4 εξυπηρετητές και μετράει πόσο χρόνο κάνει ο καθένας να απαντήσει. Τα αιτήματα αυτά δεν έχουν γίνει από χρήστες αλλά δημιουργήθηκαν από το load balancer καθαρά για χρονομέτρηση της ώρας που θα κάνει ο κάθε εξυπηρετητής να εξυπηρετήσει το αίτημα.

Το πόση ώρα έκανε ο κάθε εξυπηρετητής να απαντήσει χρησιμοποιείται στη συνέχεια για να καθοριστεί το βάρος που θα έχει το κάθε εικονικό μηχάνημα μέχρι τη στιγμή που θα επαναληφθεί αυτή η διαδικασία (για τα απόμενα 15 δευτερόλεπτα). Ανάμεσα στις

δοκιμαστικές μετρήσεις τίποτα δεν αλλάζει αυτά τα βάρη, αλλά μένουν σταθερά όπως έχουν υπολογιστεί μέχρι την επόμενη μέτρηση.

```
while(1){
    total_t = 0;
    total_weight = 0.0;
    for(i=0; i<4; i++){
        char ping_cmd[64];
        sprintf(ping_cmd, "curl --silent -o /dev/null %s -w
%%{time_total}\\n", servers_container->servers[i].ipaddress);
        FILE *ping = popen(ping_cmd, "r");
        char res[8];
        fgets(res, sizeof(res), ping);
        t[i] = (1000 * atof(res) + servers_container->time[i]) /
2; //Ως χρόνος απόκρισης υπολογίζεται ο μέσος όρος των δύο
τελευταίων μετρήσεων
        servers_container->time[i] = t[i];
        pclose(ping);
        printf("time passed: %d\n", (int)t[i]);
        total_t += t[i];
    }
}
```

Εικόνα 3.7 Κομμάτι κώδικα από το thread που τρέχει κάθε 15 δευτερόλεπτα και μετράει το latency των τεσσάρων server

Το βάρος σε σχέση με το χρόνο απάντησης υπολογίζεται ως εξής. Μετριέται σε milliseconds ο χρόνος που κάνει ο κάθε εξυπηρετητής να απαντήσει. Αθροίζουμε τους 4 χρόνους των servers σε μία μεταβλητή. Στη συνέχεια διαιρούμε το συνολικό χρόνο με κάθε ένα από τους χρόνους των εικονικών μηχανημάτων και ο αριθμός που προκύπτει είναι το βάρος για τον κάθε εξυπηρετητή. Όσο μικρότερος είναι ο χρόνος, τόσο μεγαλύτερος θα είναι αυτός ο

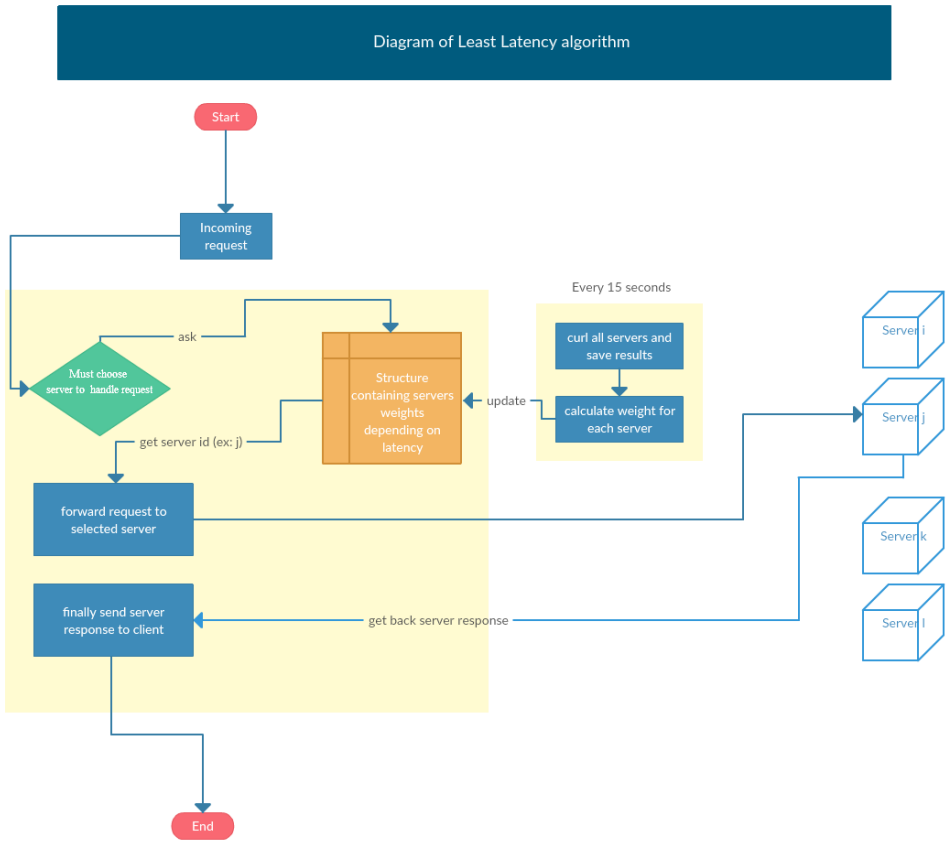
αριθμός. Τελικά ο κάθε ένας αριθμός είναι ένα «παράθυρο». Όσο μεγαλύτερο το παράθυρο τόσο πιο πιθανό να επιλεγεί ο συγκεκριμένος server για το επόμενο αίτημα.

Έτσι λοιπόν για κάθε νέο αίτημα επιλέγεται ο κατάλληλος εξυπηρετητής με τον παρακάτω τρόπο. Η τελική επιλογή γίνεται με τη χρήση της συνάρτησης `rand()`. Χρησιμοποιούμε τη rand για να παράγουμε έναν τυχαίο αριθμό και σε όποιο παράθυρο βρίσκεται ο αριθμός που διάλεξε η rand, αυτός είναι ο εξυπηρετητής στον οποίο θα ανακατευθυνθεί το τρέχον αίτημα. Αυτή η διαδικασία γίνεται κάθε φορά που έρχεται νέο αίτημα στον load balancer. Λόγω του μεγάλου αριθμού των αιτημάτων, η κατανομή τους με τη χρήση του παραπάνω μηχανισμού αντικατοπτρίζει τα βάρη που έχουν υπολογιστεί.

Ο τρόπος με τον οποίο μετριέται την καθυστέρηση (latency) του κάθε εξυπηρετητή είναι το εργαλείο `curl`. Χρησιμοποιώντας το με τα κατάλληλα ορίσματα παίρνουμε ως απάντηση ακριβώς πόση ώρα έκανε ο server να επεξεργαστεί το αίτημα. Ένα πρόβλημα που προέκυψε αρχικά ήταν κάποιες ακραίες τιμές που έδινε το curl σε ορισμένες περιπτώσεις. Ένας μέσος όρος χρόνου απάντησης καθορίζει έναν εξυπηρετητή, σίγουρα όμως υπάρχουν κάποια αιτήματα που θα πάρουν σημαντικά περισσότερο (ή λιγότερο) χρόνο από το αυτόν το μέσο όρο. Έτσι λοιπόν αν ο μέσος όρος επεξεργασίας ενός αιτήματος είναι 3 δευτερόλεπτα αλλά το συγκεκριμένο αίτημα που θα κάνουμε με το curl για να πάρουμε τον ενδεικτικό μέσο όρο χρόνου απόκρισης είναι για παράδειγμα 6 δευτερόλεπτα, τότε θα υπολογίσουμε λανθασμένα την ικανότητα, και κατά συνέπεια το βάρος του συγκεκριμένου εξυπηρετητή. Έτσι για τα επόμενα 15 δευτερόλεπτα τα βάρη δε θα αντικατοπτρίζουν τη σωστή εικόνα για τη χωρητικότητα των εξυπηρετητών και η κατανομή του φόρτου θα απέχει αρκετά από το ιδανικό. Αυτό θα συμβαίνει γιατί κάποιος εξυπηρετητές θα χρησιμοποιείται λιγότερο από ότι πρέπει ενώ αντίθετα οι υπόλοιποι θα μοιραστούν ένα μεγαλύτερο ποσοστό φόρτου καθυστερώντας και οδηγώντας την απόδοση σε χαμηλότερα επίπεδα από την πραγματική ικανότητα της υποδομής.

Για να μετριαστεί αυτό το πρόβλημα, ο τελικός χρόνος που υπολογίζεται είναι ο μέσος όρος του χρόνου που καταγράφηκε στην τελευταία μέτρηση και του χρόνου που υπολογίστηκε

στην προηγούμενη. Με αυτόν τον τρόπο λαμβάνεται υπόψη και η προηγούμενη μέτρηση, οπότε οι αυξομειώσεις μεταξύ των βαρών είναι πιο ομαλές.



Εικόνα 3.8 Διάγραμμα λειτουργίας Least Latency

3.2.4 Least Latency Alternative

Πρόκειται για μια παραλλαγή του προηγούμενου αλγορίθμου και την πρώτη προσπάθεια για βελτίωση μιας υπάρχουσας μεθόδου. Η βασική ιδέα είναι η ίδια, δηλαδή αποδίδονται βάρη στους 4 εξυπηρετητές ανάλογα με την απόδοσή τους. Η απόδοση και εδώ έχει να κάνει με το πόσο γρήγορα εκτελείται ένα αίτημα. Η διαφορά σε αυτήν την εκδοχή είναι πως ο υπολογισμός των βαρών δε γίνεται κάνοντας περιοδικά δοκιμαστικά αιτήματα στους 4 servers, αλλά χρησιμοποιώντας τα πραγματικά αιτήματα.

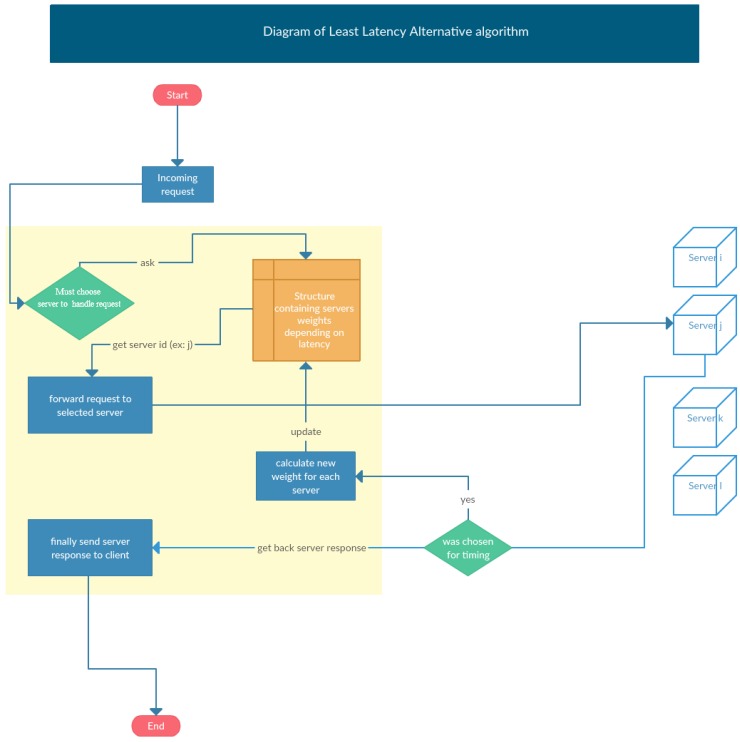
Εδώ επιλέγονται με τυχαίο τρόπο κάποια από τα πραγματικά αιτήματα που έχουν κάνει οι χρήστες για να χρονομετρηθούν. Περίπου ένα στα 200 αιτήματα επιλέγεται ώστε να

χρονομετρηθεί. Κάθε φορά που γίνεται αυτό αλλάζει η τιμή που δείχνει τον ενδεικτικό χρόνο που κάνει να απαντήσει ο αντίστοιχος εξυπηρετητής. Αυτό έχει ως συνέπεια να υπολογίζονται εκ νέου τα σχετικά βάρη των server, πράγμα το οποίο γίνεται όπως στον προηγούμενο αλγόριθμο. Έτσι ανανεώνεται πλέον ο πίνακας των βαρών με βάση τον οποίο γίνεται η κατανομή των νέων αιτημάτων που κάνουν οι χρήστες από εδώ και πέρα.

```
if(rand()%250==0){
  clock_t before, after, t;
  time(&before);
  res = curl_easy_perform(curl);
  time(&after);
  if (res != CURLE_OK) {
    fprintf(stderr, "curl_easy_perform() failed: %s\n",
curl_easy_strerror(res));
  }
  else {
    pthread_mutex_lock(&lb_state_mutex);
    // Υπολογισμός βαρών κάθε φορά που έχουμε νέα τιμή
    weight_calculator(served_by_idx, 100*difftime(after, before));
    pthread_mutex_unlock(&lb_state_mutex);
    write(client_socket, response_str, strlen(response_str));
  }
}
```

Εικόνα 3.9 Χρονομέτρηση request (με πιθανότητα 1/250) με τη βοήθεια της συνάρτησης time

Η μέθοδος αυτή στοχεύει στο να παρακάμψει το πρόβλημα που παρατηρήθηκε στην υλοποίηση της προηγούμενης μεθόδου και έχει σχέση με την ακρίβεια της μέτρησης του latency. Πλέον δε χρησιμοποιείται το εργαλείο curl σε τεχνητά δημιουργημένα αιτήματα, αλλά μετριέται ο χρόνος που παίρνει η επεξεργασία του αιτήματος με το μηχανισμό που εμείς έχουμε υλοποιήσει. Έτσι με αυτό το μηχανισμό μπορούν να συμπεριληφθούν τυχόν ιδιαιτερότητες της υποδομής μας, πράγμα που δε γίνεται με το curl.



Εικόνα 3.10 Διάγραμμα λειτουργίας Least Latency Alternative

3.2.5 Least Latency Improved

Μια ακόμα παραλλαγή του αλγορίθμου least latency, η οποία προσπαθεί να διαχωρίσει η καθυστέρηση του δικτύου από το χρόνο απόκρισης της εφαρμογής. Αποτελεί μια προσπάθεια κατανόησης και ερμηνείας του latency και του πόσο επηρεάζεται από την κάθε παράμετρο. Η λογική είναι η ίδια με την αρχική έκδοση του αλγορίθμου, δηλαδή ανά 15 δευτερόλεπτα γίνεται από ένα αίτημα στον κάθε εξυπηρετητή και με βάση τις μετρήσεις αποδίδεται ένα βάρος σε κάθε έναν. **Οι μετρήσεις εδώ είναι δύο και είναι οι εξής:**

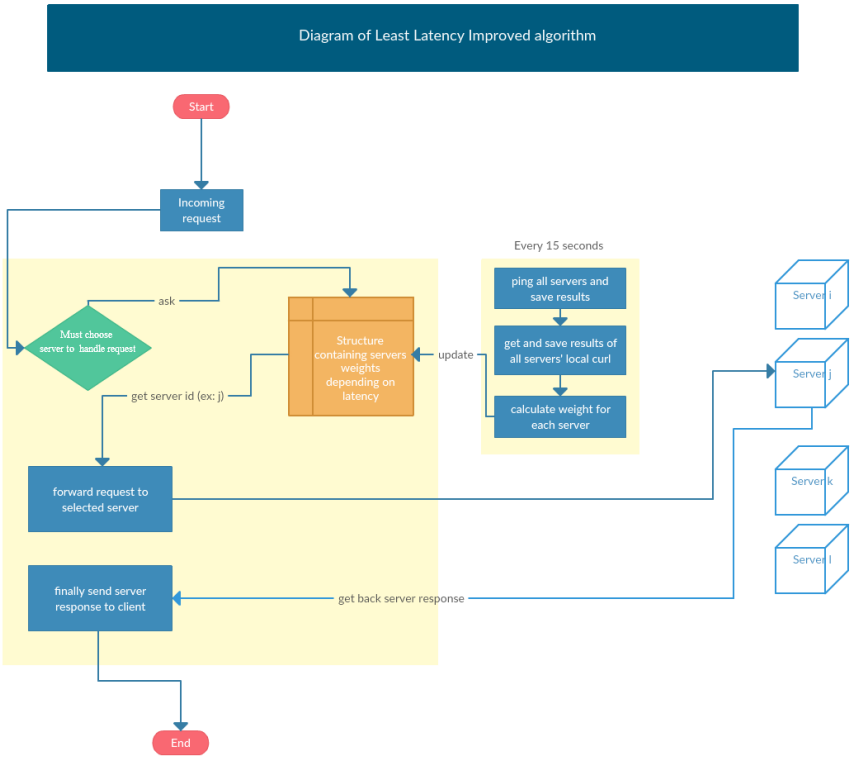
- **Μέτρηση με το εργαλείο ping.** Στοχεύει στο να μετρήσει αποκλειστικά την καθυστέρηση που μπορεί να εμφανίζεται σε έναν εξυπηρετητή λόγω φόρτου δικτύου. Πέρα από την απόδοση που οφείλεται στην επεξεργαστική ισχύ και το φόρτο του εξυπηρετητή σημαντικό ρόλο παίζει και η καθυστέρηση στην απάντηση η ταχύτητα της σύνδεσης μεταξύ του load balancer και του εξυπηρετητή.

- **Η δεύτερη μέτρηση χρησιμοποιεί το εργαλείο curl.** Η διαφορά από την αρχική έκδοση του αλγορίθμου είναι ότι το curl εδώ εκτελείται τοπικά σε κάθε εξυπηρετητή (με προορισμό το localhost) με αποτέλεσμα ο χρόνος που τελικά θα προκύψει ως απάντηση να είναι ενδεικτικός μόνο της απόδοσης της εφαρμογής που τρέχει ο εξυπηρετητής χωρίς να επηρεάζει η καθυστέρηση λόγω δικτύου.

```
while(1){
      for(i=0; i<4; i++){
         char ping_cmd[64], curl_cmd[64];
         sprintf(ping_cmd, "ping -c 3 %s | tail -1 | awk -F '/' '{print
$5}'", servers_container->servers[i].ipaddress);
         FILE *ping = popen(ping_cmd, "r");
         char p_res[8];
         fgets(p_res, sizeof(p_res), ping);
         ping_res[i] = 1000 * atof(p_res);
         pclose(ping);
         sprintf(curl_cmd, "ssh -i %s user@%s curl --silent -o /dev/null
localhost -w %%{time_total}\\n", servers_container-
>servers[i].path_to_idrsa, servers_container->servers[i].ipaddress);
         FILE *curl = popen(curl_cmd, "r");
         char c_res[8];
         fgets(c_res, sizeof(c_res), curl);
         curl_res[i] = 1000 * atof(c_res);
         pclose(curl);
         t[i]=(ping_res[i] + curl_res[i] + (servers_container->
time[i]))/2;
         servers_container->time[i] = t[i];
         total_t += t[i];
     }
}
```

Εικόνα 3.11 Ξεχωριστές μετρήσεις για ping και curl

Χάρη στον παραπάνω διαχωρισμό είναι πιο εύκολο να παρατηρηθεί αν οι διακυμάνσεις του curl στον πραγματικό αλγόριθμο οφείλονται στην απόδοση της εφαρμογής, το φόρτο του δικτύου ή και στα δύο. Εάν παρατηρηθεί μεγάλη σταθερότητα της μια από τις δύο μετρήσεις θα γίνουν δοκιμές δίνοντας ίσως λίγο μεγαλύτερη βαρύτητα σε αυτήν. Αφού έχουν καταγραφεί και οι 2 μετρήσεις, ο load balancer προσθέτει αυτά τα δύο αποτελέσματα για να υπολογίσει το συνολικό χρόνο απόκρισης για κάθε εξυπηρετητή. Τα βάρη που θα καθορίσουν το ποσοστό των αιτημάτων που θα αναλάβει ο καθένας υπολογίζονται πλέον με το συνολικό χρόνο ακριβώς με τον ίδιο τρόπο που υπολογίζονται και στην αρχική μέθοδο.



Εικόνα 3.12 Διάγραμμα λειτουργίας Least Latency Improved

3.2.6 Least Total Time

Ο τελευταίος αλγόριθμος που υλοποιήθηκε είναι ανεξάρτητος από αυτούς που έχουν περιγραφεί μέχρι τώρα. Έγιναν δυο παραλλαγές του least latency ώστε να βελτιωθεί το πρόβλημα με την ακρίβεια της μέτρησης του latency και μία τελείως διαφορετική προσέγγιση που θα περιγραφεί σε αυτήν την ενότητα.

Η κεντρική ιδέα αυτής της μεθόδου είναι το να μοιραστεί όσο το δυνατόν πιο δίκαια μεταξύ των εξυπηρετητών ο χρόνος που αφιερώνεται για επεξεργασία αιτημάτων. Εάν για παράδειγμα αθροιστικά η υποδομή έχει αφιερώσει 20 λεπτά για επεξεργασία αιτημάτων, τότε ιδανικά πρέπει να έχει αφιερώσει 5 λεπτά ο κάθε εξυπηρετητής. Ξεκινώντας λοιπόν από μια ισορροπία στην αρχή, ξεκινάνε να μοιράζονται τα αιτήματα στους εξυπηρετητές. Με το που τελειώνει η επεξεργασία ενός αιτήματος και η απάντηση επιστρέψει στον load balancer σημειώνεται ο χρόνος που αφιέρωσε αυτός ο εξυπηρετητής στην επεξεργασία.

Η ανάθεση για επεξεργασία νέων αιτημάτων γίνεται λοιπόν στον εξυπηρετητή που έχει αφιερώσει τη λιγότερη ώρα από όλους στην επεξεργασία αιτημάτων. Με αυτόν τον τρόπο ένας εξυπηρετητής που γενικά αφιερώνει αρκετό χρόνο στην επεξεργασία αιτημάτων θα πάρει συνολικά λιγότερα αιτήματα από κάποιον άλλον που τα επεξεργάζεται γρηγορότερα. Είναι σωστό αυτό επειδή το ότι κάποιος εξυπηρετητής κάνει περισσότερη ώρα να επεξεργαστεί ένα αίτημα σημαίνει πως είναι πιο φορτωμένος, άρα είναι καλύτερο να προτιμηθεί κάποιος άλλος για να διαχειριστεί το νέο αίτημα.

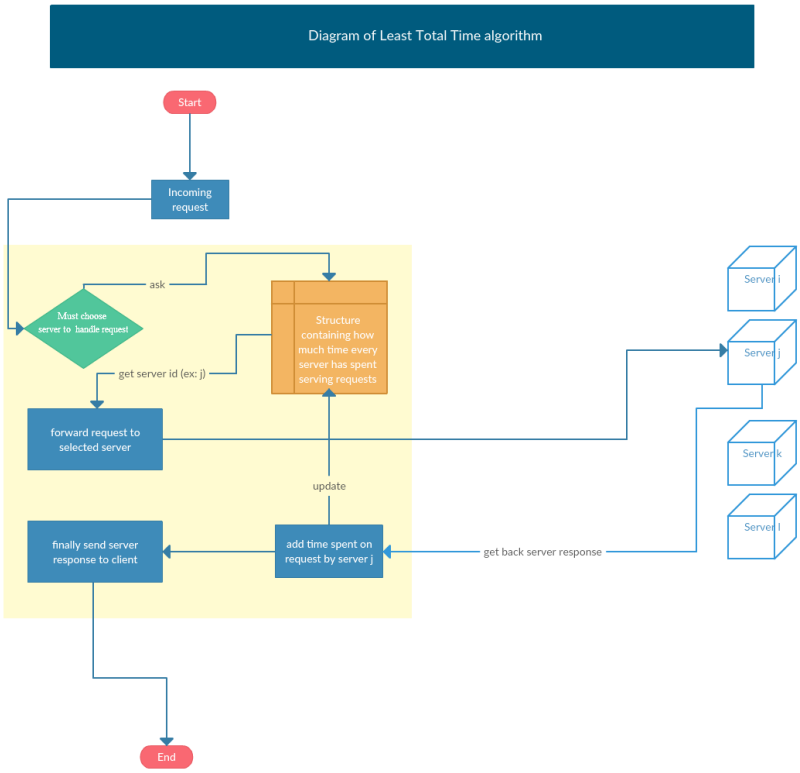
```
int least_time_value=servers_container->ms_served[0];
   for (i=1; i<=3; i++){
      if (servers_container->ms_served[i] < least_time_value){
         least_time_index = i;
         least_time_value = servers_container->ms_served[i];
      }
   }
// Βρέθηκε ο server που έχει αφιερώσει το λιγότερο χρόνο σε εξυπηρέτηση αιτημάτων
   servers_container->ms_served[least_time_index]+=1400;
// Προστίθενται 1400ms στην τιμή του χρόνου που έχει καταναλώσει. Ο λόγος εξηγείται στην επόμενη παράγραφο
   pthread_mutex_unlock(&lb_state_mutex);
   *served_by_idx=least_time_index;
return servers_container->servers[least_time_index].ipaddress;
```

Εικόνα 3.13 Ο αλγόριθμος βρίσκει τον κατάλληλο server Για να του στείλει το request

Ένα πρόβλημα που παρατηρήθηκε στην αρχική έκδοση αυτής της μεθόδου ήταν η συσσώρευση των αιτημάτων ανά ομάδες σε έναν συγκεκριμένο εξυπηρετητή. Ο ρυθμός ερχομού των αιτημάτων που έρχονται από το jmeter είναι σταθερός αλλά ο ρυθμός με τον οποίο γίνεται η επεξεργασία και η απάντηση δε μπορεί να ακολουθεί τον ίδιο σταθερό ρυθμό. Έτσι μπορεί να τύχει το φαινόμενο σε κάποιες μονάδες του χρόνου ολοκληρωθούν πολλά αιτήματα μαζί και σε κάποιες άλλες κανένα. Στη δεύτερη περίπτωση, εφ’ όσον δεν έχουμε ολοκλήρωση κανενός αιτήματος, σημαίνει πως ο πίνακας των βαρών δε θα ανανεωθεί μέσα σε αυτό το διάστημα. Άρα όλα τα αιτήματα που θα καταφθάσουν στο load balancer θα καταλήξουν στον εξυπηρετητή που έχει αφιερώσει εκείνη τη στιγμή λιγότερο χρόνο από όλους σε εξυπηρέτηση αιτημάτων. Αυτό τις περισσότερες φορές οδηγεί σε υπερφόρτωση αυτού του εξυπηρετητή και δυσανάλογο επιμερισμό του φόρτου, με αποτέλεσμα να παρατηρούνται καθυστερήσεις από το συγκεκριμένο εξυπηρετητή λόγω

υπερφόρτωσης ενώ οι υπόλοιποι χρησιμοποιούνται λιγότερο απ' όσο θα έπρεπε. Τελικά αυτό επηρεάζει τη συνολική απόδοση της υποδομής.

Αυτό το φαινόμενο μετριάζεται ενημερώνοντας τα στατιστικά του server και στην αρχή όταν του αναθέτουμε το αίτημα. Κάθε εξυπηρετητής, με το που αναλαμβάνει να επεξεργαστεί ένα νέο αίτημα, αυτόματα αυξάνεται και η μεταβλητή που δείχνει το χρόνο που έχει καταναλώσει αυτός ο server σε χρόνο επεξεργασίας. Ο χρόνος που προστίθεται είναι ίδιος για όλους, και προαποφασιμένος με βάση κάποια στατιστικά και μετρήσεις που έχουν γίνει. Όταν τελειώσει η επεξεργασία του αιτήματος και έχει έρθει η ώρα να ενημερωθεί ο load balancer, τότε στο τέλος λαμβάνεται υπόψη ο χρόνος που έχει προστεθεί και γίνεται η κατάλληλη προσαρμογή.



Εικόνα 3.14 Διάγραμμα λειτουργίας Least Total Time

3.3 Load generators και εργαλεία δοκιμών

Ακολουθεί σύντομη παρουσίαση των πιο σημαντικών γεννητριών φορτίου που χρησιμοποιούνται για cloud testing.

- **Webload**.[8] Εργαλείο δημιουργία φόρτου και ελέγχου απόδοσης για διαδικτυακές εφαρμογές. Το WebLOAD επιτρέπει στο χρήστη να κάνει stress test σε οποιαδήποτε εφαρμογή στο Διαδίκτυο που χρησιμοποιώντας πολυάριθμες web τεχνολογίες. Ο χρήστης μπορεί να δημιουργήσει φόρτο είτε από το υπολογιστικό νέφος είτε από τα τοπικά μηχανήματα που έχει στη διάθεσή του. Ένα από τα πλεονεκτήματα του συγκεκριμένου εργαλείου είναι η ευκολία που προσφέρει για χρήση features όπως DOM-based recording/playback και η χρήση της Javascript για τη δημιουργία scripts. Το εργαλείο προσφέρει μεγάλης κλίμακας τεστ για την απόδοση με μεγάλο φόρτο χρηστών και πολύπλοκα σενάρια χρήσης και προσφέρει λεπτομερή ανάλυση πάνω στη λειτουργία και την απόδοση της εφαρμογής μας. Είναι πολύ αξιόλογο εργαλείο και έχει κερδίσει αρκετά βραβεία στον τομέα του.

- **LoadComplete:**[9] Το LoadComplete είναι αρκετά εύχρηστο και επιτρέπει στο χρήστη να φτιάξει και να εκτελέσει ρεαλιστικά τεστ απόδοσης για ιστοσελίδες και διαδικτυακές εφαρμογές. Αυτοματοποιεί τη διαδικασία δημιουργίας των τεστ καταγράφοντας τη συμπεριφορά πραγματικών χρηστών. Στη συνέχεια χρησιμοποιεί δεδομένα για τη συμπεριφορά τους με τα οποία δημιουργεί εικονικούς χρήστες που τα προσομοιώνουν. Αυτοί βρίσκονται είτε στο cloud είτε στα φυσικά τοπικά μηχανήματα. Το LoadComplete μας βοηθάει να ελέγξουμε την απόδοση του server μας κάτω από μεγάλο φόρτο, να ανακαλύψουμε τα όριά του και να εκτιμήσουμε από ποιόν αριθμό χρηστών και μετά υπάρχει ανάγκη για scalability. Επιπλέον προσφέρει λεπτομερείς αναλύσεις και αναφορές που βοηθάνε σε λεπτομερή μελέτη της απόδοσης της υποδομής, τη συμπεριφορά της εφαρμογής και την τελική εμπειρία χρήστη.

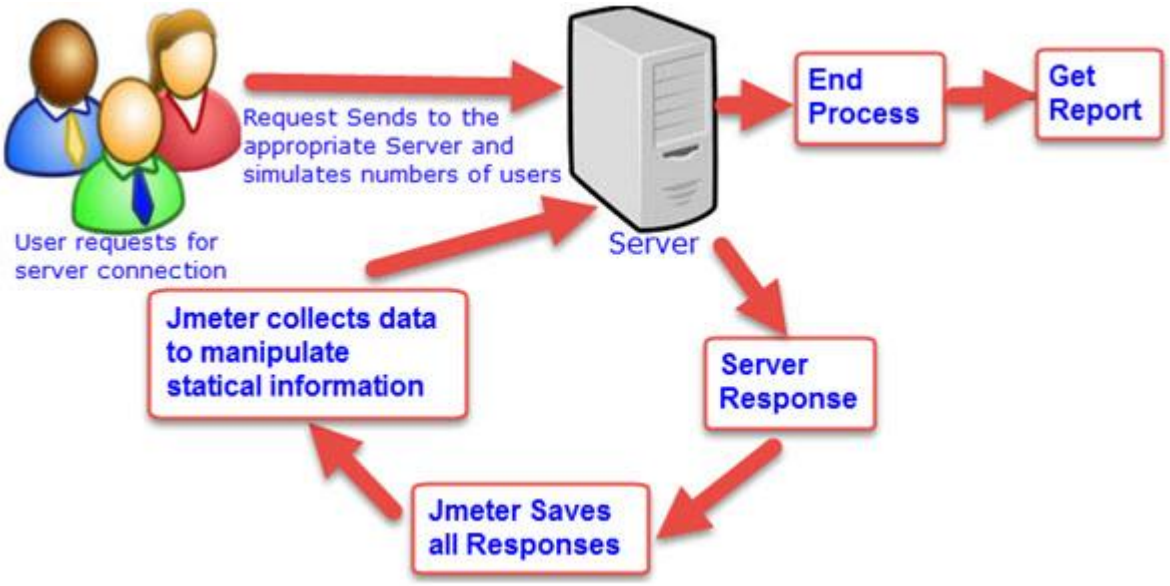
- **Apache Jmeter.**[10] Είναι μια java εφαρμογή ανοιχτού λογισμικού και αποτελεί ένα εξαιρετικό εργαλείο μέτρησης απόδοσης που μπορεί να ενσωματωθεί στο πρόγραμμα δοκιμών μιας οποιασδήποτε εφαρμογής που προσφέρει υπηρεσίες πάνω από δίκτυο.

[8] http://www.radview.com/webload-download/

[9] https://smartbear.com/product/loadcomplete/overview/

[10] http://jmeter.apache.org/

Εκτός από τεστ απόδοσης προσφέρει και λειτουργικά τεστ. Το εργαλείο αυτό εγκαθίσταται σε έναν εξυπηρετητή ή δίκτυο για να μετρήσουμε την απόδοσή του και να αναλύσουμε τη συμπεριφορά του κάτω από διαφορετικές συνθήκες. Εκτός από το βασικό πακέτο υπάρχει ένας μεγάλος αριθμός από plugins τα οποία προσφέρουν πολύ ισχυρά και χρήσιμα εργαλεία στο χρήστη που θα επιλέξει το jmeter σαν εργαλείο μετρήσεων. Αρχικά δημιουργήθηκε μόνο για testing διαδικτυακών εφαρμογών αλλά στη συνέχεια οι ικανότητές του διευρύνθηκαν. Είναι πάρα πολύ χρήσιμο στο να τεστάρει την απόδοση οντοτήτων όπως Servlets, Perl scripts και Java objects.



Εικόνα 3.15 Χρήση του Jmeter για μέτρηση της απόδοσης του server (*Πηγή:* www.techgig.com)

- **Hp LoadRunner.**[11] Είναι ένα προϊόν την HP που χρησιμοποιείται σαν εργαλείο testing. Είναι πολύ χρήσιμο για να δείχνει στο χρήστη τη συμπεριφορά του συστήματος και τα αποτελέσματα όταν υπάρχει πραγματικός φόρτος. Προσφέρει ένα σύνολο από τιμές μετρήσεων σε διάφορες παραμέτρους και δίνει τη δυνατότητα να ελεγχθεί με μεγάλη ακρίβεια η ταχύτητα της εφαρμογής. Επιπλέον μπορεί να αναλύσει και να παρουσιάσει σημαντικά στοιχεία της υποδομής. Ένα από το σημαντικότερα στοιχεία του είναι πως μπορεί να δημιουργήσει και να διαχειριστεί χιλιάδες χρήστες ταυτόχρονα. Στην ουσία

[11] http://www8.hp.com/us/en/software-solutions/loadrunner-load-testing/

αποτελείται από ένα σύνολο αρκετών εργαλείων μερικά από τα οποία είναι το Virtual User Generator, Controller, Load Generator, Analysis etc. [[25]].

- **Rational Performance tester.**[12] Έχει αναπτυχθεί από την IBM. Είναι παραπάνω από ένα αυτοματοποιημένο εργαλείο μέτρησης απόδοσης που μπορεί να χρησιμοποιηθεί για testing μιας διαδικτυακής εφαρμογής. Μπορεί να μετρήσει και να αξιολογήσει την επικοινωνία μεταξύ δυο οντοτήτων που στηρίζονται στην αρχιτεκτονική client-server δημιουργώντας ένα demo που μοντελοποιεί την επικοινωνία μεταξύ του τελικού χρήστη και της εφαρμογής. Στο τέλος όλα τα στατιστικά συλλέγονται και αναλύονται ώστε να αυξηθεί η αποτελεσματικότητα. Έχει τη δυνατότητα να εντοπίζει τα bottlenecks είτε στο κομμάτι της επικοινωνίας είτε στη μεριά του εξυπηρετητή. Έτσι ο χρήστης βελτιώνει τις αδυναμίες αλλάζοντας ακόμα και πράγματα στην αρχιτεκτονική, παρά κάνοντας μόνο μικρές αλλαγές σε παραμέτρους. Αυτό το εργαλείο ίσως είναι η καλύτερη επιλογή για να χτιστεί μια αποδοτική και χωρίς λάθη υπηρεσία βασισμένη στο υπολογιστικό νέφος.

- **NeoLoad**.[13] Ένα ακόμα εργαλείο μέτρησης φόρτου και απόδοσης το οποίο είναι γραμμένο σε java και ο χρήστης μπορεί να εγκαταστήσει όπου επιθυμεί. Χρησιμοποιείται για την ανάλυση ταχύτητας και απόδοσης ιστοσελίδων. Προσφέρει ένα είδος αξιολόγησης των αποτελεσμάτων. Αυτή η διαδικασία βοηθάει στη βελτιστοποίηση της εφαρμογής μας. Κατά τη διάρκεια των δοκιμών ελέγχει την απόδοση αυξάνοντας προοδευτικά τους χρήστες, και έτσι μπορεί να μετρηθεί η απόδοση σε όλα τα στάδια. Με αυτόν τον τρόπο εξάγονται ασφαλή συμπεράσματα για την απόδοση. Έτσι ο διαχειριστής της εφαρμογής μπορεί να διαπιστώσει πόσους χρήστες μπορεί να διαχειριστεί η εφαρμογή του και την απόδοσή της με βάση τον αριθμό τους.

- **LoadUI**.[14] Εργαλείο ανοιχτού λογισμικού για stress tests. Ένα ακόμα εργαλείο για δημιουργία φόρτου και μέτρηση απόδοσης διαδικτυακών εφαρμογών. Συνδυάζεται και

[12] http://www-03.ibm.com/software/products/en/performance

[13] https://www.neotys.com/neoload/overview

[14] https://www.loadui.org/

συνεργάζεται με το εργαλείο SoapUI[15] που έχει σχεδιαστεί για functional testing. Μια πολύ σημαντική λειτουργία που προσφέρει είναι ότι επιτρέπει δημιουργία και τροποποίηση των τεστ ακόμα και την ώρα που τρέχουν. Το γραφικό του περιβάλλον το κάνει εύχρηστο για ένα πολύ μεγάλο φάσμα χρηστών αφού προσφέρει ακόμα και λειτουργίες όπως drag and drop. Δεν είναι ένα απλό εργαλείο μέτρησης απόδοσης, αλλά προσφέρει πολλές πληροφορίες και εκτενή ανάλυση σε βάθος η οποία ανανεώνεται κατά τη διάρκεια των τεστ και με βάση πληροφορίες που αναλύονται σε πραγματικό χρόνο.

- **WAPT:[16]** Το όνομά του προέρχεται από τα αρχικά (Web Application Performance Tool). Είναι ένα εργαλείο μέτρησης απόδοσης για όλα τα είδη των διαδικτυακών εφαρμογών αλλά και εφαρμογών intranet. Με το WAPT μπορούμε να μετρήσουμε την απόδοση της εφαρμογής σε πολλά διαφορετικά περιβάλλοντα και διαφορετικές συνθήκες φόρτου. Προσφέρει αναλυτική πληροφόρηση για τους εικονικούς χρήστες που έχει δημιουργήσει και την παρουσιάζει κατά τη διάρκεια των τεστ σε πραγματικό χρόνο. **Loadster.**[17] Το Loadster είναι ένα εργαλείο μέτρησης απόδοσης που εγκαθίσταται στον υπολογιστή του χρήστη. Παρέχει web interface από το οποίο ο χρήστης μπορεί να παραμετροποιήσει τα scripts που χρησιμοποιεί. Μπορεί επίσης να παραμετροποιήσει αρκετές δυναμικές μεταβλητές και να ελέγξει τη σωστή συμπεριφορά της εφαρμογής του. Παρέχοντας τη δυνατότητα ελέγχου του εύρους ζώνης που χρησιμοποιείται μπορούμε να προσομοιώσουμε την ύπαρξη πολλών χρηστών που χρησιμοποιούν την εφαρμογή μας ταυτόχρονα. Οι αναφορές αποτελεσμάτων για τα τεστ μπορούν να παραχθούν σε μορφή html για ευκολότερη ανάγνωση και για λόγους παρουσίασης.

- **LoadImpact:[18]** Το LoadImpact χρησιμοποιείται κυρίως για testing σε υπηρεσίες που βασίζονται στο υπολογιστικό νέφος. Επίσης βοηθάει στη βελτιστοποίηση και την παραμετροποίηση οποιασδήποτε διαδικτυακής εφαρμογής. Δοκιμάζει τα όρια της

[15] https://www.soapui.org/

[16] http://www.loadtestingtool.com/

[17] https://www.loadsterperformance.com/

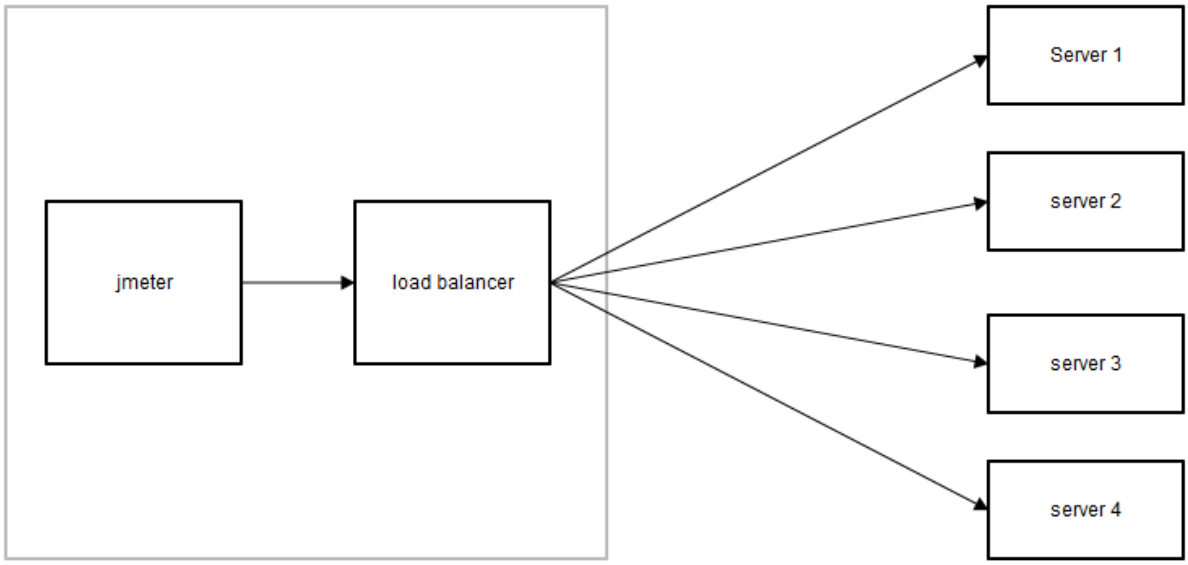
[18] https://loadimpact.com/

εφαρμογής αυξάνοντας σταδιακά το φόρτο μέχρι να βρεθεί ο αριθμός χρηστών ο οποίος την καθιστά μη λειτουργική λόγω φόρτου. Αποτελείται από δύο μεγάλα κομμάτια, το εργαλείο μέτρησης απόδοσης και έναν αναλυτή για ιστοσελίδες. Το πρώτο μπορεί να χωριστεί σε 3 κομμάτια, το Fixed, το Ramp up και το Timeout, καθένα από τα οποία καθορίζει τον τρόπο δημιουργίας των χρηστών. Ο αναλυτής των HTML σελίδων δίνει πληροφορίες σχετικά με τη λειτουργία και τα στατιστικά της ιστοσελίδας.

3.4 Το Apache Jmeter ως γεννήτρια φορτίου

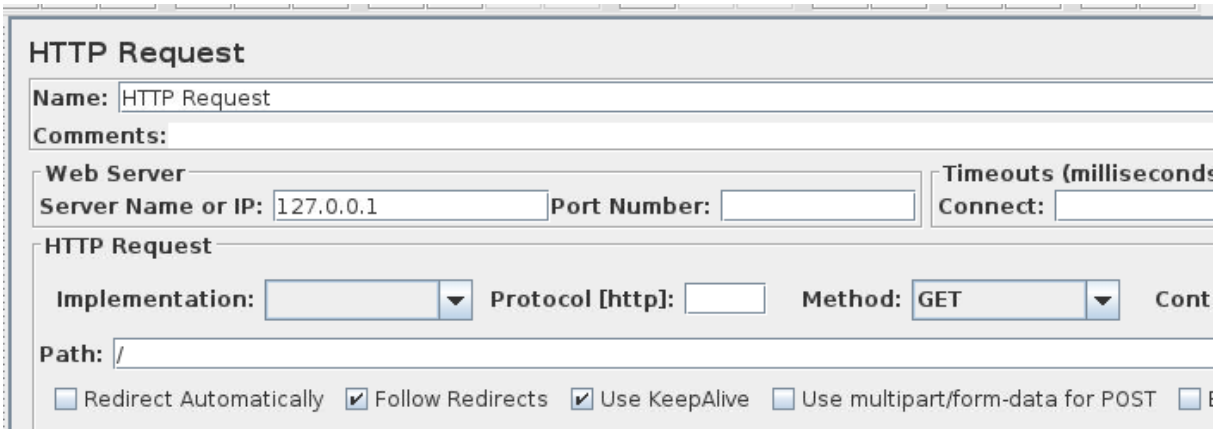
Μετά την αξιολόγηση των εργαλείων για δημιουργία φόρτου και μέτρηση των αποτελεσμάτων επιλέχθηκε το jmeter. Είναι ένα εργαλείο γραμμένο σε java το οποίο δημιουργεί requests τα οποία στέλνει σε μια συγκεκριμένη ip και μετράει τους χρόνους απάντησης. Στα θετικά συγκαταλέγονται το ότι δεν απαιτεί κάποια χρέωση για χρήση χωρίς περιορισμούς σε εύρος ζώνης ή άλλους περιορισμούς όπως τα περισσότερα online εργαλεία, και η εύκολη και πολύ μεγάλη δυνατότητα παραμετροποίησής του μέσω ρυθμίσεων ή plugins τα οποία μπορεί κανείς να προσθέσει πολύ εύκολα.

Στην εικόνα 3.9 φαίνεται ένα σχεδιάγραμμα της υποδομής και η θέση του jmeter σε αυτό. Η εικόνα αυτή απεικονίζει τα πέντε εικονικά μηχανήματα που χρησιμοποιήθηκαν για τα πειράματα στα πλαίσια της παρούσας εργασίας και τις σχέσεις μεταξύ τους. Τα τέσσερα μηχανήματα που τρέχουν τη διαδικτυακή εφαρμογή είναι αυτά που φαίνονται δεξιά στο σχήμα και που δεν επικοινωνούν μεταξύ τους, παρά μόνο με το load balancer. Κανένας από τους αλγορίθμους που υλοποιήθηκαν δεν απαιτεί την επικοινωνία μεταξύ των εικονικών μηχανημάτων, οπότε κανένα μηχάνημα από αυτά δε γνωρίζει για την ύπαρξη, τον αριθμό και τις δυνατότητες των υπολοίπων. Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, υπάρχουν μέθοδοι στις οποίες αυτό είναι απαραίτητο αλλά δε θα εξεταστούν στο πλαίσιο της παρούσας εργασίας.



Εικόνα 3.16 Σχηματική αναπαράσταση των εικονικών μηχανημάτων της υποδομής

Το αριστερό μεγάλο κουτί αναπαριστά το «κεντρικό» εικονικό μηχάνημα. Πάνω σε αυτό τρέχει μόνο ο load balancer και όχι η εφαρμογή, οπότε η μόνη του δουλειά είναι να τρέχει τους αλγορίθμους που παρουσιάστηκαν και να προσπαθεί να εξισορροπήσει το φόρτο στα 4 εικονικά μηχανήματα, ανάλογα με τον αλγόριθμο που τρέχει κάθε φορά.



Εικόνα 3.17 Ρύθμιση jmeter ώστε να στέλνει τα requests στο localhost, όπου τρέχει ο load balancer

Όπως φαίνεται από το σχήμα, το jmeter τρέχει μέσα στο ίδιο εικονικό μηχάνημα με το load balancer. Αυτό έγινε για δύο λόγους. Αρχικά λόγω περιορισμένων πόρων, θα ήταν λάθος

επιλογή να αφιερώσουμε ένα ολόκληρο VM στο jmeter γιατί μετά θα έμεναν μόνο 3 τα οποία τρέχουν την εφαρμογή μειώνοντας τις δυνατότητες της υποδομής του πειράματος. Εκτός από αυτήν την παράμετρο όμως, αυτή η τοποθέτηση του jmeter βγάζει έξω από την εξίσωση την πιθανή καθυστέρηση που θα υπήρχε στο δίκτυο ανάμεσα στο VM του jmeter και τον load balancer. Αυτό το μη μετρήσιμο delay λόγω δικτύου θα επηρέαζε σε άγνωστο βαθμό τα αποτελέσματα και ακόμα χειρότερα τη σύγκριση μεταξύ των μεθόδων.

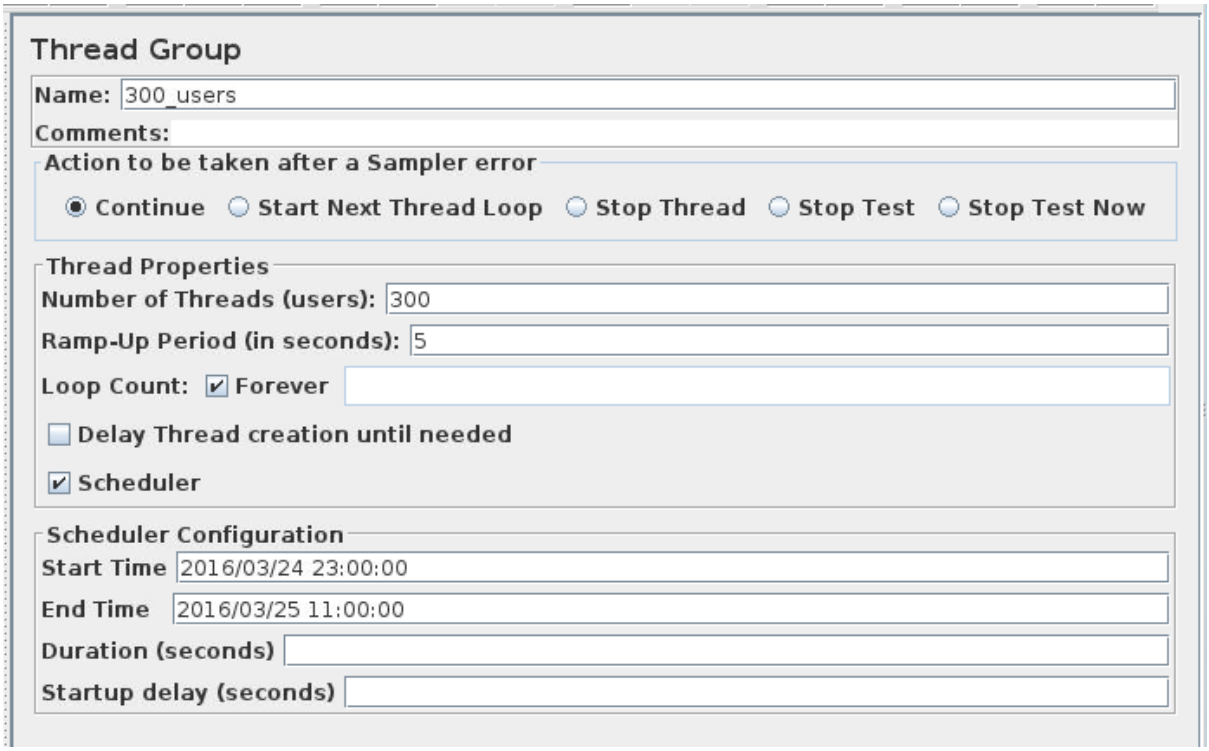
4. ΑΠΟΤΕΛΕΣΜΑΤΑ – ΣΥΖΗΤΗΣΗ

4.1 Περιγραφή πειραμάτων

Σε αυτό το υποκεφάλαιο θα περιγραφεί αναλυτικά ο τρόπος με τον οποίο έγιναν τα πειράματα.

Όπως παρουσιάστηκε στο προηγούμενο κεφάλαιο, το εργαλείο για τη δημιουργία φόρτου και μέτρηση των αποτελεσμάτων είναι το jmeter. Είναι ένα πολύ εύχρηστο εργαλείο και με την προσθήκη plugins μπορεί να κάνει σχεδόν οτιδήποτε θελήσει ο χρήστης. Έτσι το jmeter ήταν το αποκλειστικό εργαλείο μετρήσεων σε όλες τις φάσεις του πειράματος:

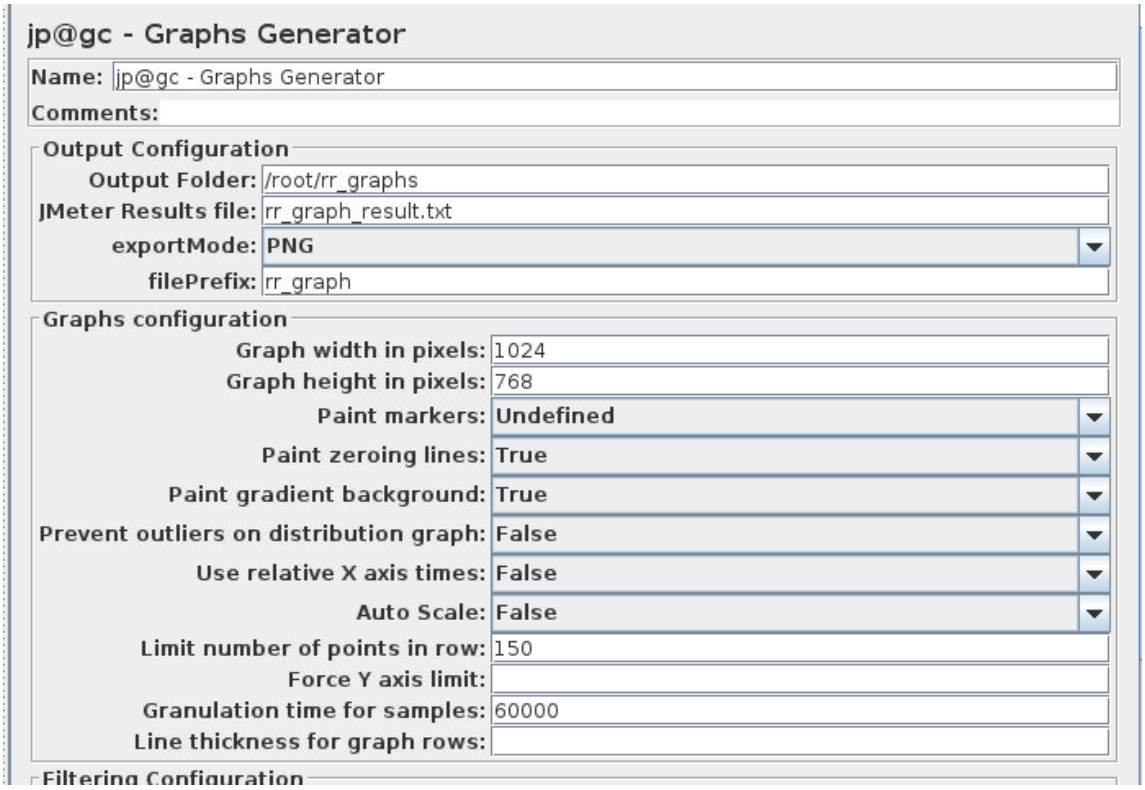
- **Δημιουργία φόρτου.** Αρχικά έπρεπε να αποφασιστεί ο όγκος του φόρτου που θέλουμε να δημιουργήσουμε στην εφαρμογή. Το jmeter δίνει τη δυνατότητα να επιλέξουμε αριθμό χρηστών που θα προσομοιωθούν για τα πειράματα, πόσο συχνά θα κάνει ο καθένας αίτημα στην εφαρμογή, πώς θα κατανέμονται αυτά σε βάθος χρόνου και αρκετές άλλες επιλογές. Για το πείραμα στην παρούσα εργασία επιλέχθηκε να κάνουν αίτημα στο load balancer 300 χρήστες ταυτόχρονα. Εφ’ όσον ένας χρήστης λάβει απάντηση θα περιμένει ένα δευτερόλεπτο και μετά θα εκκινήσει νέο αίτημα προς την εφαρμογή. Αυτή είναι η συμπεριφορά των χρηστών σε όλη τη διάρκεια του πειράματος με όλες τις μεθόδους.



Εικόνα 4.1 Configuration του jmeter για τον αριθμό χρηστών και τις ώρες πειραμάτων

- **Μέτρηση και καταγραφή των μεγεθών.** Στη συνέχεια πρέπει να γίνει μέτρηση και καταγραφή των μεγεθών προς σύγκριση. Το jmeter προσφέρει αρκετά δεδομένα που μπορεί να αναλύσει κάποιος. Στα πλαίσια της παρούσας εργασίας αρκεί η καταγραφή των χρόνων απόκρισης. Αυτό είναι το κύριο χαρακτηριστικό με βάση το οποίο θα γίνουν οι συγκρίσεις. Σκόπιμα επιλέχθηκε να μη δοθεί σημασία σε μετρικές όπως για παράδειγμα χρήση CPU ή μνήμης στα μηχανήματα.

- **Δημιουργία γραφημάτων.** Τέλος χρειάστηκε ένα εργαλείο δημιουργίας γραφημάτων για ευκολότερη σύγκριση και παρουσίαση των αποτελεσμάτων. Μελετήθηκαν κάποια εργαλεία τα οποία παίρνουν δεδομένα και δημιουργούν γραφήματα. Το jmeter όμως έχει δυνατότητα και για απ' ευθείας δημιουργία γραφημάτων από τα αποτελέσματα που παράγει με τη χρήση κάποιων plugins. Οπότε επιλέχθηκε το jmeter και σε αυτό το στάδιο.



Εικόνα 4.2 Ρυθμίσεις για την εξαγωγή των γραφημάτων

Όσον αφορά τη διάρκεια των πειραμάτων ο χρόνος δοκιμών είναι 12 ώρες. Είναι απαραίτητο ένα τόσο μεγάλο διάστημα ώστε οι μετρήσεις να είναι όσο πιο αντικειμενικές γίνεται και τα αποτελέσματα να είναι όσο το δυνατόν πιο άμεσα συνδεδεμένα με την απόδοση παρά με οποιονδήποτε αστάθμητο παράγοντα που μπορεί να επηρεάσει το αποτέλεσμα.

Λόγω του ότι τα πειράματα γίνονται σε κοινόχρηστο cloud περιβάλλον, έπρεπε να ληφθεί υπ’ όψιν ένας ακόμα παράγοντας. Τα πειράματά μας είναι λογικό να επηρεάζονται από τον φόρτο και τη χρήση της πλατφόρμας της υπηρεσίας Okeanos. Ο αριθμός χρηστών και εφαρμογών που κάνουν χρήση της υποδομής είναι πολύ μεγάλος. Αυτό σημαίνει πως η διακύμανση χρήσης και φόρτου της εφαρμογής μπορεί να ποικίλει εισάγοντας έτσι μια υπολογίσιμη αλλά μη ελεγχόμενη παράμετρο στο πείραμα.

Προσπαθήσαμε να μετριάσουμε πόσο επηρεάζει αυτή η παράμετρος το πείραμα με τους παρακάτω τρόπους:

1. Οι μετρήσεις έγιναν ώρες που η υποδομή θα έχει όσο το δυνατόν πιο σταθερή και μειωμένη χρήση από άλλους χρήστες και εφαρμογές. Συγκεκριμένα αποφασίστηκε για όλες τις μετρήσεις να γίνουν ώρες 11 το βράδυ με 11 το πρωί. Αυτές οι ώρες δεν είναι ώρες αιχμής και η απόδοση της πλατφόρμας του ωκεανού θεωρείται πιο σταθερή εκείνες τις ώρες. Λήφθηκε υπόψη ακόμα και το ότι τα σαββατοκύριακα η κίνηση θα είναι ακόμα πιο μειωμένη.

2. Οι μετρήσεις έγιναν αρκετές φορές για κάθε αλγόριθμο. Κάτι που επιβεβαιώνει το ότι η απόδοση του ωκεανού δεν είναι σταθερή, είναι οι παρατηρήσιμες αποκλίσεις σε μετρήσεις του ίδιου αλγορίθμου. Υπάρχουν αποκλίσεις στο χρόνο μετρήσεων έως και 20% όσον αφορά την ίδια μέθοδο. Έτσι για να έχουμε μια ολοκληρωμένη εικόνα για κάθε αλγόριθμο κάναμε το πείραμα αρκετές φορές. Εδώ θα παρουσιαστεί μία 12ωρη μέτρηση για κάθε αλγόριθμο που είναι ενδεικτική της απόδοσής του. Λόγω της δυνατότητας που περέχει το jmeter για την εξαγωγή διαγραμμάτων ανά μέτρηση, η κάθε μία συνοδεύεται από το αντίστοιχο διάγραμμα. Δεν κρίθηκε απαραίτητη η δημιουργία διαγραμμάτων από το σύνολο των μετρήσεων γιατί η εικόνα που θα παρουσιαζόταν θα ήταν ακριβώς η ίδια.

Εκτός από την επιρροή του μη ελεγχόμενου φόρτου στα πειράματα, όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, έπρεπε να δημιουργήσουμε ελεγχόμενα δικό μας φόρτο στους εξυπηρετητές για να προσομοιώσουμε πραγματικές συνθήκες χρήσης και να αποφύγουμε το πρόβλημα που δημιουργεί ο ίδιος φόρτος και στα 4 μηχανήματα με τις υπάρχουσες συνθήκες.

Οι τέσσερεις εξυπηρετητές τρέχουν ακριβώς την ίδια εφαρμογή και έχουν ακριβώς ίδια τεχνικά χαρακτηριστικά. Αυτό λοιπόν καθιστά άχρηστη οποιαδήποτε μέθοδο εξισορρόπησης φορτίου πέρα του round robin, ο οποίος μοιράζει εξίσου το φόρτο και στους τέσσερεις. Έτσι, χρησιμοποιήθηκε το εργαλείο stresslinux για να δημιουργηθεί τεχνητός φόρτος στους εξυπηρετητές.

Σε κάθε server μπήκε ένα cron job το οποίο τρέχει κάθε 5 λεπτά. Δημιουργήθηκε ένα script το οποίο έχει 10% πιθανότητα να καλέσει το stresslinux και να δημιουργήσει σημαντικό φόρτο σε cpu, μνήμη και δίσκο ώστε να ρίξει σημαντικά την απόδοση εκείνου του

εξυπηρετητή για αυτό το χρονικό διάστημα. Σε όλους τους εξυπηρετητές η πιθανότητα αλλά και η ποσότητα φόρτου που δημιουργείται είναι ακριβώς η ίδια ώστε σε βάθος χρόνου να μη βγαίνει κανένας εξυπηρετητής ευνοημένος.

Οι αλγόριθμοι θα πρέπει να αντιλαμβάνονται αυτήν την πτώση στην απόδοση και να προσπαθούν να αποφεύγουν αυτό το μηχάνημα. Είναι πιθανό να τύχει και σε 2 ή περισσότερους server ταυτόχρονα ή σε κανένα να τρέχει το stresslinux πράγμα το οποίο είτε επηρεάζει αρνητικά είτε ευνοεί αντίστοιχα τη συνολική απόδοση της εφαρμογής. Λόγω όμως της μεγάλης διάρκειας της μέτρησης και της μικρής διάρκειας που τρέχει το stresslinux, όλα τα σενάρια θα συμβούν αρκετές φορές σε κάθε πείραμα με όλες τις τεχνικές, πράγμα που διασφαλίζει ότι καμία τεχνική θα βγει τυχαία ευνοημένη.

4.2 Περιορισμοί

Αφού έχει παρουσιαστεί η υποδομή, σε αυτό το υποκεφάλαιο θα αναφερθούν και οι περιορισμοί που προκύπτουν από την παραπάνω πειραματική διάταξη.

Αρχικά, ένας εμφανής περιορισμός είναι ο αριθμός των μηχανημάτων. Έχοντας διαθέσιμα 5 εικονικά μηχανήματα περιοριζόμαστε στη σχετικά απλή διάταξη που παρουσιάστηκε στο προηγούμενο κεφάλαιο. Βεβαίως αυτή η διάταξη είναι αρκετή για να γίνουν οι βασικές συγκρίσεις μεταξύ των αλγορίθμων και να εξαχθούν πολλές χρήσιμες πληροφορίες για αυτούς. Παρ' όλα αυτά θα ήταν χρήσιμο εάν υπήρχαν πολλά περισσότερα διαθέσιμα εικονικά μηχανήματα επειδή σε τέτοια περίπτωση θα μπορούσαν να δοκιμαστούν και άλλα σενάρια (όπως ξαφνική διακοπή λειτουργίας πολλών μηχανημάτων) τα οποία συναντάει κανείς σε cloud υποδομές.

Επιπλέον μία ακόμη παράμετρος που πρέπει να ληφθεί υπόψη είναι η έλλειψη πληροφορίας για το συνολικό φόρτο της πλατφόρμας του ωκεανού. Το πόσο φορτωμένος είναι ο ωκεανός επηρεάζει την απόδοση των αλγορίθμων. Φυσικά αυτό είναι επιθυμητό, αφού μέρος του πειράματος είναι η παρατήρηση της συμπεριφοράς της υποδομής επάνω σε δημόσιο υπολογιστικό νέφος το οποίο συνεπάγεται ταυτόχρονη χρήση της υποδομής από άγνωστο αριθμό χρηστών. Έτσι μπορούμε σύμφωνα με το θεωρητικό υπόβαθρο να εξάγουμε πολλά συμπεράσματα για το φόρτο της υποδομής, τα οποία όμως ισχύουν με

πολύ μεγάλη πιθανότητα και όχι βεβαιότητα, αφού λείπουν οι αριθμοί και τα στατιστικά που θα στοιχειοθετούσαν την απόδειξη αυτών των λογικών ισχυρισμών.

4.3 Παρουσίαση αποτελεσμάτων

Εδώ παρουσιάζονται και αναλύονται τα τελικά αποτελέσματα των μετρήσεων. Ο επόμενος πίνακας παρουσιάζει τους μέσους όρους και την τυπική απόκλιση των τελικών μετρήσεων. Ο μέσος όρος είναι μια μέτρηση που επιτρέπει να έχουμε μια σαφή εικόνα για την απόδοση του αλγορίθμου. Χαμηλότερος μέσος όρος φανερώνει μεγαλύτερη ταχύτητα συνολικά στον τρόπο που ο αλγόριθμος επεξεργάζεται τα αιτήματα. Η τυπική απόκλιση είναι επίσης μια πολύ σημαντική μετρική που είναι χρήσιμη σε ένα τέτοιο πείραμα. Πολύ σημαντικό για ένα σύστημα cloud είναι να μπορεί να προσφέρει στους πελάτες σταθερότητα στην απόδοση ώστε η συμπεριφορά των συστημάτων να είναι όσο το δυνατό πιο αναμενόμενη. Μικρή τυπική απόκλιση σημαίνει πως όλες οι μετρήσεις δεν κυμαίνονται μακριά από το μέσο όρο, άρα ο αλγόριθμος είναι πιο σταθερός από κάποιον με μεγαλύτερη τυπική απόκλιση.

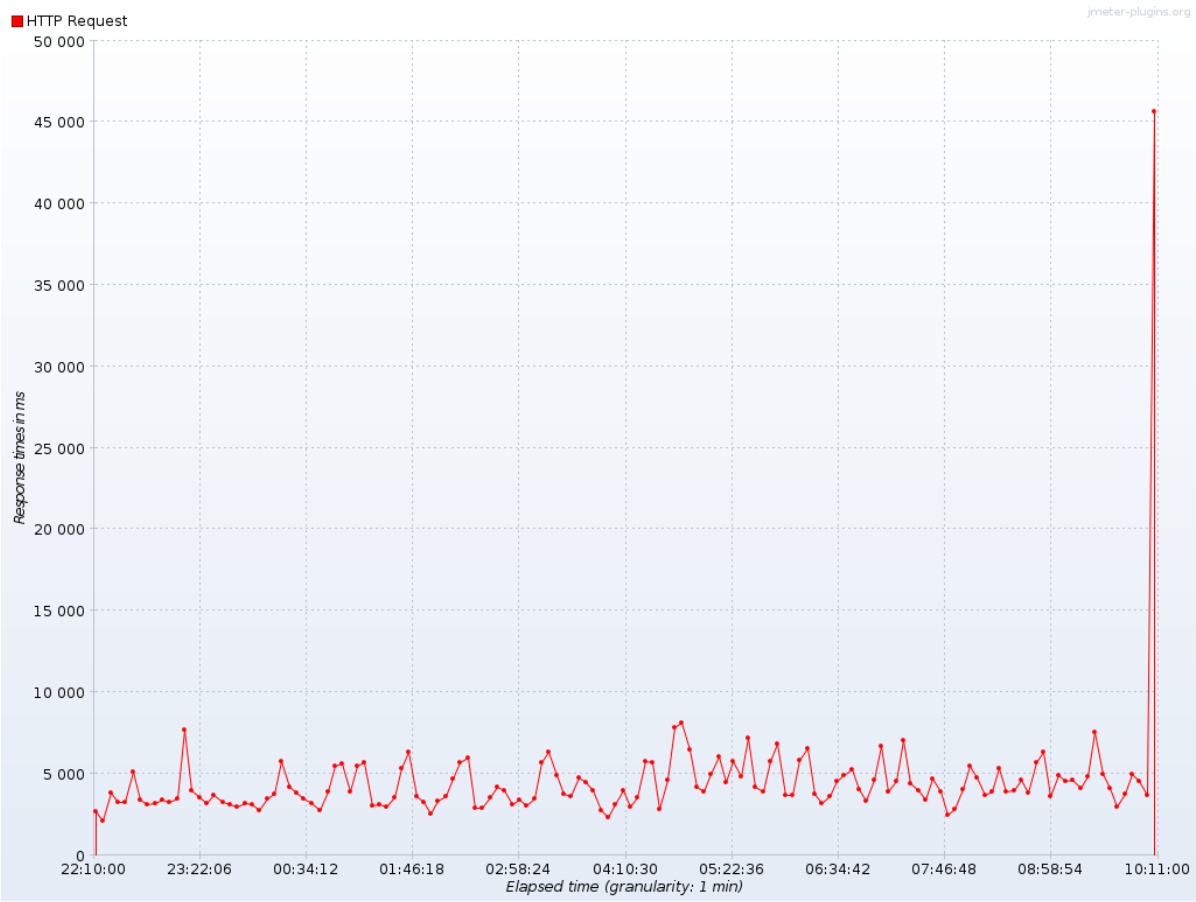
	Round Robin	**Least Connections**	**Least Latency**	**Least Latency Improved**	**Least Latency Alternative**	**Least Total Time**
Average Response time (ms)	3542,005	2083,22	2856,487	2776,841	2705,792	3679,642
Standard Deviation (ms)	5143,807	2074,965	3144,574	3020,843	2990,663	3280,204

Πίνακας 4.1 Αποτελέσματα μετρήσεων αλγορίθμων

Αμέσως εξάγονται πολύ χρήσιμα συμπεράσματα για την απόδοση των αλγορίθμων.

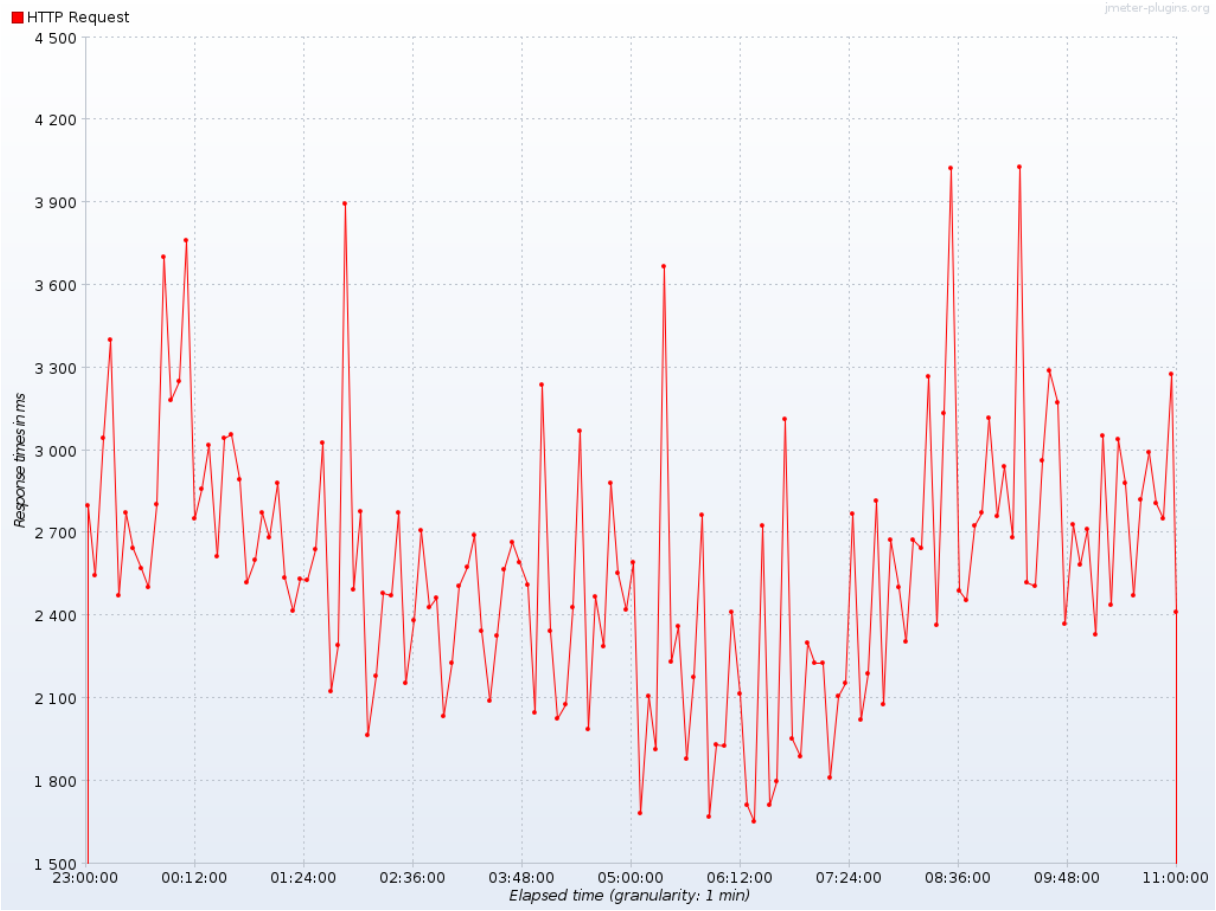
Αρχικά, όσον αφορά το round robin βλέπουμε πως έχει τη χειρότερη απόδοση μεταξύ των γνωστών αλγορίθμων. Και ο μέσος χρόνος απόκρισης είναι κακός αλλά και η τυπική απόκλιση είναι αρκετά μεγάλη, πράγμα που υποδηλώνει μεγάλη αστάθεια στην απόδοση. Λόγω ότι αυτός ο αλγόριθμος δεν έχει κάποια λογική από πίσω η οποία λαμβάνει υπόψη παραμέτρους για να αποφασίσει σε ποιόν θα προωθήσει το request και απλά προωθεί τα αιτήματα εξ ίσου στους εξυπηρετητές με προκαθορισμένη σειρά δε μπορεί να κατανοήσει τις καθυστερήσεις που προκύπτουν από την υπερφόρτωση των server λόγω του stresslinux ή άλλων παραγόντων. Έτσι συνεχίζει να προωθεί αιτήματα με την ίδια συχνότητα σε κάποιο

server είτε είναι πολύ φορτωμένος είτε όχι. Άρα οι ήδη φορτωμένοι εξυπηρετητές δέχονται αιτήματα, κάτι το οποίο ρίχνει την απόδοσή τους ακόμα περισσότερο και κατά συνέπεια την απόδοση ολόκληρης της εφαρμογής. Φυσικά ίδιο αριθμό αιτημάτων δέχονται και οι λιγότερο φορτωμένοι εξυπηρετητές τα οποία όπως είναι φυσικό εξυπηρετούνται αρκετά πιο γρήγορα. Γι' αυτό παρατηρείται και τόσο μεγάλη τυπική απόκλιση.



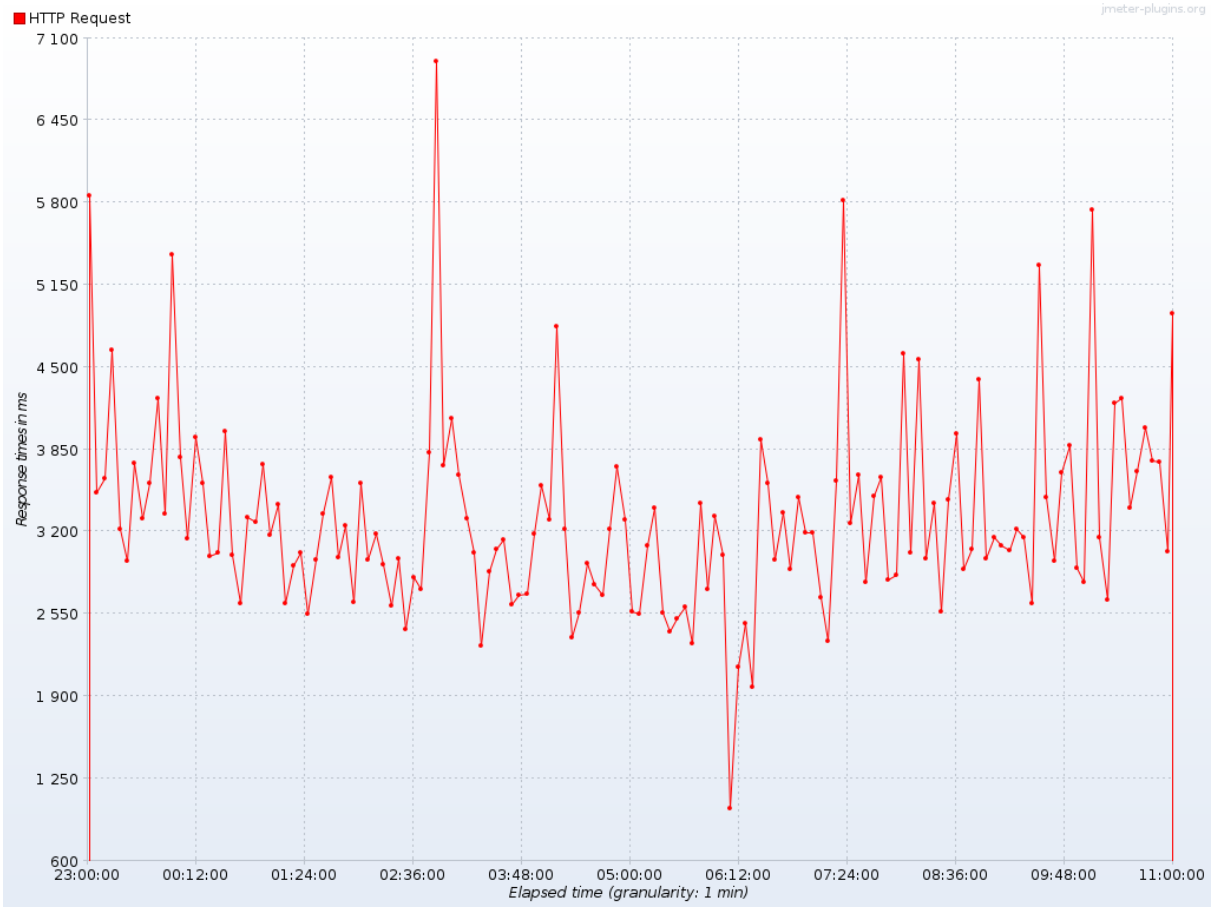
Εικόνα 4.3 Γράφημα απόδοσης Round Robin

Ακολουθεί ο αλγόριθμος least connections ο οποίος είναι και ο πιο γρήγορος αλγόριθμος του πειράματος. Ο μέσος χρόνος εξυπηρέτησης των αιτημάτων από αυτόν τον αλγόριθμο είναι με διαφορά ο πιο μικρός, όπως και η τυπική απόκλιση που δείχνει μεγάλη σταθερότητα στην απόδοση έναντι των υπόλοιπων αλγορίθμων. Η λογική του είναι απλή και γρήγορη ενώ το κύριο κριτήριό του (αριθμός ενεργών συνδέσεων) μπορεί να υπολογιστεί με ακρίβεια.

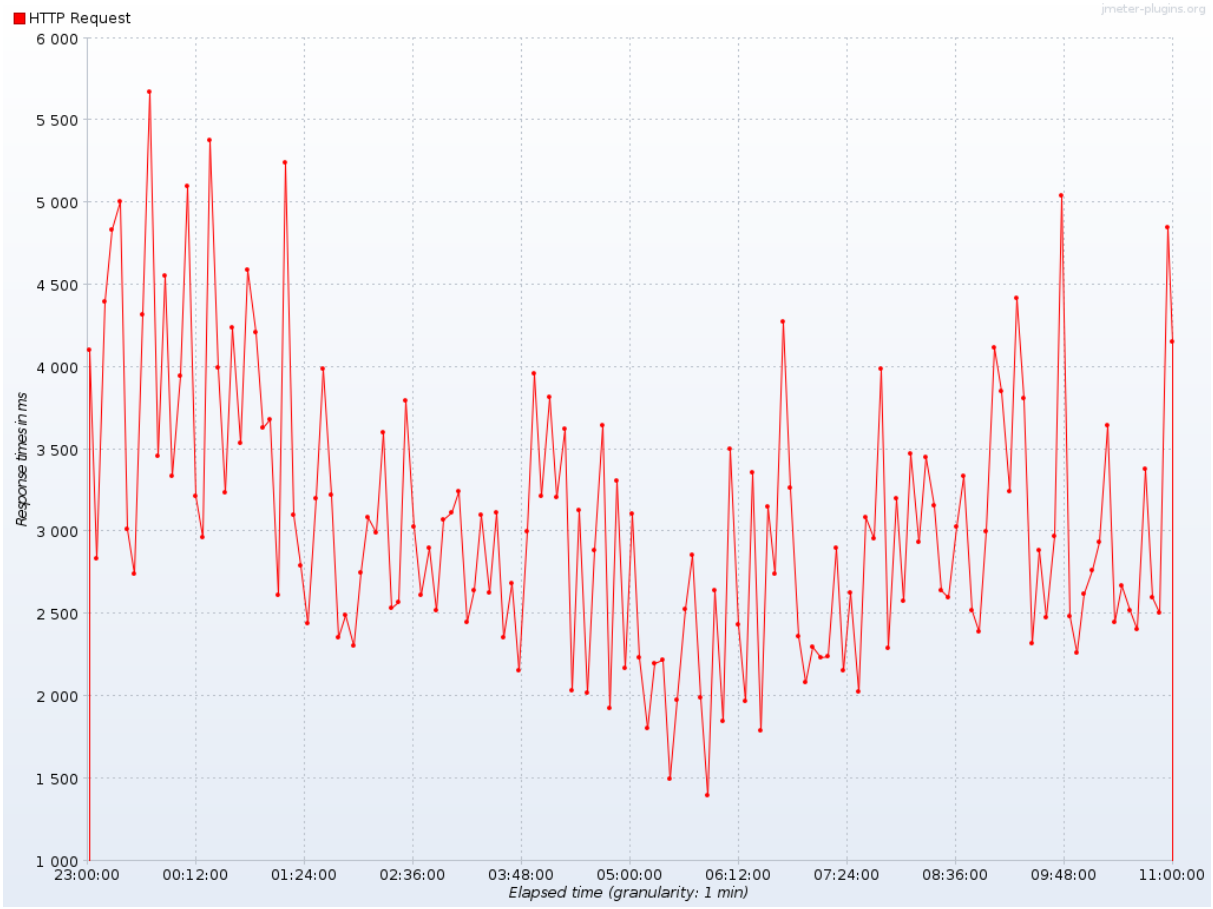


Εικόνα 4.4 Γράφημα απόδοσης Least Connections

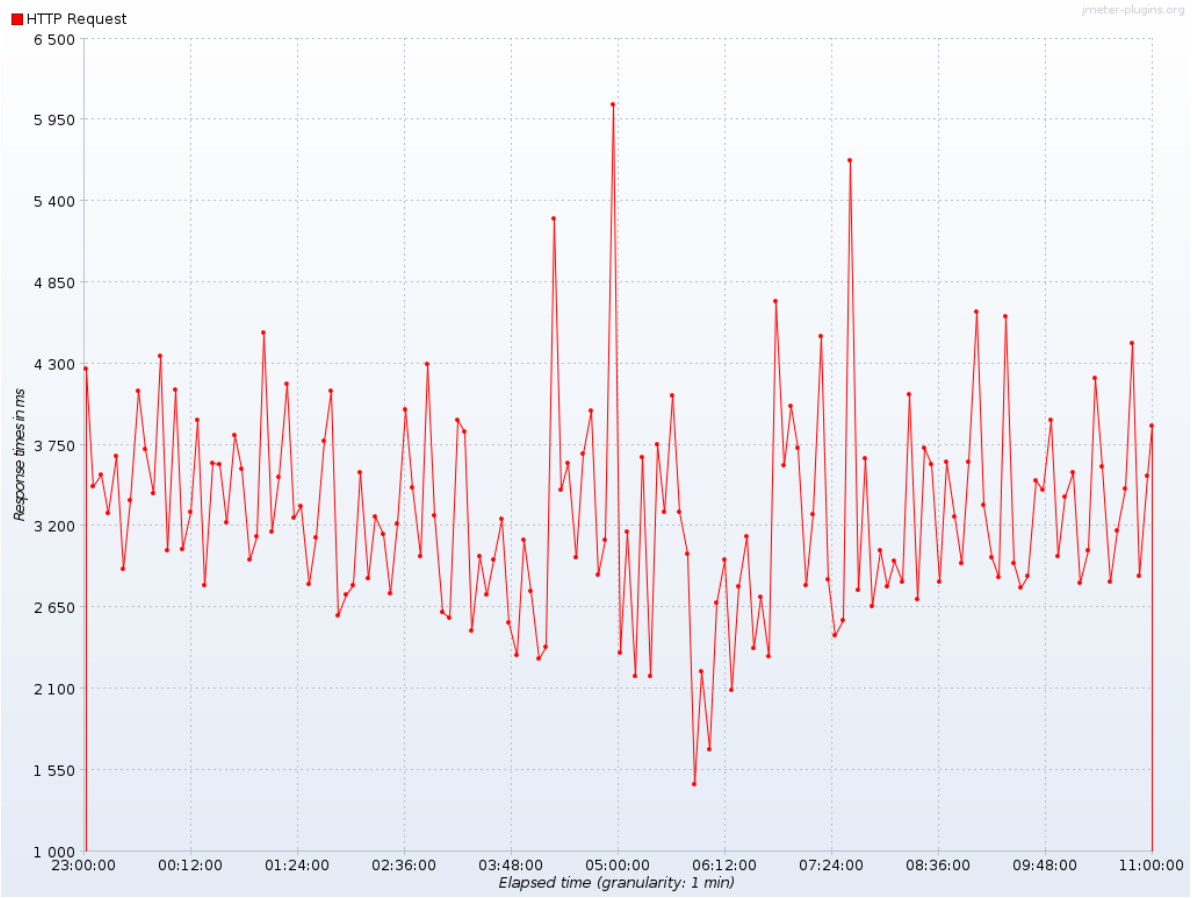
Όσον αφορά τους αλγορίθμους που χρησιμοποιούν σα μετρική το latency τα πράγματα είναι σαφώς πολύ καλύτερα από το round robin αλλά απέχουν απ το να συναγωνιστούν σε απόδοση τον least connections. Οι παραλλαγές που φτιάξαμε βελτίωσαν ελαφρώς την απόδοση αλλά όχι σημαντικά. Και ο μέσος όρος αλλά και η τυπική απόκλιση είναι λίγο βελτιωμένα αλλά οι βελτιώσεις που υλοποιήθηκαν δεν κατάφεραν να υπερκαλύψουν το μεγάλο πρόβλημα, το οποίο είναι η ακριβής μέτρηση του latency.



Εικόνα 4.5 Γράφημα απόδοσης Least Latency



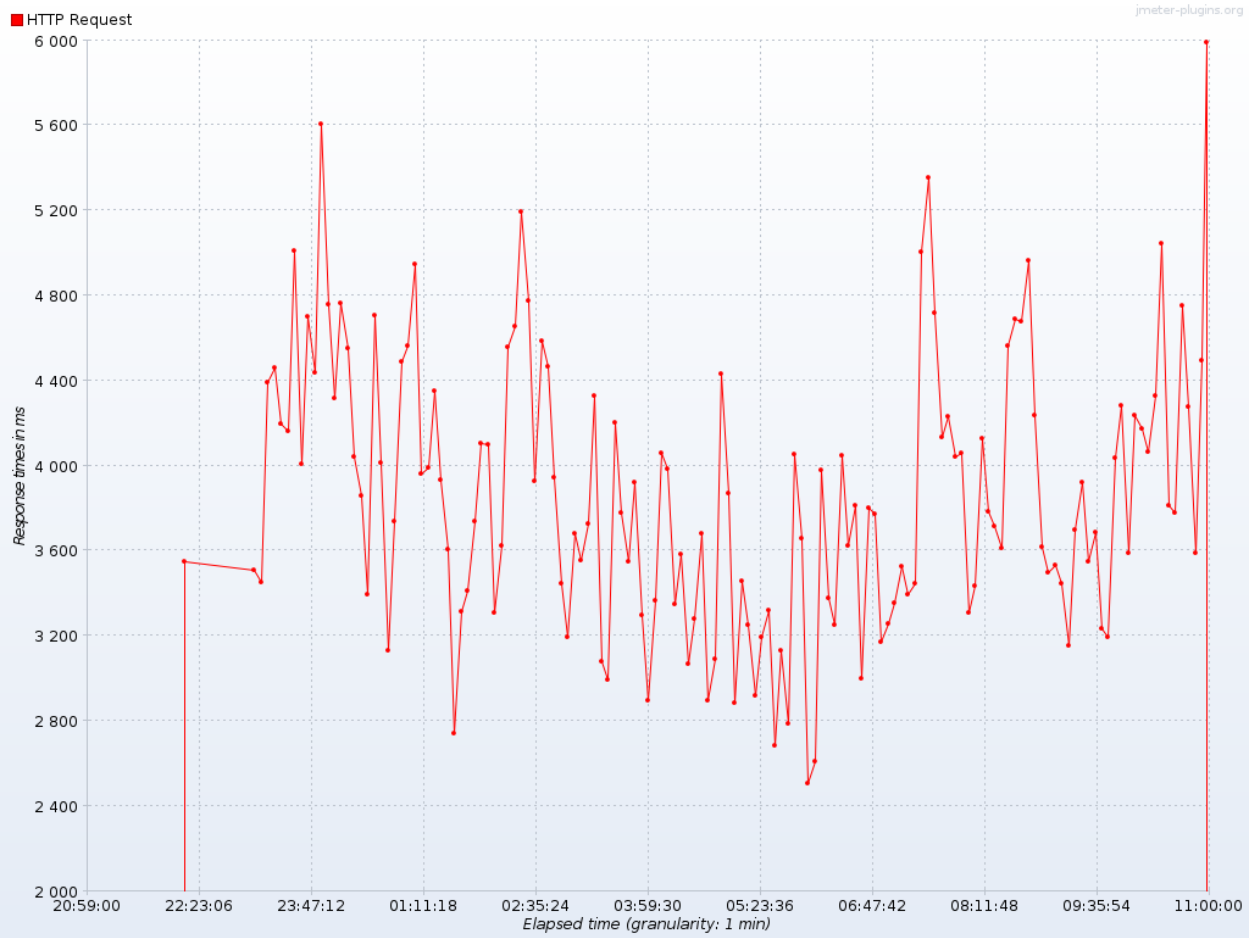
Εικόνα 4.6 Γράφημα απόδοσης Least Latency improved



Εικόνα 4.7 Γράφημα απόδοσης Least Latency alternative

Το συμπέρασμα είναι πως σε αυτό το επίπεδο, ο έλεγχος του latency είναι δύσκολο να είναι ακριβής. Η μέτρηση μόνο του χρόνου απόδοσης της εφαρμογής σποραδικά, είτε μέσω δικτύου είτε τοπικά, δεν είναι 100% ασφαλές στοιχείο για να χρησιμοποιηθεί για την κατανομή φόρτου, λόγω των πολλών παραγόντων που είναι πιθανό να επηρεάσουν τη συγκεκριμένη μέτρηση με βάση την οποία θα υπολογιστεί το βάρος κάθε εξυπηρετητή για ένα σημαντικό χρονικό διάστημα. Σε αυτό το επίπεδο θα ήταν ασφαλέστερο το latency να μετριέται υπολογίζοντας το μέσο όρο αρκετών requests το οποίο όμως αρχίζει να αυξάνει σημαντικά το overhead σε σημείο που πλέον η διαδικασία υπολογισμού του latency και των βαρών είναι κάτι υπολογίσιμο στην τελική απόδοση της εφαρμογής μας. Έτσι διορθώνοντας ένα πρόβλημα θα προκαλούσαμε ένα καινούργιο.

Οι μετρικές latency πρέπει να γίνονται σε πιο χαμηλά επίπεδα και όχι σε αυτό της εφαρμογής. Έτσι μπορούν να ληφθούν υπόψη πιο λεπτομερή στοιχεία για το δίκτυο αλλά και τους πόρους των εικονικών μηχανημάτων, κάτι το οποίο δίνει μια πιο αντικειμενική εικόνα για το latency με σαφώς μικρότερο αντίκτυπο στην απόδοση.



Εικόνα 4.8 Γράφημα απόδοσης Least Total Time

Τέλος, ο αλγόριθμος που υλοποιήθηκε εξ ολοκλήρου από την αρχή, ο least total time, δεν καταφέρνει να ικανοποιήσει τις αρχικές προσδοκίες. Η απόδοση είναι παρόμοια με τον round robin, αν και ο least total time έχει σημαντικά μικρότερη τυπική απόκλιση. Το πρόβλημα αυτού του αλγορίθμου είναι πως δεν έχουμε με ακρίβεια κάθε στιγμή πόσο χρόνο έχει πραγματικά αφιερώσει κάθε server στην εξυπηρέτηση αιτημάτων, κάτι που

αποτελεί τη βασική ιδέα του αλγορίθμου. Το αρχικό πρόβλημα, με το μη υπολογισμό της εξυπηρέτησης των τρέχοντων αιτημάτων, διορθώθηκε προσθέτοντας έναν εικονικό χρόνο σε κάθε εξυπηρετητής κάθε φορά που αναλαμβάνει ένα αίτημα, όπως περιγράφηκε στην αντίστοιχη ενότητα παρουσίασης του αλγορίθμου. Σαφώς και βελτίωσε αρκετά τα πράγματα όμως και πάλι ο αλγόριθμος αυτός δεν καταφέρνει να είναι ανταγωνιστικός.

Παρατηρώντας τα διαγράμματα εξάγεται ένα ακόμα συμπέρασμα, αυτή τη φορά όχι σχετικό με την απόδοση των αλγορίθμων αλλά σχετικά με την υποδομή. Σε όλα σχεδόν τα διαγράμματα μπορεί κανείς εύκολα να παρατηρήσει πως υπάρχει μια γενικότερη πτώση των χρόνων απόκρισης κοντά στη μέση των διαγραμμάτων, ενώ στις άκρες οι χρόνοι απόκρισης είναι ελαφρώς αυξημένοι. Αυτό έχει σχέση και επιβεβαιώνει την υπόθεση που έγινε για την πλατφόρμα του ωκεανού, που ισχύει και για κάθε άλλη πλατφόρμα παροχής υπηρεσιών υπολογιστικού νέφους. Δηλαδή ότι η χρήση από άλλους χρήστες και εφαρμογές επηρεάζει την απόδοση των εικονικών μηχανημάτων όλων των χρηστών.

Στη μέση των διαγραμμάτων απεικονίζονται ώρες πολύ αργά το βράδυ και πολύ νωρίς το πρωί, ένα διάστημα που σαφώς οι ενεργοί χρήστες είναι λιγότεροι από οποιαδήποτε άλλη ώρα. Γι αυτό οι χρόνοι της δικής μας εφαρμογής είναι ελαφρώς καλύτεροι τότε. Αντίθετα στην αρχή του διαγράμματος, όπως και στο τέλος, όπου είναι ώρες που οι χρήστες έχουν αρχίσει και χρησιμοποιούν τους πόρους της πλατφόρμας οι χρόνοι απόκρισης είναι λίγο μεγαλύτεροι.

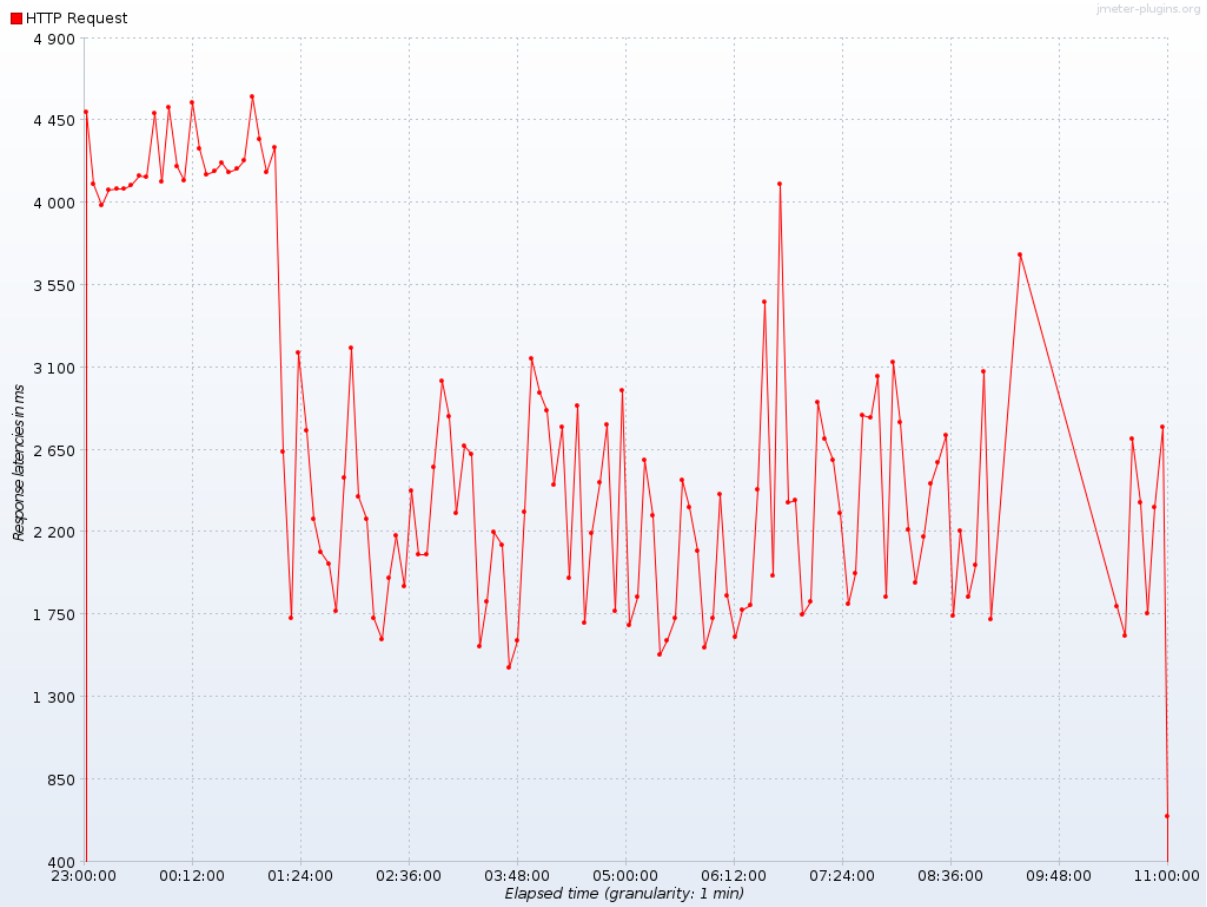
4.4 Σύγκριση με αποτελέσματα από χρήση Nginx

Στην τελευταία ενότητα γίνεται μια παρουσίαση των αποτελεσμάτων για την απόδοση της υποδομής με χρήση του nginx ως load balancer. Ο nginx έχει 3 υλοποιημένους αλγορίθμους για εξισορρόπηση φορτίου. Το round robin, τον least connections και τον iphash. Η 3^{η} τεχνική δεν είναι δυνατό να δοκιμαστεί με την παρούσα υποδομή. Σύμφωνα με τον iphash, το αίτημα προωθείται στον ανάλογο εξυπηρετητή στον οποίο αντιστοιχεί το αποτέλεσμα μια συνάρτηση κατακερματισμού από την οποία περνάει η ip του χρήστη. Στην περίπτωσή μας, εφόσον το jmeter παράγει φόρτο από το ίδιο μηχάνημα και προωθεί τα αιτήματα στο load

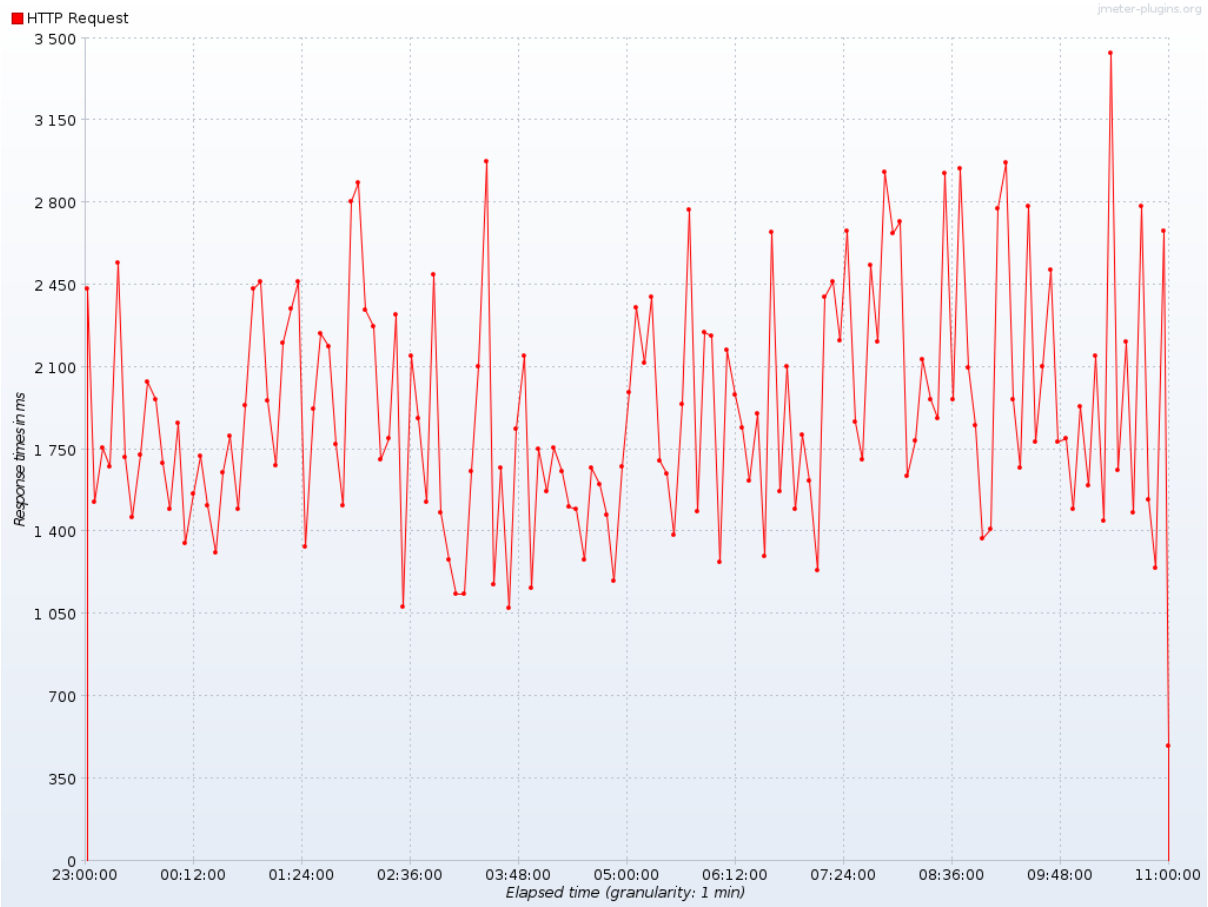
balancer, η ip από την οποία παρέχονται τα σιτήμστς θα είναι πάντα η ίδια. Συνεπώς το αποτέλεσμα της συνάρτησης κατακερματισμού θα είναι πάντα το ίδιο, άρα όλα τα αιτήματα θα προωθούνται στον ίδιο εξυπηρετητή.

Έτσι, θα παρουσιαστούν οι μετρήσεις για το round robin και για τον least connections, των οποίων η απόδοση μπορεί άμεσα να συγκριθεί με τους αντίστοιχους αλγορίθμους που υλοποιήθηκαν για το load balancer που αναπτύχθηκε στα πλαίσια της παρούσας εργασίας.

Η χρησιμοποίηση του nginx ως load balancer ήταν η αρχική λύση που είχαμε επιλέξει. Εγκαταλείφθηκε όμως λόγω μεγάλου φόρτου που θα απαιτούνταν για τροποποίηση και πιθανή βελτίωση των τεχνικών εξισορρόπησης φορτίου. Παρ όλα αυτά, οι μετρήσεις με τον nginx έγιναν και παρουσιάζονται για να γίνει μια σύγκριση του load balancer που υλοποιήθηκε με μία δοκιμασμένη λύση που χρησιμοποιείται ευρέως.



Εικόνα 4.9 Γράφημα απόδοσης Round robin υλοποίησης Nginx



Εικόνα 4.10 Γράφημα απόδοσης Least Connections υλοποίησης Nginx

	Round Robin	**Least Connections**	**Round Robin by nginx**	**Least Connections by nginx**
Average Response time (ms)	3542,005	2083,22	3701,031	2054,45
Standard Deviation (ms)	5143,807	2074,965	4163,529	1930,191

Πίνακας 4.2 Σύγκριση αποτελεσμάτων των δύο υλοποιήσεων για round robin και least connections

Από τα διαγράμματα και τα αποτελέσματα μπορούν να γίνουν και εδώ οι παρατηρήσεις που έγιναν μελετώντας τα αποτελέσματα των μετρήσεων της απόδοσης του load balancer που υλοποιήθηκε για το πείραμα.

- Τα αποτελέσματα και εδώ είναι αναμενόμενα για τις δύο τεχνικές. Ο απλός αλγόριθμος round robin είναι φανερά πιο αργός από την πιο έξυπνη τεχνική least connections.

- Οι ώρες των πειραμάτων είναι βεβαίως οι ίδιες για να είναι πιο σωστή η σύγκριση ανάμεσα στα εργαλεία. Και εδώ παρατηρείται το φαινόμενο κατά το οποίο οι χρόνοι απόκρισης είναι ελαφρώς πιο μικροί στη μέση των διαγραμμάτων δείχνοντας πως εκείνες τις ώρες η υποδομή του ωκεανού είναι λιγότερο φορτωμένη αφού ο αριθμός των χρηστών φτάνει το χαμηλότερο σημείο.

Επιπλέον μπορούμε να παρατηρήσουμε την αστάθεια στα αποτελέσματα για τον αλγόριθμο round robin. Παρατηρούμε πως τα αποτελέσματα είναι μεν ελαφρώς διαφορετικά στην περίπτωση που η μέτρηση έγινε με τον nginx, όμως ο κύριος λόγος που συμβαίνει αυτό δεν είναι η διαφορά στην υλοποίηση των δύο αλγορίθμων. Ο κύριος λόγος για τον οποίο παρατηρείται αυτή η αστάθεια είναι ο ίδιος ο αλγόριθμος και η φύση του πειράματος. Λόγω του ότι ο round robin δεν παίρνει αποφάσεις με βάση δυναμικές παραμέτρους, όπως πχ ο χρόνος απόκρισης, τα αποτελέσματα είναι πιο επιρρεπή στο πόσο φορτωμένη είναι η υποδομή του ωκεανού και στο κάθε πότε τρέχουν τα scripts που «στρεσάρουν» τα εικονικά μηχανήματα. Τέτοιας κλίμακας διαφορές παρατηρούνται δοκιμάζοντας και συγκρίνοντας αποτελέσματα μεταξύ της ίδιας υλοποίησης του αλγορίθμου είτε με nginx είτε με τη δικιά μας.

Τέλος παρατηρείται στην αρχή του διαγράμματος υστέρηση στην απόδοση του αλγορίθμου. Εφ’ όσον οι συνθήκες του πειράματος δεν άλλαξαν με κάποιον τρόπο εκείνη τη στιγμή, το συμπέρασμα είναι πως μέχρι περίπου τη 1.30 το βράδυ η υποδομή του ωκεανού ήταν πιο φορτωμένη απ’ ότι αργότερα.

Αντίθετα, τα αποτελέσματα για τον αλγόριθμο least connections δείχνουν δύο σημαντικά πράγματα. Πρώτον ότι η υλοποίηση που κάναμε συμβαδίζει απόλυτα με τη δοκιμασμένη υλοποίηση του nginx και δεύτερον πως ο αλγόριθμος least connections είναι πράγματι πολύ σταθερός και γρήγορος αφού οι διακυμάνσεις του είναι πολύ μικρότερες κατά απόλυτη τιμή.

5. ΕΠΙΛΟΓΟΣ

5.1 Σύνοψη

Η υπολογιστική νέφους κερδίζει συνεχώς έδαφος τα τελευταία χρόνια και γίνεται απαραίτητο εργαλείο για μικρές και μεγάλες εταιρείες, όχι μόνο στον τομέα της πληροφορικής αλλά σε πολύ περισσότερους. Κρίνεται λοιπόν απαραίτητη η ταχεία βελτίωση του cloud computing και αυτό μπορεί να γίνει βελτιώνοντας οτιδήποτε από ένα μεγάλο σύνολο πραγμάτων είτε σε υποδομές είτε σε πρωτόκολλα που χρησιμοποιούνται σε εφαρμογές του cloud computing.

Ένας τομέας πάνω στον οποίο γίνονται συνεχώς έρευνες με αποτέλεσμα τη σταδιακή βελτίωση είναι η εξισορρόπηση φορτίου. Είναι μία από τις πιο σημαντικές λειτουργίες για μια υποδομή υπολογιστικού νέφους, αφού ο αποδοτικός διαμοιρασμός του φόρτου στους διαθέσιμους πόρους είναι ένας από τους βασικούς πυλώνες της υπολογιστικής νέφους και το κλειδί για την επιτυχία της.

Στο πλαίσιο της μεταπτυχιακής διατριβής μελετήθηκε το πεδίο της εξισορρόπησης φορτίου ως προς τις μεθόδους και τους αλγορίθμους που χρησιμοποιούνται. Δημιουργήθηκε μια υποδομή επάνω στην πλατφόρμα Okeanos και υλοποιήθηκαν κάποιοι γνωστοί αλγόριθμοι για εξισορρόπηση φορτίου. Αυτοί είναι οι Round Robin, Least Connections, Least Latency. Έγιναν αρκετά πειράματα πάνω σε αυτούς τους αλγορίθμους ώστε να εξαχθούν χρήσιμα συμπεράσματα για αυτούς αλλά και να αξιολογηθεί η πειραματική διάταξη.

Στη συνέχεια, αφού εντοπίστηκαν κάποια πιθανά σημεία βελτίωσης αυτών των αλγορίθμων, υλοποιήθηκαν 2 παραλλαγές του αλγορίθμου least latency οι οποίες μετρήθηκαν και αξιολογήθηκαν ακριβώς στις ίδιες συνθήκες. Στη συνέχεια δημιουργήθηκε μία ακόμα τεχνική για load balancing η οποία προσπαθεί να εξισορροπήσει το χρόνο που αφιερώνουν όλα τα μηχανήματα στην επεξεργασία αιτημάτων των πελατών.

Τέλος, έγιναν κάποιες μετρήσεις στους αλγορίθμους round robin και least connections χρησιμοποιώντας τον nginx, ο οποίος έχει υλοποιημένους και παρέχει σαν επιλογή αυτούς

τους αλγορίθμους για εξισορρόπηση φορτίου. Έτσι έγινε μια σύγκριση του nginx με τις υλοποιήσεις που έγιναν στα πλαίσια της παρούσας διπλωματικής.

Μια συνολική αποτίμηση είναι ότι εξάχθηκαν πολλά και σημαντικά συμπεράσματα μέσα από τη μελέτη αυτών που προαναφέρθηκαν. Χρησιμοποιήθηκαν και δοκιμάστηκαν αρκετά εργαλεία και μέθοδοι για να καταλήξουμε στα τελικά που θα χρησιμοποιηθούν έχοντας υπ' όψη πλεονεκτήματα και μειονεκτήματα. Η υλοποίηση εναλλακτικών και νέων τεχνικών δεν έδωσε κάποια μέθοδο που υπερτερεί έναντι των ήδη υπαρχόντων τεχνικών, μπορεί όμως να δώσει την κατεύθυνση και να αποτελέσει τη βάση για περαιτέρω μελέτη.

5.2 Μελλοντική εργασία

Η παρούσα μεταπτυχιακή διατριβή θα μπορούσε να αποτελέσει τη βάση για ένα σύνολο εργασιών ή εφαρμογών σε σχέση με την υλοποίηση τεχνικών εξισορρόπησης φορτίου σε υποδομές υπολογιστικού νέφους.

- Χρησιμοποιώντας την παρούσα υποδομή μπορούν να μελετηθούν τεχνικές για την ακριβή μέτρηση του latency ανάμεσα σε δύο εικονικά μηχανήματα. Η μη ακριβής μέτρηση του latency ήταν το σημαντικότερο μειονέκτημα τριών από τις μεθόδους που μελετήθηκαν παραπάνω.

- Η τελευταία μέθοδος η οποία υλοποιήθηκε είναι βέβαιο πως επιδέχεται αρκετές βελτιώσεις, αφού παρ' όλο που πατάει σε μια σωστή ιδέα δεν κατάφερε να είναι ανταγωνιστική με τις άλλες μεθόδους που εξετάστηκαν. Κάποιοι από τους λόγους είναι εμφανείς και άλλοι μπορούν να διερευνηθούν. Προσφέρεται για παραπάνω μετρήσεις και επιμέρους αναλύσεις για να εντοπιστεί ποιο είναι το bottleneck και να διορθωθεί.

6. ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Dialogic (2010). *Introduction to Cloud Computing White Paper,* [Διαθέσιμο: http://www.dialogic.com/~/media/products/docs/whitepapers/12023-cloud-computing-wp.pdf], [Προσπελάστηκε: 23η Σεπτ. 2016]

[2] N. Swarnkar (2013). A Survey of Load Balancing Techniques in Cloud Computing. *International Journal of Engineering Research & Technology*, 2(18), pp. 800-804.

[3] B. P. Rimal, E. Choi and I. Lumb, *A Taxonomy and Survey of Cloud Computing Systems*, Fifth International Joint Conference on INC, IMS and IDC, 2009. NCM '09., Seoul, 25-27 Αυγ. 2009, pp. 44-51.

[4] H. E. Chihoub, S. Ibrahim, G. Antoniu, M. Pérez (2013). *Consistency in the Cloud: When Money Does Matter!*. 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2013), 17 May, Delft, Netherlands.

[5] P. Mell, T. Grance (2011). *The NIST Definition of Cloud Computing*. NIST Special Publication 800-145.

[6] Z. Chaczko, V. Mahadevan, S. Aslanzadeh, C. Mcdermid (2011). *Availability and Load Balancing in Cloud Computing*. International Proceedings of Computer Science and Information Technology, International Conference on Computer and Software Modeling IPCSIT 2011, pp. 134 – 140.

[7] Z. Fang, B. Cao, and X. Jiang (2012). *A Research on Resource Allocation Solution with Cost Optimization in Cloud Computing System*. Proceedings on the International Conference on Internet Computing (ICOMP): 1-6. Athens: The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

[8] D. Thazhathethil, N. Katre, J. Mane-Deshmukh, Mahesh Kshirsagar, Prof. Anisaara Nadaph (2014). A Model for load balancing by Partitioning the Public Cloud,

International Journal of Innovative Research in Computer and Communication Engineering, 4(3), pp. 626-630.

[9] M. Randles, D. Lamb and A. Taleb-Bendiab, (2010) A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing, *2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA),* Perth, WA, pp. 551-556.

[10] K. Yelick, (2007). *Dynamic Load Balancing.* [Διαθέσιμο: https://www.dcc.fc.up.pt/~fds/aulas/PPD/0708/lect10-loadbalance-1x2.pdf], [Προσπελάστηκε: 23η Σεπτ. 2016]

[11] Y. Sahu, R. K. Pateriya (2013). Cloud Computing Overview with Load Balancing Technics. *International Journal of Computer Applications*, 65(24), pp. 40-44.

[12] K. A. Nuaimi, N. Mohamed, M. A. Nuaimi and J. Al-Jaroodi, (2012) *A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms,* Second Symposium on Network Cloud Computing and Applications (NCCA), London, 3-4 Dec., pp. 137-142.

[13] W. Wang and G. Casale, (2014). *Evaluating Weighted Round Robin Load Balancing for Cloud Web Services*, 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timisoara, 22-25 Σεπτ. 2014, pp. 393-400. doi: 10.1109/SYNASC.2014.59

[14] M. Shah, A. Kariyani, D. Agrawal (2008). Allocation Of Virtual Machines In Cloud Computing Using Load Balancing Algorithm. *IRACST - International Journal of Computer Science and Information Technology & Security (IJCSITS)*, 3(1), pp. 93-95.

[15] B. Yuce, M. S. Packianather, E. Mastrocinque, D. T. Pham and A. Lambiase (2013). Honey Bees Inspired Optimization Method: The Bees Algorithm. *Insects Open Access Journals*, 4(4) , pp. 646-662.

[16] M. S. Narasimha Murthy and V. Suma (2014). *A Study on Cloud Computing Testing Tools*. ICT and Critical Infrastructure: Proceedings of the 48th Annual

Convention of CSI - Volume I, Advances in Intelligent Systems and Computing, pp. 605-612.

[17] Sylor M. - EXFO Service Assurance (2013), *Testing the cloud White Paper*, [Διαθέσιμο: http://docplayer.net/4338176-Testing-the-cloud-mark-sylor-new-technologist-exfo-service-assurance.html], [Προσπελάστηκε: 24 Σεπτ. 2016]

[18] M. Yan, H. Sun, X. Liu, T. Deng, X. Wang (2014). Delivering Web service load testing as a service with a global cloud. *Concurrency and Computation: Practice and Experience*, 27(3), pp. 526-545.

[19] Crystal Bedell (2011). *Performance Testing in the Cloud*, [Διαθέσιμο: http://searchsoftwarequality.techtarget.com/feature/Performance-testing-in-the-cloud], [Προσπέλαστηκε: 25 Σεπτ. 2016]

[20] Google (2016), *Load Balancing and Scaling*, [Διαθέσιμο: https://cloud.google.com/compute/docs/load-balancing-and-autoscaling], [Προσπελάστηκε: 25 Σεπτ. 2016]

[21] F. Hu, M. Qiu, J. li, T. Grant, D. Tylor, S. McCaleb, L. Butler and R. Hamner (2011). *A Review on Cloud Computing: Design Challenges in Architecture and Security*. Journal of Computing and Information Technology, CIT 19, pp. 25-55.

[22] Laura Strassman (2015), *Performance Monitoring across the Development Lifecycle*, Διαθέσιμο: [https://www.cmcrossroads.com/article/performance-testing-considerations-public-cloud-environment], [Προσπελάστηκε: 25 Σεπτ. 2016]

[23] J. Ni, Y. Huang, Z. Luan, J. Zhang and D. Qian (2011), *Virtual machine mapping policy based on load balancing in private cloud environment*, International Conference on Cloud and Service Computing (CSC), Hong Kong, pp. 292-295.

[24] Ms. Nitika, Ms. Shaveta, Mr. Gaurav Raj (2012). Comparative Analysis of Load Balancing Algorithms in Cloud Computing. *International Journal of Engineering and Science*, 1(1), pp. 34-38.

[25] Software Testing Help (2016), *Top 15 Performance Testing Tools*, [Διαθέσιμο: http://www.softwaretestinghelp.com/performance-testing-tools-load-testing-tools/], [Προσπελάστηκε: 25 Σεπτ. 2016]