ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΝΟΡΧΗΣΤΡΩΣΗ ΕΙΚΟΝΙΚΩΝ ΠΟΡΩΝ ΜΕΣΩ ΤΗΣ ΠΛΑΤΦΟΡΜΑΣ KUBERNETES

Διπλωματική Εργασία

του

Φακριάδη Πασχάλη-Σάββα

Θεσσαλονίκη, Μάρτιος 2020

ΕΝΟΡΧΗΣΤΡΩΣΗ ΕΙΚΟΝΙΚΩΝ ΠΟΡΩΝ ΜΕΣΩ ΤΗΣ ΠΛΑΤΦΟΡΜΑΣ KUBERNETES

Φακριάδης Πασχάλης-Σάββας

Μηχανικός Η/Υ, Τηλεπικοινωνιών και Δικτύων, Τμήμα Ηλεκτρολόγων και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας, 2014

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Μαμάτας Ελευθέριος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 03/03/2020

Μαμάτας Ελευθέριος	Μαργαρίτης Κωνσταντίνος	Σακελλαρίου Ηλίας
.....	.....	.....

Φακριάδης Πασχάλης-Σάββας

.....

Περίληψη

Το υπολογιστικό νέφος κερδίζει μεγάλη δημοτικότητα την τελευταία δεκαετία καθώς όλο και περισσότερες νέες απαιτητικές εφαρμογές αναπτύσσονται σε αυτό. Συνεπώς, η αποδοτική διαχείριση των υπολογιστικών πόρων σε τέτοια περιβάλλοντα αποτελεί μία πρόκληση η οποία έχει λάβει μεγάλη προσοχή τόσο από τον βιομηχανικό όσο και από τον ακαδημαϊκό κόσμο. Μέσω των πλατφόρμων ενορχήστρωσης υπηρεσιών παρέχεται η δυνατότητα για αποδοτική διαχείριση των υπολογιστικών πόρων, με την εφαρμογή διαφόρων πολιτικών για κάθε μία τις εκάστοτε παρεχόμενες υπηρεσίες.

Ενα απο τα βασικά χαρακτηριστικά του υπολογιστικού νέφους είναι η ελαστικότητα, δηλαδή η ικανότητα του να προσαρμόζει δυναμικά το ποσό της κατανομής των πόρων για την κάλυψη των αλλαγών στις απαιτήσεις του φόρτου εργασίας. Η παρούσα διπλωματική εργασία επικεντρώνεται στη συγκριτική πειραματική ανάλυση εναλλακτικών επιλογών ενορχήστρωσης υπηρεσιών με στόχο την αποδοτική ελαστικότητα των παρεχόμενων υπηρεσιών, μέσω της πλατφόρμας ενορχήστρωσης Kubernetes, με σκοπό την αποδοτικότερη αξιοποίηση των υπολογιστικών και δικτυακών πόρων των διακομιστών για την βελτιστοποίηση της απόδοσης των παρεχόμενων υπηρεσιών σε περιβάλλοντα υπολογιστικού νέφους.

Το περιεχόμενο της παρούσας διπλωματικής εργασίας μελετάει το θεωρητικό υπόβαθρο α) της ελαφριάς εικονικοποίησης, των μικρό-υπηρεσιών και την ενορχήστωση αυτών μέσω των ειδικών πλατφορμών ενορχήστρωσης υπηρεσιών και β) την ελαστικότητας στο υπολογιστικό νέφος. Στη συνέχεια προχωράμε στο πειραματικό σκέλος της εργασίας με σκοπό την συγκριτική μελέτη μηχανισμών ελαστικότητας στο υπολογιστικό νέφος μέσω της πλατφόρμας Kubernetes. Πραγματοποιήσαμε διάφορα πειραματικά σενάρια αξιολογώντας την επίδοση των παρεχόμενων υπηρεσιών αναδεικνύωντας έτσι τα υπέρ και τα κατά της κάθε προσέγγισης.

Λέξεις Κλειδιά: Ελαστικότητα, Kubernetes, Υποδοχείς, Ενορχήστρωση Υποδοχέων

Abstract

Cloud computing has been gaining popularity over the last decade as more and more demanding applications are being developed to it. Therefore, efficient compute resource management in such environments is a challenge that has received attention from both the industrial and academic fields. Service orchestration platforms optimize resource management efficiently by implementing different policies for each service provided in a cloud environment.

One of the key features of cloud computing is elasticity, that is, its ability to dynamically adjust the amount of resource allocation to meet changes in workload requirements. This thesis focuses on a comparative experimental analysis of alternatives to orchestration services aimed at the efficient elasticity of the services provided, through the Kubernetes orchestration platform, in order to make the most efficient use of servers' computing and network resources targeting to application performance optimization.

The content of this thesis is to study the theoretical background of (a) lightweight virtualization, microservices and orchestration through special orchestration platforms and (b) elasticity in cloud computing. We then proceed to the experimental part of our work aiming to study the comparative elasticity mechanisms in cloud computing via the Kubernetes platform. We conducted various experimental scenarios evaluating the performance of the services provided, thereby highlighting the pros and cons of each approach.

Keywords: Elasticity, Kubernetes, Containers, Container orchestration

Πρόλογος – Ευχαριστίες

Θα ήθελα να ευχαριστήσω ιδιαίτερα τον συμφοιτητή και υποψήφιο Διδακτωρ Καλαφατίδη Σαράντη για την πολύτιμη συμβολή του για την επιτυχή ολοκλήρωση της παρούσας εργασίας και τον κ. Μαμάτα Ελευθέριο για την καθοδήγησή του για την επίτευξη του στόχου της πτυχιακής αυτής εργασίας.

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Γραφημάτων

1 Εισαγωγή

1.1 Πρόβλημα – Σημαντικότητα του θέματος

Λόγω της αύξησης των χρηστών του διαδικτύου αλλα και της ανάπτυξης εφαρμογών με σημαντικές απαιτήσεις, η αποδοτικότερη διαχείριση των πόρων επεξεργασίας στα κέντρα δεδομένων είναι ένα ζήτημα το οποίο χρήζει περισσότερης διερεύνησης [12]. Η διαχείριση των πόρων σε ένα περιβάλλον υπολογιστικού νέφους είναι ένα δύσκολο πρόβλημα, λόγω: της κλίμακας των σύγχρονων κέντρων δεδομένων, την ετερογένεια των τύπων των πόρων και τις αλληλεξαρτήσεις τους, την μεταβλητότητα και την μη προβλεψιμότητα του φορτίου καθώς και το φάσμα των στόχων των διαφόρων παραγόντων ενός περιβάλλοντος υπολογιστικού νέφους. Κατά συνέπεια, και οι ακαδημαϊκή κοινότητα αλλά και η βιομηχανία ξεκίνησαν σημαντικές ερευνητικές προσπάθειες στον τομέα αυτό [13].

Με την ευρεία πρόοδο της τεχνολογίας εικονικοποίησης, η ελαφριά εικονικοποίηση έχει συγκεντρώσει σημαντική προσοχή. Οι υποδοχείς (containers) [14] ως ένα μέσο ελαφριάς εικονικοποίησης παρέχουν απομόνωση διεργασιών στο επίπεδο του λειτουργικού συστήματος αρκετά διαφοροποιημένη σε σχέση με τις γνωστές τεχνολογίες εικονικοποίησης (π.χ. εικονικές μηχανές). Συγκεκριμένα, οι υποδοχείς στον ίδιο κεντρικό υπολογιστή, μοιράζονται το ίδιο λειτουργικό σύστημα. Αυτή η μέθοδος αφαίρεσης μειώνει τα γενικά έξοδα της εικονικοποίησης, αυξάνοντας σημαντικά την πυκνότητα της ανάπτυξης ανά κεντρικό υπολογιστή. Εφαρμογές που τρέχουν σε υποδοχείς βοηθούν στην απομόνωση εργασιών του ενός από τον άλλο, τρέχουν λογισμικό με προσυσκευασμένο σύστημα αρχείων και επομένως, βελτιώνεται η φορητοτητα τους. Οι εφαρμογές αναπτύσσονται ως ένα σύνολο μικρο-υπηρεσιών που ενθυλακώνονται σε διαφορετικούς υποδοχείς με ανεξάρτητους κύκλους ζωής.

Επιπλέον, η αρχιτεκτονική μικρο-υπηρεσιών βελτιώνει την ευκινησία του συστήματος μέσω της αποσύνδεσης και της διευκόλυνσης της ανάπτυξης, την αναβάθμιση, τη συνεχή ανάπτυξη και την τροποποίηση του χρόνου εκτέλεσης. [1]

Αυτά τα οφέλη έχουν οδηγήσει σε σημαντική αύξηση της υιοθέτηση και τη δημοτικότητα αυτής της τεχνολογίας. Οι υποδοχείς χρησιμοποιούνται ευρέως σε κέντρα

δεδομένων για να αντιμετωπίσουν τις αυξανόμενες διαφορές στους φόρτους εργασίας που προέρχονται από σύγχρονες εφαρμογές όπως υπηρεσίες ιστού, μεγάλα δεδομένα και διαδίκτυο των πραγμάτων είτε σε ιδιωτικά (private) ή σε δημόσια (public) κέντρα δεδομένων.

Αυτό, με τη σειρά του, έχει οδηγήσει στην εμφάνιση πλατφορμών ενορχήστρωσης των υποδοχέων. Σχεδιασμένα για τη διαχείριση της ανάπτυξης των εφαρμογών με υποδοχείς σε μεγάλης κλίμακας συστοιχίες (clusters), αυτά τα συστήματα είναι ικανά να εκτελούν εκατοντάδες χιλιάδες εργασίες σε χιλιάδες μηχανές [2].

Η ελαστικότητα [3] ορίζεται ως η ικανότητα ενός συστήματος να προσθέτει και να αφαιρεί υπολογιστικούς πόρους (όπως πυρήνες CPU, μνήμη, VM και στιγμιότυπα υποδοχέων) "on the fly" για να προσαρμοστούν στην μεταβολή του φορτίου σε πραγματικό χρόνο. Η ελαστικότητα είναι μια δυναμική ιδιότητα για το υπολογιστικό νέφος. Υπάρχουν δύο τύποι ελαστικότητας οριζόντια και κάθετη. Η οριζόντια ελαστικότητα συνίσταται στην προσθήκη ή την αφαίρεση περιπτώσεων υπολογιστικών πόρων που σχετίζονται με μια εφαρμογή. Η κάθετη ελαστικότητα συνίσταται στην αύξηση ή μείωση των στιγμιοτύπων των υπολογιστικών πόρων, όπως χρόνος CPU, πυρήνες, μνήμη και εύρος ζώνης δικτύου [3].

Μια πολύ ενδιαφέρουσα πτυχή όλων αυτών είναι ο συνδυασμός των πλεονεκτημάτων της εικονικοποίησης στα πλαίσια των πλατφορμών ενορχήστωσης των υποδοχέων με το ιδιαίτερα ενδιαφέρον κομμάτι της ελαστικότητας.

Ένα σημείο κλειδί των μηχανισμών ελαστικότητας και αυτό που παρατηρήσαμε στα πλαίσια της εργασίας αυτής είναι ότι ο οριζόντιος μηχανισμός ελαστικότητας αποδίδει καλύτερα σε αρχικά στάδια χρονικά και στη συνέχεια μετά την πάροδο ενός χρονικού διαστήματος αποδίδει καλύτερα ο κατακόρυφος μηχανισμός ελαστικότητας, ο οποίος αυξομειώνει τον χώρο που μπορεί η εφαρμογή να χρησιμοποιεί από τους πόρους του συστήματος με βάση τον αριθμό των αιτημάτων και του υπολογιστικού φόρτου που έχει παρατηρήσει στο προαναφερθέν αρχικό διάστημα.

1.2 Σκοπός – Στόχοι

Η παρούσα διπλωματική εργασία επικεντρώνεται στη συγκριτική πειραματική ανάλυση εναλλακτικών επιλογών ενορχήστρωσης, σε διαφορετικά πειραματικά σενάρια με σκοπό την αποδοτικότερη αξιοποίηση των υπολογιστικών και δικτυακών πόρων των διακομιστών σε κέντρα δεδομένων αλλά και την βελτιστοποίηση της απόδοσης των παρεχόμενων υπηρεσιών. Πιο συγκεκριμένα θέλουμε να δούμε πως ανταποκρίνονται οι διάφοροι μηχανισμοί ελαστικότητας σε ένα υπολογιστικό περιβάλλον και να αναδειχθούν με αυτόν τον τρόπο τα υπέρ και τα κατά του κάθε μηχανισμού.

1.3 Ερωτήματα – Υποθέσεις

Η συγκεκριμένη εργασία καλείται να απαντήσει στα ακόλουθα ερωτήματα:

1. Τι σημαίνει ελαφριά εικονικοποίηση, ποιά τα οφέλη της;
2. Ποιά η διάρθρωση των πλατφορμών ενορχήστρωσης των υποδοχέων, ποιές λειτουργίες συντελούνται σε αυτές;
3. Τι σημαίνει ελαστικότητα στο υπολογιστικό νέφος, πως κατηγοριοποιείται, ποιά τα βασικά της χαρακτηριστικά;
4. Ποιό το πειραματικό αντίκτυπο στο περιβάλλον μιας πλατφόρμας ενορχήστρωσης των υποδοχέων των διάφορων μηχανισμών ελαστικότητας;
5. Ποιός μηχανισμός αποδίδει καλύτερα σε συγκεκριμένη πλατφόρμα ενορχήστρωσης των υποδοχέων και πότε;

1.4 Συνεισφορά

Η συνεισφορά της παρούσας εργασίας έγκειται στα παρακάτω:

1. Παρουσίαση των εννοιών της εικονικοποίησης, των πλατφορμών ενορχήστρωσης των υποδοχέων και της ελαστικότητας.
2. Ανάλυση των λειτουργιών της βασικής δομής των πλατφορμών ενορχήστρωσης των υποδοχέων.
3. Ανάλυση του όρου ελαστικότητα και της κατηγοριοποίησης της στο υπολογιστικό νέφος.

4. Συγκριτική πειραματική μελέτη των διαφόρων μηχανισμών ελαστικότητας σε συγκεκριμένη πλατφόρμα ενορχήστρωσης.
5. Συμπεράσματα για την απόδοση των μηχανισμών που εξετάστηκαν και μελλοντικές προτάσεις – επεκτάσεις.

1.5 Διάρθρωση της μελέτης

Στο κεφάλαιο που ακολουθεί παρουσιάζεται το θεωρητικό υπόβαθρο της εργασίας μας, η εικονικοποίηση, οι πλατφόρμες ενορχήστρωσης των υποδοχέων και η δομή τους, η έννοια της ελαστικότητας στο υπολογιστικό νέφος και η κατηγοριοποίησή της. Στη συνέχεια στο 3ο κεφάλαιο αναλύουμε τα εργαλεία των πειραμάτων μας. Στο 4ο κεφάλαιο παρουσιάζουμε την πειραματική προσέγγιση στο θέμα που καταπιάνεται η εργασία και καταλήγουμε σε συμπεράσματα. Στο 5ο και τελευταίο κεφάλαιο, αναδεικνύουμε τα συμπεράσματα της γενικότερης μελέτης μας προβάλουμε τα όρια και τους περιορισμούς της έρευνας μας και αναφερόμαστε σε μελλοντικές επεκτάσεις της παρούσας εργασίας.

2 Θεωρητικό Υπόβαθρο

Στο συγκεκριμένο κεφάλαιο θα επικεντρωθούμε στην ανάλυση των παρακάτω ερωτημάτων και των απαντήσεων που προκύπτουν: α) τι είναι οι μικρο-υπηρεσίες, υποδοχείς (microservices – containers), β) τι είναι τα συστήματα ενορχήστρωσης υποδοχέων (service management orchestration systems) [2], ποιά τα βασικά τους χαρακτηριστικά, γ) τι είναι η ελαστικότητα (τι σημαίνει ο όρος ελαστικότητα) και που μας χρησιμεύει, πως κατηγοριοποιείται, ποιοι βασικοί μηχανισμοί ελαστικότητας υπάρχουν και που θα επικεντρωθούμε;

Τα συγκεκριμένα θέματα που θα αναπτυχθούν στο παρόν κεφάλαιο θα αποτελέσουν τη θεωρητική βάση για το υπόλοιπο της εργασίας.

2.1 Μικρουπηρεσίες (Microservices)

Οι μικρουπηρεσίες [15] είναι μια τεχνική ανάπτυξης λογισμικού - μια παραλλαγή του αρχιτεκτονικού στυλ αρχιτεκτονικής προσανατολισμένης στις υπηρεσίες (SOA) - που δομεί μια εφαρμογή ως μια συλλογή χαλαρά συνδεδεμένων υπηρεσιών. Σε μια αρχιτεκτονική μικρουπηρεσιών, οι υπηρεσίες είναι λεπτομερείς (fine-grained) και τα πρωτόκολλα απλοποιημένα (lightweight). Το πλεονέκτημα της διάσπασης μιας εφαρμογής σε διαφορετικές μικρότερες υπηρεσίες είναι ότι βελτιώνει τη διαμόρφωση (modularity). Αυτό καθιστά την εφαρμογή πιο εύκολη στην κατανόηση, την ανάπτυξη, τη δοκιμή και την αντοχή στη διάβρωση της αρχιτεκτονικής. Η ανάπτυξη μπορεί να πραγματοποιηθεί παράλληλα επιτρέποντας σε μικρές αυτόνομες ομάδες να αναπτύσσουν, να κάνουν deploy και να κλιμακώνουν ανεξάρτητα τις αντίστοιχες υπηρεσίες. Επιτρέπει, επίσης την εμφάνιση της αρχιτεκτονικής μιας μεμονωμένης υπηρεσίας μέσω συνεχών αναδομήσεων (refactoring). Οι αρχιτεκτονικές που βασίζονται σε μικρουπηρεσίες επιτρέπουν τη συνεχή παράδοση και ανάπτυξη [4].

Πού τοποθετούμε τις μικρουπηρεσίες; Σε υποδοχείς. Οι υποδοχείς είναι πακέτα λογισμικού που περιλαμβάνουν όλα όσα χρειάζεται για να τρέξει, όπως κώδικας, εξαρτήσεις, βιβλιοθήκες, δυαδικά αρχεία και πολλά άλλα. Το Docker είναι μια από τις

δημοφιλέστερες επιλογές για την κατασκευή και τη λειτουργία υποδοχέων, και το Kubernetes γίνεται γρήγορα το πρότυπο defacto που χρησιμοποιείται για την ενορχήστρωση πολλών υποδοχέων σε περιβάλλοντα επιχειρήσεων. Σε σύγκριση με τα εικονικά μηχανήματα, οι υποδοχείς μοιράζονται τον πυρήνα του λειτουργικού συστήματος αντί να έχουν ένα πλήρες αντίγραφο του - όπως κάνουν πολλαπλά VM σε έναν υπολογιστή. Παρόλο που είναι δυνατό να τοποθετηθουν οι μικρουπηρεσίες σε πολλαπλά VM, χρησιμοποιούνται συνήθως οι υποδοχείς σε αυτή την περίπτωση, δεδομένου ότι καταλαμβάνουν λιγότερο χώρο και εκκινούνται πιο γρήγορα[5].

2.2 Ελαφριά εικονικοποίηση - Υποδοχείς

Με την ευρεία πρόοδο της τεχνολογίας εικονικοποίησης ή ελαφριάς εικονικοποίησης [16] (lightweight virtualization) η τεχνολογία των υποδοχέων έχει συγκεντρώσει σημαντική προσοχή στον τομέα της έρευνας και της βιομηχανίας.

Η παραδοσιακή τεχνολογία εικονικοποίησης (virtualization) που βασίζεται σε επόπτες (hypervisors) αποτελείται από ένα επίπεδο αφαίρεσης υλικού που χρησιμοποιεί εικονικές μηχανές (virtual machines, VMs), η κάθε μία εκ των οποίων εκτελούν ένα πλήρες λειτουργικό σύστημα πάνω από το στρώμα ενός επόπτη. Ενώ αυτή η μέθοδος αφαίρεσης προσφέρει ανεξαρτησία υλικού, έχει σημαντικό αντίκτυπο στην απόδοση του συστήματος.

Αντίθετα, οι υποδοχείς παρέχουν απομόνωση των διεργασιών στο επίπεδο του λειτουργικού συστήματος. Συγκεκριμένοι υποδοχείς στον ίδιο κεντρικό υπολογιστή, μοιράζονται το ίδιο λειτουργικό σύστημα. Αυτή η μέθοδος αφαίρεσης επίσης ονομάζεται εικονικοποίηση λειτουργικού συστήματος, μειώνει τα γενικά έξοδα της εικονικοποίησης, αυξάνοντας σημαντικά την πυκνότητα της ανάπτυξης ανά κεντρικό υπολογιστή. Εφαρμογές που τρέχουν σε υποδοχείς βοηθούν στην απομόνωση εργασιών του ενός από τον άλλο, τρέχουν λογισμικό με προσυσκευασμένο σύστημα αρχείων και επομένως, βελτιώνεται η φορητότητά τους. Οι εφαρμογές αναπτύσσονται ως ένα σύνολο μικρο-υπηρεσιών που ενθυλακώνονται σε διαφορετικούς υποδοχείς με ανεξάρτητους κύκλους ζωής.

Για να αποφευχθεί η υπερπροσφορά και να βελτιωθεί η χρήση των πόρων, ο προγραμματιστής μπορεί να καθυστερήσει κάποια καθήκοντα χαμηλότερης

προτεραιότητας και να δώσει την ευκαιρία σε άλλες εργασίες πραγματικού χρόνου με αυστηρές προθεσμίες. Επιπλέον, η αρχιτεκτονική μικρο-υπηρεσιών βελτιώνει την ευελιξία του συστήματος μέσω της αποσύνδεσης και της διευκόλυνσης της ανάπτυξης, την αναβάθμιση, τη συνεχή ανάπτυξη και την τροποποίηση του χρόνου εκτέλεσης.

Οι υποδοχείς ξεκινούν πολύ γρήγορα, γεγονός που αποτελεί πλεονέκτημα για την ελαστικότητα των εφαρμογών του νέφους(cloud applications) να αυξηθεί ή να συρρικνωθεί(προσθήκη ή αφαίρεση πόρων) σύμφωνα με τις αλλαγές του φόρτου εργασίας. Οι υποδοχείς ή οι ελαφριές τεχνολογίες εικονικοποίησης εξελίσσονται γρήγορα και αντικαθιστούν τις παραδοσιακές εικονικές μηχανές. Αρκετές λύσεις υποδοχέων υπάρχουν, π.χ. OpenVZ, Linux-VServer, LXC, Docker και Rocket [1].

2.3 Συστήματα ενορχήστρωσης συστοιχίας (cluster) με βάση τους υποδοχείς

Οι υποδοχείς χρησιμοποιούνται ευρέως από οργανώσεις για να υποστηρίξουν τις αυξανόμενες αλλαγές στις απαιτήσεις φόρτου εργασίας που προέρχονται από σύγχρονες εφαρμογές όπως υπηρεσίες ιστού, μεγάλα δεδομένα και διαδίκτυο των πραγμάτων είτε σε ιδιωτικά (private) ή σε δημόσια (public) κέντρα δεδομένων. Αυτό, με τη σειρά του, έχει οδηγήσει στην εμφάνιση πλατφορμών ενορχήστρωσης των υποδοχέων. Σχεδιασμένα για τη διαχείριση της ανάπτυξης των εφαρμογών των υποδοχέων (containerized) σε μεγάλης κλίμακας συστοιχίες (clusters), αυτά τα συστήματα είναι ικανά να εκτελούν εκατοντάδες χιλιάδες εργασίες σε χιλιάδες μηχανές.

Αυτά τα συστήματα ενορχήστρωσης σχεδιάζονται συνήθως για τον προγραμματισμό του φόρτου εργασίας των εφαρμογών των υποδοχέων ενός ή περισσοτέρων τύπων. Κάθε τύπος εφαρμογής έχει τα δικά του χαρακτηριστικά και τις απαιτήσεις, όπως εργασίες μεγάλης διαθεσιμότητας, περιορισμένες εντός χρονικού ορίου εργασίες ή εργασίες ευαίσθητες στην καθυστέρηση για παράδειγμα. Η πλειονότητα των συστημάτων υποστηρίζουν την πολυχρησία, για παράδειγμα προγραμματίζουν τις εφαρμογές που ανήκουν σε πολλούς χρήστες χρησιμοποιώντας ένα κοινό σύνολο υπολογιστικών πόρων, επιτρέποντας την καλύτερη χρησιμοποίηση των πόρων.

Ως εκ τούτου, καθώς οι εφαρμογές υποβάλλονται για ανάπτυξη, το σύστημα ενορχήστρωσης πρέπει να τις τοποθετεί όσο το δυνατόν πιο γρήγορα σε έναν από τους

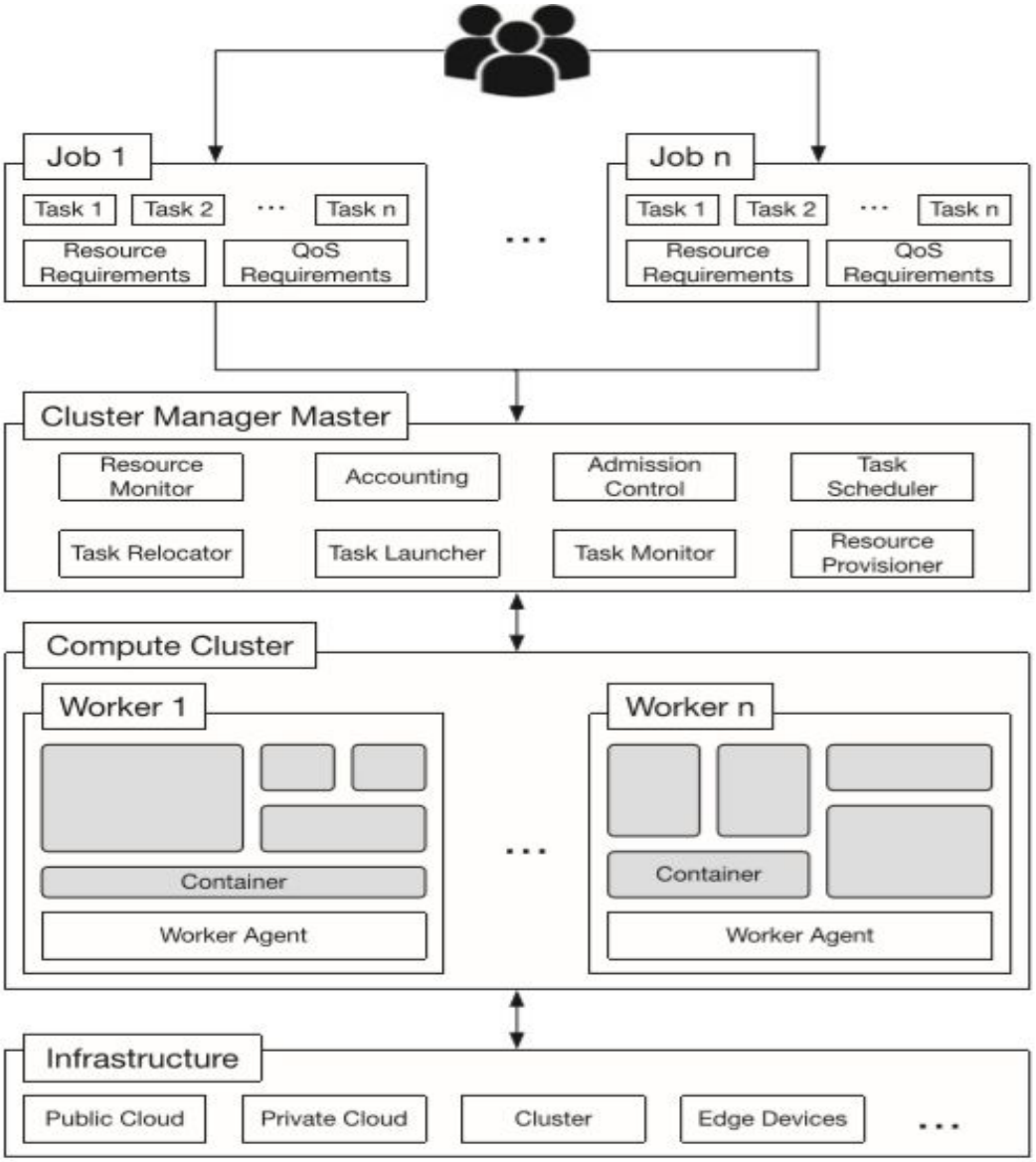
διαθέσιμους πόρους, λαμβάνοντας υπόψη τους ιδιαίτερους περιορισμούς και μεγιστοποιώντας την χρήση των υπολογιστικών πόρων μειώνοντας το λειτουργικό κόστος του οργανισμού. Αυτά τα συστήματα πρέπει επίσης να το επιτυγχάνουν αυτό ενώ ταυτόχρονα χειρίζονται έναν σημαντικό αριθμό υπολογιστικών πόρων, παρέχοντας ανοχή στα σφάλματα, υψηλή διαθεσιμότητα και δίκαιη κατανομή των πόρων.

Συνοπτικά, για να επιτευχθεί ο στόχος τους, τα εργαλεία ενορχήστρωσης των υποδοχέων πρέπει να διαχειρίζονται αποτελεσματικά ένα ευρύ φάσμα εφαρμογών των υποδοχέων και των κατανεμημένων πόρων που υποστηρίζουν την εκτέλεσή τους. Αυτό είναι ένα δύσκολο πρόβλημα που περιέχει αρκετά ζητήματα που πρέπει να αντιμετωπιστούν, όπως η κλιμάκωση σε μεγάλο αριθμό μηχανών, η μεγιστοποίηση της απόδοσης της εφαρμογής, ελαχιστοποιώντας την καθυστέρηση ανάπτυξης εφαρμογών, μεγιστοποιώντας τη χρήση των πόρων, καλύπτοντας τις συγκεκριμένες απαιτήσεις και τους περιορισμούς των διαφορετικών εφαρμογών, παρέχοντας ανοχή σφάλματος και υψηλή διαθεσιμότητα, υποστηρίζοντας διαφορετικούς τύπους εφαρμογών και δίκαιη κατανομή πόρων, μεταξύ άλλων. Στη συνέχεια, στοχεύουμε να μελετήσουμε πως τα διαφορετικά συστήματα ενορχήστρωσης των υποδοχέων επιτυγχάνουν αυτές τις απαιτήσεις καθώς και τις διαφορετικές δυνατότητες που προσφέρουν. Στο πεδίο του υπολογιστικού νέφους (cloud computing), ένα παρόμοιο πρόβλημα έχει ήδη αντιμετωπιστεί από τους διαχειριστές των πόρων των VM (RMs), οι οποίοι είναι υπεύθυνοι για την κατανομή των υπολογιστικών, των αποθηκευτικών καθώς και των δικτυακών πόρων σε εφαρμογές εντός ενός κέντρου δεδομένων.

2.3.1 Αρχιτεκτονική για τα συστήματα ενορχήστρωσης των containers

Τα συστήματα ενορχήστρωσης των υποδοχέων επιτρέπουν την ανάπτυξη των εφαρμογών των υποδοχέων σε κοινόχρηστη συστοιχία (shared cluster). Ενεργοποιούν την εκτέλεσή τους και την παρακολούθησή τους με τη διαφανή διαχείριση των καθηκόντων (tasks) και των δεδομένων που αναπτύσσονται σε ένα σύνολο κατανεμημένων πόρων. Τα συστατικά (components) που παρουσιάζονται είναι κοινά στην πλειονότητα των συστημάτων ενορχήστρωσης των υποδοχέων. Ωστόσο, δεν είναι απαραίτητο να εφαρμοστούν όλα τα συστατικά αυτά για να έχουμε ένα πλήρως λειτουργικό σύστημα. Τέσσερις κύριες οντότητες ή επίπεδα προσδιορίζονται στην

παρουσιαζόμενη αρχιτεκτονική, συγκεκριμένα, μία ή περισσότερες εργασίες, ένας διαχειριστής συστοιχίας (cluster manager master), μία υπολογιστική συστοιχία (compute cluster) και η φυσική υποδομή. Από μια προοπτική υψηλού επιπέδου, οι χρήστες υποβάλλουν εργασίες που αποτελούνται από ένα ή περισσότερα κομμάτια (tasks) στο μάνατζερ διαχειριστή της συστοιχίας (cluster manager master). Αυτή η οντότητα τότε αναθέτει τις υποβληθείσες εργασίες (tasks) στους κόμβους που εκτελούν τις εργασίες (worker nodes) στην υπολογιστική συστοιχία (compute cluster), όπου αυτές εκτελούνται. Η υπολογιστική συστοιχία είναι μια αφαίρεση των διασυνδεδεμένων κόμβων που μπορούν να είναι είτε φυσικά μηχανήματα είτε VMs σε διάφορες υποδομές, όπως νέφος (cloud) ή ιδιωτικές συστοιχίες (private clusters). Μια λεπτομερής εξήγηση που φαίνεται και στην παρακάτω εικόνα για κάθε επίπεδο και τις ευθύνες του παρουσιάζεται ως εξής.



Εικόνα 2 - Αρχιτεκτονική συστημάτων ενορχήστρωσης υποδοχέων[2]

Εργασίες (Jobs)

Οι χρήστες υποβάλλουν τις αιτήσεις τους με τη μορφή εργασιών. Αυτές οι εργασίες συνήθως ανήκουν σε διαφορετικούς χρήστες και είναι ετερογενείς, μπορεί να κυμαίνονται από μακροπρόθεσμες ευαίσθητες υπηρεσίες σε καθυστέρηση έως βραχυπρόθεσμες εργασίες με έντονη κατανάλωση πόρων. Μια εργασία αποτελείται από ένα ή περισσότερα μικρότερα κομμάτια (tasks). Τα κομμάτια αυτά είναι γενικά ομοιογενή και ανεξάρτητα, αλλά ορισμένες πλατφόρμες (frameworks) διευρύνουν αυτόν

τον ορισμό και επιτρέπουν στους χρήστες να ορίζουν εργασίες που περιλαμβάνουν αλληλεξαρτώμενα και ετερογενή κομμάτια. Οι χρήστες μπορούν επίσης να εκφράσουν τις απαιτήσεις σε πόρους των εργασιών από την άποψη της ποσότητας της CPU και της μνήμης που θα απαιτήσουν για παράδειγμα. Άλλες απαιτήσεις ποιότητας υπηρεσιών (QoS) όπως οι απαιτήσεις ανοχής σε σφάλματα, οι χρονικοί περιορισμοί, οι προτεραιότητες και οι κλάσεις QoS μπορούν να συμπεριληφθούν ως μέρος του ορισμού της εργασίας (job definition).

Cluster Manager Master

Ο διαχειριστής συστοιχίας (master component) είναι ο πυρήνας του συστήματος ενορχήστρωσης. Έχει μια μονάδα παρακολούθησης πόρων υπεύθυνη για τη διατήρηση της κατάστασης των μετρήσεων κατανάλωσης πόρων σε πραγματικό χρόνο για κάθε κόμβο εργάτη (worker node) στη συστοιχία (cluster). Αυτές οι πληροφορίες είναι συνήθως προσπελάσιμες από άλλες μονάδες στο σύστημα, όπως απαιτείται. Για παράδειγμα, ο προγραμματιστής εργασιών (task scheduler) και οι λειτουργικές μονάδες μετεγκατάστασης μπορούν να χρησιμοποιήσουν αυτά τα δεδομένα για να λάβουν καλύτερες αποφάσεις βελτιστοποίησης. Η μονάδα λογιστικής (accounting module) έχει μια παρόμοια λειτουργικότητα με την παρακολούθηση πόρων, αλλά εστιάζει στη συλλογή της πραγματικής χρήσης πόρων και μετρήσεων που αφορούν τον ιδιοκτήτη του συστήματος διαχείρισης συστοιχίας (cluster management system). Από τη μία πλευρά, οι μετρήσεις που σχετίζονται με την υποδομή που ενοποιούνται από αυτήν την μονάδα περιλαμβάνουν τη συνολική χρήση πόρων, την χρήση της ενέργειας και το κόστος αν αναπτυχθεί σε περιβάλλον νέφους (cloud environment). Από την άλλη πλευρά, οι μετρήσεις που σχετίζονται με το χρήστη ενδέχεται να περιλαμβάνουν τον αριθμό και τον τύπο των εργασιών που υποβάλλονται από τους χρήστες, καθώς και τον όγκο των πόρων που καταναλώνονται από αυτές τις εργασίες. Αυτές οι μετρήσεις μπορεί να βοηθήσουν στην επιβολή ποσοστώσεων στο χρήστη ή στην εκτίμηση ποσών χρέωσης για παράδειγμα.

Η μονάδα ελέγχου εισαγωγής (admission control module) είναι υπεύθυνη για τον καθορισμό του εάν (1) η ποσόστωση πόρων του χρήστη είναι ίση ή μεγαλύτερη από το ποσό των πόρων που ζητήθηκαν ή (2) υπάρχουν αρκετοί διαθέσιμοι πόροι στη συστοιχία

για την εκτέλεση των υποβληθέντων εργασιών. Για το τελευταίο σενάριο, μπορούν να παρθούν πολλές αποφάσεις σε περίπτωση που οι πόροι είναι ανεπαρκείς. Στη μία πλευρά του φάσματος, οι εργασίες θα μπορούσαν απλώς να απορριφθούν. Από την άλλη πλευρά, μια πιο σύνθετη λύση θα λαμβάνει υπόψη τις προτεραιότητες των χρηστών και τις κλάσεις QoS για τις εργασίες που θεωρούνται λιγότερο σημαντικές και να ελευθερώσει πόρους για τις εισερχόμενες εργασίες που θεωρούνται πιο σημαντικές. Μια άλλη πιθανή λύση θα ήταν το ενδεχόμενο αύξησης του αριθμού των κόμβων της συστοιχίας για την τοποθέτηση των εισερχομένων εργασιών.

Ο χρονοπρογραμματιστής εργασιών (task scheduler) αντιστοιχίζει τις εργασίες, ή πιο συγκεκριμένα τα κομμάτια των εργασιών (tasks), στους πόρους της συστοιχίας. Αυτό γίνεται συνήθως με την εξέταση πολλών παραγόντων και αντιτιθέμενων στόχων. Πρώτον, πρέπει να λαμβάνονται υπόψη οι απαιτήσεις και οι διαθέσιμες δυνατότητες των πόρων. Δεύτερον, τα συστήματα διαχείρισης συστοιχίας ασχολούνται με την αποτελεσματική χρήση των πόρων και ως εκ τούτου η μεγιστοποίηση της αξιοποίησης των κόμβων της συστοιχίας είναι συνήθως στόχος του χρονοπρογραμματιστή εργασιών. Τέλος, η αντιστοίχιση των εργασιών (tasks) έτσι ώστε πρόσθετες απαιτήσεις QoS των εργασιών με όρους προτεραιοτήτων ή περιορισμών να ικανοποιούνται είναι μια άλλη βασική ευθύνη του χρονοπρογραμματιστή.

Ο μετατοπιστής εργασιών (task relocator) μπορεί να θεωρηθεί ως πρόγραμμα αλλαγής προγράμματος (rescheduler). Όποτε οι εργασίες πρέπει να μεταφερθούν είτε επειδή έχουν προληφθεί ή για σκοπούς εξυγίανσης για παράδειγμα, αυτό το στοιχείο (component) είναι υπεύθυνο για τον προσδιορισμό της τύχης τους. Μια πολιτική μετεγκατάστασης εργασιών μπορεί απλά να επιλέξει να απορρίψει την εργασία ή να την επαναφέρει στην ουρά χρονοπρογραμματισμού. Πιο εξελιγμένες προσεγγίσεις μπορούν να αναλύσουν την κατάσταση του συστήματος και να καθορίσουν μια νέα βέλτιστη θέση για την εργασία με στόχο τη βελτίωση της χρησιμοποίησης των πόρων.

Ο εκκινητής εργασιών (task launcher) είναι υπεύθυνος για την εκκίνηση των containers των εργασιών σε συγκεκριμένες μηχανές της συστοιχίας αφού η απόφαση αυτή έχει οριστικοποιηθεί από τον χρονοπρογραμματιστή (scheduler) ή τον μετατοπιστή (relocator). Επιπλέον, για την υποστήριξη της διαχείρισης των εργασιών που

εκτελούνται, ο υπεύθυνος παρακολούθησης εργασιών (task monitor) είναι υπεύθυνος για τον έλεγχο των εργασιών που εκτελούνται και την παρακολούθηση της κατανάλωσης πόρων και των μετρήσεων QoS. Αυτές οι πληροφορίες βοηθούν στην ανίχνευση αποτυχιών ή παραβιάσεων του QoS και δίνουν τη δυνατότητα στο σύστημα να παίρνει καλύτερες αποφάσεις χρονοπρογραμματισμού ή μετεγκατάστασης.

Τέλος, ο υπεύθυνος παροχής πόρων (resource provisioner) είναι υπεύθυνος για τη διαχείριση της προσθήκης νέων κόμβων στη συστοιχία. Αυτό μπορεί να είναι είτε μια χειροκίνητη ή μια αυτόματη διαδικασία. Σε μια χειροκίνητη διαδικασία, συνήθως οι διαχειριστές συστημάτων (system administrators) θα εκκινήσουν έναν νέο κόμβο με το λογισμικό πράκτορα εργάτη (worker agent software) να περιέχεται σε αυτόν και να εκτελέσει μια κλήση για να ενημερώσει για τον εαυτό του στον master. Αυτή η κλήση μπορεί να υποβληθεί σε επεξεργασία από τον υπεύθυνο παροχής πόρων (resource provisioner) έτσι ώστε ο νέος κόμβος να υπολογίζεται από τον master.

Ωστόσο, ένας προμηθευτής πόρων (resource provisioner) δεν είναι πάντοτε απαραίτητος σε μια τέτοια περίπτωση, καθώς ο κόμβος-εργάτης μπορεί να στείλει αυτόματα ένα σήμα στην οθόνη παρακολούθησης πόρων για να διαφημιστεί για παράδειγμα. Στην περίπτωση μιας αυτόματης διαδικασίας ωστόσο, ο υπεύθυνος παροχής πόρων (resource provisioner) αποτελεί βασικό στοιχείο της αρχιτεκτονικής καθώς θα είναι υπεύθυνος για τη δυναμική προσθήκη εικονικών κόμβων (π.χ. VMs) στη συστοιχία όταν οι υπάρχοντες πόροι είναι ανεπαρκείς για την ικανοποίηση των απαιτήσεων των εφαρμογών. Επίσης, θα αποφασίσει πότε οι κόμβοι δεν χρειάζονται πλέον στην συστοιχία και θα απενεργοποιήσει τους κόμβους για να αποφευχθεί η επιβάρυνση με επιπλέον κόστος.

Compute Cluster

Κάθε μηχανή στη συστοιχία που είναι διαθέσιμη για την ανάπτυξη εργασιών είναι ένας κόμβος-εργάτης. Κάθε ένας από αυτούς τους κόμβους έχει έναν πράκτορα-εργάτη με διάφορες ευθύνες. Κατ 'αρχάς, συλλέγει τοπικές πληροφορίες, όπως μετρήσεις κατανάλωσης πόρων που μπορούν να αναφέρονται περιοδικά στον master, και συγκεκριμένα στον ελεγκτή πόρων (resource monitor). Δεύτερον, ξεκινά και σταματά τις εργασίες και διαχειρίζεται τοπικά τους πόρους, συνήθως μέσω ενός εργαλείου

διαχειριστή υποδοχέων όπως υποδοχείς Docker ή Linux. Τέλος, οπτικοποιεί (monitors) τις εργασίες των υποδοχέων που αναπτύσσονται στον κόμβο, δηλαδή πληροφορίες, οι οποίες συνήθως βασίζονται στο στοιχείο παρακολούθησης εργασιών (task monitor component) του master.

Infrastructure

Ένα από τα βασικά πλεονεκτήματα των υποδοχέων είναι η ευελιξία τους να αναπτυχθούν σε πλήθος πλατφορμών. Εξαιτίας αυτών, οι μηχανές της συστοιχίας μπορούν να είναι είτε VMs σε δημόσιες ή ιδιωτικές υποδομές νέφους (cloud), φυσικές μηχανές σε μια συστοιχία ή ακόμα και κινητές (mobile) συσκευές ή συσκευές άκρων (edge) μεταξύ άλλων.

2.3.2 Συστήματα Ενορχήστρωσης των Υποδοχέων

Παρακάτω θα παρουσιαστούν ενδεικτικά κάποια συστήματα ενορχήστρωσης των υποδοχέων που είναι βασισμένα και δομημένα στις βασικές αρχές της αρχιτεκτονικής των συστημάτων των υποδοχέων που παρουσιάστηκαν παραπάνω.

Borg

Το σύστημα διαχείρισης συστοιχίας Borg της Google έχει σχεδιαστεί για να διαχειρίζεται εκατοντάδες χιλιάδες ετερογενών εργασιών σε όλη την Ευρώπη, κατά μήκος αρκετών συστοιχιών, με δεκάδες χιλιάδες μηχανές. Οι χρήστες υποβάλλουν εργασίες στο Borg, οι οποίες αποτελούνται από ένα ή περισσότερα ομοιογενή κομμάτια. Κάθε εργασία τρέχει σε ένα κελί, το οποίο είναι ένα σύνολο ετερογενών μηχανών που χαρακτηρίζονται ως μονάδα. Το φόρτο εργασίας στα κελιά του Borg αποτελείται από δύο τύπους εφαρμογών. Οι πρώτες είναι οι μακροχρόνιες υπηρεσίες που πρέπει να παραμείνουν διαθέσιμες όλες τις ώρες. Αυτές οι υπηρεσίες πρέπει να εξυπηρετούν αιτήματα βραχείας διάρκειας με ελάχιστη καθυστέρηση που αντιστοιχίζονται κυρίως στην αντιμετώπιση των τελικών χρηστών των εφαρμογών ιστού. Αυτές είναι συνήθως ταξινομημένες ως εργασίες υψηλής προτεραιότητας ή εργασίες παραγωγής. Ο δεύτερος τύπος φόρτου εργασίας αντιστοιχεί στις εργασίες παρτίδας (batch jobs). Αυτές μπορεί να διαρκέσουν από μερικά δευτερόλεπτα έως μέρες και συνήθως ταξινομούνται ως

χαμηλότερης προτεραιότητας εργασίες ή μη παραγωγικές εργασίες (non production jobs). Οι εργασίες εκτελούνται σε υποδοχείς που αναπτύσσονται σε φυσικά μηχανήματα και έχουν απαιτήσεις πόρων που καθορίζονται από την άποψη του χώρου στο δίσκο, της μνήμης RAM και των πυρήνων της CPU, μεταξύ άλλων πόρων. Η στοίβα του υποδοχέα που χρησιμοποιείται είναι ιδιόκτητη και βασίζεται σε cgroups του Linux. Υπόψιν του προγραμματιστή (scheduler), παρακολουθείται ασύγχρονα μια ουρά εκκρεμών εργασιών. Αυτή η ουρά αποτελείται με υψηλής έως χαμηλής προτεραιότητας σειρά από επιλεγμένες εργασίες που βασίζονται σε ένα κυκλικό (round-robin) σχήμα για κάθε προτεραιότητα. Ο αλγόριθμος προγραμματισμού έχει δύο μέρη, δηλαδή εύρεση εφικτών μηχανών που ταιριάζουν με τις απαιτήσεις της εργασίας και επιλογή μιας από αυτές τις μηχανές (βαθμολόγηση). Ο μηχανισμός βαθμολόγησης ευνοεί τις μηχανές που έχουν ήδη τα πακέτα των εργασιών, διαχέει κομμάτια (από την ίδια εργασία) στις περιοχές ισχύος και αποτυχίας και πακετάρει την ποιότητα, όπως την ανάμειξη εργασιών χαμηλής και υψηλής προτεραιότητας στο ίδιο μηχάνημα για να επιτρέψει σε αυτές με υψηλή προτεραιότητα να καταναλώνουν περισσότερους πόρους όταν απαιτείται.

Το Borg δεν βασίζεται αποκλειστικά στο ποσό των πόρων που ζητήθηκε για μια εργασία για να δεσμεύσει CPU και RAM για παράδειγμα. Αντι αυτου, εκτιμά το ποσό των πόρων που θα χρησιμοποιήσει μία εργασία και ανακτά το υπόλοιπο για εργασία που μπορεί να ανεχθεί χαμηλότερης ποιότητας πόρους όπως οι εργασίες παρτίδας (batch jobs). Αυτή η κράτηση αλλάζει δυναμικά με βάση την ακριβή κατανάλωση πόρων του έργου μετρημένη κάθε λίγα δευτερόλεπτα. Το Borg διαφοροποιεί μεταξύ συμπιεστικών πόρων όπως κύκλους CPU που μπορούν να αιτηθεί η απόδοσή τους από μια εργασία μειώνοντας το QoS της χωρίς να τερματιστεί η εργασία και μη συμπιεστούς πόρους όπως η μνήμη, που γενικά δεν μπορεί να αιτηθεί η χρήση της χωρίς να τερματιστεί η εργασία. Ως εκ τούτου, οι εργασίες που προσπαθούν να καταναλώσουν περισσότερη ram τερματίζονται, ενώ η CPU αποδίδεται-κατακερματίζεται στο ζητούμενο ποσό.

Το Borg χρησιμοποιεί διάφορες τεχνολογίες ως δομικά στοιχεία για να πετύχει τους στόχους του. Για παράδειγμα, ο Chubby, μια υπηρεσία κατανεμημένης κλειδαριάς που εφαρμόζεται στην Google και παρέχει ισχυρή συνέπεια, υποστηρίζει το σύστημα ονομάτων του framework το οποίο επιτρέπει τις εργασίες να τοποθετούνται από πελάτες

και άλλες υπηρεσίες. Για να ενεργοποιηθεί η παρακολούθηση των εργασιών, αυτές έχουν ενσωματωμένο διακομιστή HTTP που δημοσιεύει πληροφορίες σχετικά με την κατάστασή τους καθώς και διάφορες μετρήσεις απόδοσης. Υποστηρίζει τη μονάδα λογιστικής (accounting module) με την καταγραφή δεδομένων όπως την υποβολή εργασιών και την χρήση πόρων των εργασιών μέσω του Infrastore, ένα κλιμακωτό περιβάλλον δεδομένων μόνο για ανάγνωση με μια διαδραστική δομημένη διεπαφή γλώσσας μέσω του Dremel που αναπτύχθηκε επίσης στην Google. Ένα περιβάλλον με βάση το Paxos χρησιμοποιείται για τη συντήρηση του αντιγράφου της κατάστασης των πέντε κύριων διεργασιών με έναν ιδιαίτερα διαθέσιμο τρόπο. Τέλος, το chroot jail του Linux χρησιμοποιείται ως μέρος της στοίβας του υποδοχέα για να απομονώνει με ασφάλεια τις πολλαπλές εργασίες που εκτελούνται στο ίδιο μηχάνημα.

Συνολικά, το Borg είναι ένα από τα πιο εξελιγμένα συστήματα. Υποστηρίζει χαρακτηριστικά όπως υπερεγγραφή με εκτίμηση κατανάλωσης πόρων, το μικτό φόρτο εργασίας, τα λεπτομερή αιτήματα για τους πόρους και την προτίμηση των εργασιών, μεταξύ άλλων. Ο ρυθμός χρησιμοποίησης της συστοιχίας του αναφέρθηκε ότι κυμαινόταν μεταξύ 60% και 70% και η παραγωγή του ήταν περίπου 10.000 εργασίες ανά λεπτό σε μία συστοιχία αποτελούμενη από δεκάδες χιλιάδες κόμβους.

Kubernetes

Το Kubernetes είναι ένα framework σχεδιασμένο για να διαχειρίζεται φόρτους εργασίας των υποδοχέων σε συστοιχίες. Το βασικό δομικό στοιχείο στο Kubernetes είναι το pod. Ένα pod περικλείει έναν ή περισσότερους σφιχτά συνδεδεμένους υποδοχείς οι οποίοι είναι τοποθετημένοι σε κοντινά σημεία μεταξύ τους και μοιράζονται το ίδιο σετ πόρων. Τα pods περιλαμβάνουν επίσης τους πόρους αποθήκευσης, μία δικτυακή IP και ένα σύνολο επιλογών που διέπουν τον τρόπο με τον οποίο οι υποδοχείς του pod θα πρέπει να τρέξουν. Ένα pod έχει σχεδιαστεί για να τρέχει ένα μοναδικό στιγμιότυπο μιας εφαρμογής. Με αυτόν τον τρόπο, μπορούν να χρησιμοποιηθούν πολλαπλά pods για να κλιμακωθεί μια εφαρμογή οριζόντια για παράδειγμα. Η ποσότητα της CPU, της μνήμης και η προσωρινή αποθήκευση που χρειάζεται ένας υποδοχέας μπορεί να οριστεί κατά τη δημιουργία ενός pod. Αυτές οι πληροφορίες μπορούν στη συνέχεια να χρησιμοποιηθούν από τον υπεύθυνο χρονοπρογραμματισμού (scheduler) για να παρθούν αποφάσεις

σχετικά με την τοποθέτηση των pods. Αυτοί οι υπολογιστικοί πόροι μπορούν να καθοριστούν τόσο ως ζητούμενο ποσό όσο και ως ανώτατο όριο για το ποσό που ο υποδοχέας επιτρέπεται να καταναλώσει.

Ο χρονοπρογραμματιστής (scheduler) εξασφαλίζει ότι το συνολικό ποσό των αιτήσεων για υπολογιστικούς πόρους όλων των pods που τοποθετούνται σε έναν κόμβο (node) δεν υπερβαίνει την χωρητικότητα του κόμβου (node). Αυτό ισχύει ακόμη και αν η πραγματική κατανάλωση πόρων είναι πολύ χαμηλή. Ο λόγος πίσω από αυτό είναι η προστασία των εφαρμογών από την έλλειψη πόρων σε έναν κόμβο όταν η χρήση πόρων αυξάνεται αργότερα χρονικά (π.χ. κατά τη διάρκεια μιας ημερήσιας αιχμής). Αν ένας υποδοχέας υπερβαίνει το όριο της μνήμης του, ενδέχεται να τερματιστεί και να επανεκκινηθεί αργότερα. Εάν υπερβαίνει το αίτημα για μνήμη, μπορεί να τερματιστεί όταν εξαντληθεί η μνήμη του κόμβου. Όσον αφορά τη χρήση της CPU, οι υποδοχείς ενδέχεται να μην επιτρέπονται να υπερβούν τα όριά της για χρονικά διαστήματα, αλλά δεν θα τερματιστούν γι 'αυτό. Από την άλλη πλευρά, οι υποδοχείς και τα pods που υπερβαίνουν το όριο αποθήκευσης τερματίζονται. Άλλοι πόροι (που ονομάζονται εκτεταμένοι πόροι) μπορούν να καθοριστούν, για τη ενημέρωση νέων πόρων σε επίπεδο κόμβου. η διαχείριση αυτών των πόρων γίνεται από τον προγραμματιστή (scheduler) για να διασφαλιστεί ότι δεν υπερβαίνεται το διαθέσιμο ποσό που κατανέμεται ταυτόχρονα σε pods.

Από τεχνική άποψη, το Kubernetes επιτρέπει τη χρήση διαφόρων τύπων εκτελέσιμων υποδοχέων, με το Docker και το rkt να υποστηρίζονται από την πλατφόρμα. Επιπλέον, η υποστήριξη του Kubernetes στη διαχείριση των κόμβων και των εργασιών της συστοιχίας (cluster) είναι το etcd, ένα εξαιρετικά διαθέσιμο κατανεμημένο κλειδί αποθήκευσης ανοιχτού κώδικα. Συγκεκριμένα, το etcd χρησιμοποιείται για την αποθήκευση όλων των δεδομένων της συστοιχίας και ενεργεί ως η μόνη πηγή της πρόσβασης για όλα τα στοιχεία (components) της πλατφόρμας (framework).

Συνολικά, το Kubernetes είναι ένα εξαιρετικά ώριμο σύστημα. προέκυψε μετά από 10 χρόνια εμπειρίας της Google με το Borg και το Omega και είναι το κορυφαίο σύστημα διαχείρισης συστοιχίας που βασίζεται σε υποδοχείς, με εκτεταμένη υποστήριξη από την κοινότητα, και βάση ανάπτυξης. Παρέχει στους χρήστες ένα ευρύ φάσμα

επιλογών για τη διαχείριση των pods τους και τον τρόπο με τον οποίο έχουν προγραμματιστεί, επιτρέποντας ακόμη και στους προσαρμοσμένους προγραμματιστές (customized schedulers) να συνδέονται εύκολα στο σύστημα. Αξίζει να παρατηρηθεί ότι ο ενσωματωμένος προγραμματιστής (built-in scheduler) του Kubernetes χαρακτηρίζεται ως αποκεντρωμένος και μονολιθικός. Ωστόσο, αυτό μπορεί να ξεπεραστεί με προγραμματιστές-προσθήκες στο σύστημα (pluggable schedulers), οι οποίοι στη βάση της υλοποίησής τους, μπορεί να είναι είτε συγκεντρωτικοί είτε μονολιθικοί αποκεντρωμένοι ή αρθρωτοί αποκεντρωμένοι. Συμπερασματικά, αν και η απόδοση και η κλιμάκωση του Kubernetes μπορεί να μην έχει ακόμα φτάσει στα επίπεδα βιομηχανικών συστημάτων όπως το Borg, από την έκδοση 1.10, το Kubernetes είναι σε θέση να υποστηρίξει συστοιχίες μέχρι 5000 εκατοντάδες κόμβους, κάτι που ταιριάζει στις ανάγκες πολλών οργανισμών στις μέρες μας[2].

Το Kubernetes παρέχει τους ακόλουθους μηχανισμούς ελαστικότητας.

HPA (Horizontal Pod Autoscaler)

Το HPA κλιμακώνει αυτόματα τον αριθμό των pods σε έναν replication controller, deployment ή replica set και βασίζεται στην παρατήρηση της χρησιμοποίησης της CPU (ή, με την υποστήριξη προσαρμοσμένων μετρήσεων, σε μερικές άλλες μετρήσεις που παρέχονται από την εφαρμογή). Σημειώστε ότι το Horizontal Pod Autoscaling δεν ισχύει για αντικείμενα που δεν μπορούν να κλιμακωθούν, για παράδειγμα DaemonSets.

Το HPA υλοποιείται ως Kubernetes API πόρος και controller. Ο πόρος καθορίζει τη συμπεριφορά του ελεγκτή. Ο ελεγκτής ρυθμίζει περιοδικά τον αριθμό των replicas σε έναν controller ή το deployment, ώστε να ταιριάζει με την παρατηρούμενη μέση χρήση της CPU στο στόχο που καθορίζει ο χρήστης.

Το HPA υλοποιείται ως βρόχος ελέγχου, με μια χρονική περίοδο που ελέγχεται από τη σημαία --horizontal-pod-autoscaler-sync-period του διαχειριστή ελεγκτή (με προκαθορισμένη τιμή 15 δευτερολέπτων).

Κατά τη διάρκεια κάθε περιόδου, ο διαχειριστής ελεγκτή ρωτά τη χρήση των πόρων κατά τις μετρήσεις που καθορίζονται σε κάθε ορισμό του HPA. Ο διαχειριστής του ελεγκτή αποκτά τις μετρήσεις είτε από το API μετρήσεων πόρων (για μετρήσεις

πόρων ανά pod) είτε από το προσαρμοσμένο API μετρήσεων (για όλες τις άλλες μετρήσεις).

Λεπτομέρειες του Αλγόριθμου

Από την πιο βασική άποψη, ο ελεγκτής του HPA λειτουργεί με βάση την αναλογία μεταξύ της επιθυμητής τιμής μετρικής και της τρέχουσας μετρικής τιμής:

```
desiredReplicas = ceil[currentReplicas * ( currentMetricValue / desiredMetricValue )]
```

Για παράδειγμα, εάν η τρέχουσα τιμή μέτρησης είναι 200m και η επιθυμητή τιμή είναι 100m, ο αριθμός των αντιγράφων θα διπλασιαστεί, αφού 200.0 / 100.0 == 2.0. Αν η τρέχουσα τιμή είναι 50m, θα μειωθούν κατά το ήμισυ ο αριθμός των αντιγράφων, από 50,0 / 100,0 == 0,5 [9].

VPA (Vertical Pod Autoscaler)

Το VPA (Vertical Pod Autoscaler) απαλλάσσει τους χρήστες από την ανάγκη να ορίσουν ενημερωμένα όρια πόρων και αιτήσεων για τους containers στα pods τους. Όταν διαμορφώνεται, ορίζει τις αιτήσεις αυτόματα με βάση τη χρήση και έτσι επιτρέπει τον σωστό προγραμματισμό στα nodes έτσι ώστε να είναι διαθέσιμο το κατάλληλο ποσό πόρων για κάθε pod. Διατηρεί επίσης τους λόγους μεταξύ των ορίων και των αιτήσεων που καθορίστηκαν στην αρχική διαμόρφωση των containers.

Μπορεί να κάνει down-scale τα pods που είναι υπερβολικά απαιτητικά σε πόρους, όσο και να κάνει up-scale τα pods εκείνα που ζητάνε σε μειωμένη ποσότητα πόρους με βάση τη χρήση τους με την πάροδο του χρόνου.

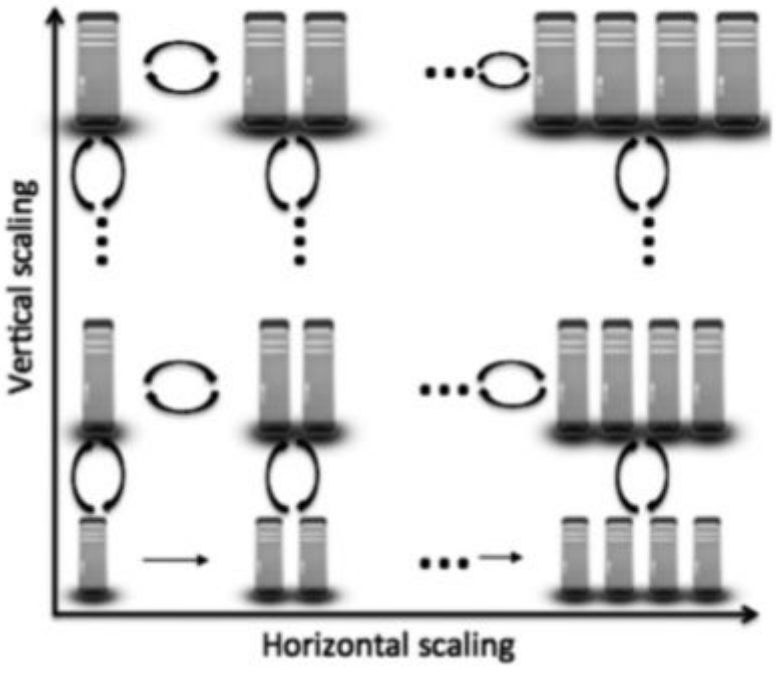
Στοιχεία του VPA

Το VPA αποτελείται από 3 στοιχεία:

- Recommender - παρακολουθεί την τρέχουσα και την προηγούμενη κατανάλωση πόρων και βασισμένο σε αυτό, παρέχει τις συνιστώμενες τιμές των αιτήσεων σε CPU και μνήμη των containers.
- Updater - ελέγχει ποιά από τα διαχειριζόμενα pods έχουν σωστά καθορισμένους πόρους και, αν όχι, τα τερματίζει έτσι ώστε να μπορούν να δημιουργηθούν από τους controllers με τα ενημερωμένα αιτήματα.
- Admission Plugin - ορίζει τα σωστά αιτήματα πόρων στα νέα pods [10].

2.4 Ελαστικότητα (Elasticity)

Η ελαστικότητα ορίζεται ως η ικανότητα ενός συστήματος να προσθέτει και να αφαιρεί πόρους (όπως πυρήνες CPU, μνήμη, VM και στιγμιότυπα υποδοχέων) "on the fly" για να προσαρμοστούν στην μεταβολή του φορτίου σε πραγματικό χρόνο. Η ελαστικότητα είναι μια δυναμική ιδιότητα για το υπολογιστικό νέφος. Υπάρχουν δύο τύποι ελαστικότητας η οριζόντια και κάθετη. Η οριζόντια ελαστικότητα συνίσταται στην προσθήκη ή την αφαίρεση περιπτώσεων υπολογιστικών πόρων που σχετίζονται με μια εφαρμογή. Η κάθετη ελαστικότητα συνίσταται στην αύξηση ή μείωση των στιγμιοτύπων των υπολογιστικών πόρων, όπως χρόνος CPU, πυρήνες, μνήμη και εύρος ζώνης δικτύου. Ενδεικτικά παρουσιάζονται η κάθετη ελαστικότητα και η οριζόντια παρακάτω.



Οριζόντια και κάθετη ελαστικότητα[3]

Υπάρχουν άλλοι όροι όπως η επεκτασιμότητα και η αποδοτικότητα, που συνδέονται με την ελαστικότητα, αλλά το νόημά τους είναι διαφορετικό από την ελαστικότητα, ενώ χρησιμοποιούνται εναλλακτικά σε μερικές περιπτώσεις. Η επεκτασιμότητα είναι η ικανότητα του συστήματος να διατηρεί αυξανόμενα φόρτη

εργασίας χρησιμοποιώντας πρόσθετους πόρους, είναι ανεξάρτητη του χρόνου και είναι παρόμοια με την κατάσταση τροφοδοσίας στην ελαστικότητα αλλά ο χρόνος δεν έχει καμία επίδραση στο σύστημα (στατική ιδιότητα). Για να έχουμε μια πλήρη κατανόηση, συνάγουμε την ακόλουθη εξίσωση που συνοψίζει το ελαστικότητα στο υπολογιστικό νέφος:

Ελαστικότητα = επεκτασιμότητα + αυτοματοποίηση + βελτιστοποίηση

Αυτό σημαίνει ότι η ελαστικότητα είναι χτισμένη πάνω στην επεκτασιμότητα. Μπορεί να θεωρηθεί ως η αυτοματοποίηση της έννοιας της επεκτασιμότητας, ωστόσο, στοχεύει να βελτιστοποιήσει στην καλύτερη περίπτωση και το συντομότερο δυνατόν τους πόρους σε δεδομένη στιγμή. Ένας άλλος όρος που σχετίζεται με την ελαστικότητα είναι η απόδοση, η οποία χαρακτηρίζει τον τρόπο με τον οποίο ο πόρος του υπολογιστικού νέφους μπορεί να χρησιμοποιηθεί αποτελεσματικά καθώς κλιμακώνεται προς τα πάνω ή προς τα κάτω. Είναι το ποσό των πόρων που καταναλώνονται για την επεξεργασία ενός δεδομένου όγκου εργασίας, όσο χαμηλότερο είναι αυτό το ποσό, τόσο υψηλότερη είναι η αποδοτικότητα ενός συστήματος. Το ποσό των πόρων μπορεί να σχετίζεται με το κόστος, την κατανάλωση ενέργειας κλπ., ανάλογα με τον στοχευόμενο πόρο. Γενικά, αυτό είναι ένα μέτρο για το πόσο καλά χρησιμοποιεί η εφαρμογή τους πόρους που παρέχονται. Περισσότερη ελαστικότητα στο υπολογιστικό νέφος οδηγεί σε υψηλότερη απόδοση. Οι διαδικασίες παροχής πόρων και χρονοπρογραμματισμού (δηλ. εργασίες ή αιτήματα πελατών για στιγμιότυπα) σχετίζονται με την ελαστικότητα καθώς προσπαθούν να παρέχουν στιγμιότυπα αλλά σε αντιστοιχία με τα θέλω του παρόχου και του πελάτη.

Αξίζει να σημειωθεί ότι η αύξηση ή η μείωση των πόρων μπορεί να οδηγήσει σε απόκλιση του τρέχοντος ποσού των διατεθημένων πόρων από την πραγματική ζήτηση πόρων. Η ακρίβεια των συστημάτων ελαστικότητας ποικίλλει από το ένα σύστημα στο άλλο. Η υπερ-χρησιμοποίηση και η υποχρησιμοποίηση είναι δύο σημαντικοί παράγοντες που χαρακτηρίζουν ένα ελαστικό σύστημα. Το σύστημα εισέρχεται σε κατάσταση υπερχρησιμοποίησης μόλις οι πόροι που παρέχονται είναι περισσότεροι από τους

απαιτούμενους πόρους από τον καταναλωτή. Δηλαδή αν έχουμε έναν όρο παροχή (Π) και έναν όρο ζήτηση (Z), τότε έχουμε Π > Z. Αν και το QoS μπορεί να επιτευχθεί, η κατάσταση αυτή υπερ-χρησιμοποίησης οδηγεί σε πρόσθετο και περιττό κόστος ενοικίασης των πόρων του νέφους (cloud). Η υποχρησιμοποίηση πραγματοποιείται όταν οι πόροι που παρέχονται είναι λιγότεροι από τους πόρους που απαιτούνται από τον καταναλωτή, δηλαδή όταν Π < Z. Και αυτό προκαλεί υποβάθμιση της απόδοσης. Δεν υπάρχει κοινή μεθοδολογία για τη μέτρηση ή τον προσδιορισμό των χρονικών ή ποσοτικών μετρήσεων της ελαστικότητας. Ένας καταναλωτής μπορεί να μετρήσει την καθυστέρηση που χρειάζεται για να του παραχτούν πόροι και να του αποσβεφτούν, προσθετικά στο άθροισμα των καθυστερήσεων για την υπερ-χρησιμοποίηση και την υποχρησιμοποίηση δηλώνει την εκτίμηση των διαφόρων ελαστικών συστημάτων [3].

2.5 Ελαστικότητα στο υπολογιστικό νέφος (Cloud Elasticity)

Το υπολογιστικό νέφος αποτελεί ένα μοντέλο ανάπτυξης, το οποίο στοχεύει στη μείωση του στιγμιαίου κόστους των υπολογιστικών πόρων μέσω της μίσθωσης δυναμικά προσαρμοσμένων εικονικών πόρων, οι οποίοι μπορούν να χρησιμοποιηθούν κατά παραγγελία. Οι εικονικοί πόροι είναι εικονικές εκδοχές πραγματικών πηγών, συνήθως με τη μορφή εικονικών μηχανών (VM), τα οποία αξιοποιούν την τεχνολογία εικονικοποίησης. Το προσφερόμενο pay-as-you-go μοντέλο τιμολόγησης που συνοδεύεται από τον ελαστικό χειρισμό πόρων, βοήθησε στην ευρεία αποδοχή της ανάπτυξης σε νέφος (cloud), καθότι ο πελάτης είναι υποχρεωμένος να πληρώσει μόνο για τον χρησιμοποιούμενο πόρο. Ως εκ τούτου, το υπολογιστικό νέφος (cloud computing) κατάφερε να κάνει την πρόβλεψη των απομακρυσμένων υπολογιστικών πόρων (π.χ. VMs) την κύρια επιλογή όχι μόνο για επιστημονικά ιδρύματα αλλά κάθε μέγεθος οργανισμών και επιχειρήσεων. Ωστόσο, ο αποτελεσματικός χειρισμός των πόρων αποτελεί βασική πτυχή της μείωσης του κόστους ανάπτυξης.

2.5.1 Κατηγοριοποίηση και ταξινόμηση

Προκειμένου να δοθεί μια συνοπτική ταξινόμηση των υφιστάμενων προσεγγίσεων στην ελαστικότητα του νέφους (cloud elasticity), παρουσιαζουμε μια ταξινόμηση που θα επιτρέψει στο έργο μας να ρίξει φως στις διαφοροποιημένες πτυχές των διαφόρων προτάσεων. Η ταξινόμηση αποτελείται από τις ακόλουθες διαστάσεις:

Πεδίο εφαρμογής (Scope)

Αυτή η πτυχή χωρίζεται σε δύο κατηγορίες ταξινόμησης (i) τον ενεργοποιητή (enactor) και (ii) τον τύπο εφαρμογής (application type). Το πρώτο δείχνει αν η ελαστική τεχνολογία εφαρμόζεται από τον πάροχο υπολογιστικού νέφους (Cloud Provider - CP) ή από τον χρήστη της υποδομής του υπολογιστικού νέφους, ο οποίος αναπτύσσει και διαχειρίζεται εφαρμογές νέφους πάνω στην υποδομή του νέφους (Service Provider - SP). Ο τύπος της εφαρμογής δείχνει εάν η πρόταση αναφέρεται στον ελαστικό χειρισμό (elastic handling) ενός ειδικού τύπος εφαρμογής σύννεφου από την ακόλουθη λίστα: σχεσιακές βάσεις δεδομένων (DBs), βάσεις δεδομένων NoSQL (DBs NoSQL), εφαρμογές πολλαπλών επιπέδων (π.χ., τυπικές business web εφαρμογές), γενικές (αν το εργαλείο είναι application-agnostic) ή αποθήκευσης.

Σκοπός (Purpose)

Σε αυτή τη διάσταση, κατατάσσουμε τις τεχνικές σύμφωνα με το σκοπό των ενεργειών ελαστικότητας. Ο σκοπός μπορεί να είναι ένας από τους ακόλουθους: (i) Απόδοση, (ii) Διαθεσιμότητα, (iii) Κόστος, (iv) Ενέργεια. Η απόδοση, αναφέρεται στη συντήρηση ή εγγύηση των αποδεκτών, είτε στη συμφωνία χρήστη στην εξειδικευμένη στην απόδοση των εφαρμογών. Η διαθεσιμότητα αναφέρεται στον βαθμό στον οποίο οι εφαρμογές και οι πόροι βρίσκονται σε λειτουργική κατάσταση στην συγκεκριμένη χρονική στιγμή όταν την χρειάζονται οι τελικοί χρήστες. Το κόστος αναφέρεται είτε στο μείωση του λειτουργικού κόστους της εφαρμογής που αναπτύσσεται στο νέφος, συνήθως διατηρώντας επίσης το στόχο απόδοσης ή τη συντήρηση του σε κατώτατα όρια κόστους κάτω από συγκεκριμένους περιορισμούς επιδόσεων. Τέλος, η κατηγορία ενέργεια, συνδέεται στενά με το κόστος, αλλά καλύπτει ελαστικές τεχνικές, οι οποίες έχουν ως άμεσο στόχο την ελαχιστοποίηση του ενεργειακού αποτυπώματος (footprint).

Λήψη απόφασης (Desicion Making)

Υπάρχουν τέσσερα διαφορετικά κριτήρια κατηγοριοποίησης που χαρακτηρίζουν την επεξεργασία της διαδικασίας λήψης αποφάσεων κάθε εργασίας στην ταξινόμησή μας, δηλαδή (1) Trigger, που δείχνει αν ενεργοποιείται ο μηχανισμός ελαστικότητας με αντιδραστικό ή προνοητικό (reactive or proactive) τρόπο. (2) Μηχανισμός, ο οποίος αναφέρεται στην μεθοδολογία λήψης απόφασης· (3) μοντέλο πρόβλεψης (Prediction

Model), το οποίο υποδηλώνει τη χρήση μοντέλου για την πρόβλεψη μελλοντικών μεταβολών εισερχόμενου φορτίου ή ειδικών τιμών μετρήσεων και (4) μοντέλο συστήματος (SM), το οποίο αναφέρεται στη χρήση ενός μοντέλου που αντιπροσωπεύει την (ελαστική) συμπεριφορά του συστήματος, πάνω στο οποίο η ολοκληρωμένη πολιτική ελαστικότητας έχει βασιστεί (π.χ., ουρές). Οι μηχανισμοί ελαστικότητας ταξινομούνται περαιτέρω στις ακόλουθες κατηγορίες: (1) Βασισμένη σε κανόνες, (2) Μαθηματική / Στατιστική Βελτιστοποίηση, (3) Μηχανική Μάθηση, (4) Θεωρία Ελέγχου και (5) Έλεγχος μοντέλου σύμφωνα με το κύριο πεδίο στο οποίο η πολιτική ελαστικότητας ανήκει.

Ελαστική Δράση (Elastic Action)

Η ελαστικότητα των πηγών του νέφους μπορεί να εφαρμοστεί σε διάφορες μορφές και μπορεί να αναφέρεται σε τροποποιήσεις στο (1) μέγεθος (Vertical Scaling (VS)), (2) την τοποθεσία (VMLM)) ή (3) τον αριθμό των VM που χρησιμοποιούνται (Οριζόντια κλιμάκωση (HS)). Παραδείγματα αυτών των τριών τύπων ελαστικότητας είναι η κατανομή περισσότερης μνήμης ή CPU σε VM, η μετακίνηση VM σε λιγότερο φορτωμένη φυσική μηχανή και αύξηση του αριθμού των VM μιας συστοιχίας εφαρμογών, αντίστοιχα. Η ελαστική δράση περιλαμβάνει επιπλέον δύο άλλους τύπους ελαστικότητας, (4) την επαναδιαμόρφωση εφαρμογής (AR), όπου το ελαστικό εργαλείο είναι ικανό για το χειρισμό συγκεκριμένων πτυχών της εφαρμογής (π.χ., μέγεθος cache DB) και (5) μετανάστευση της εφαρμογής (Application Live Migration), όπου μόνο εξαρτήματα που σχετίζονται με την εφαρμογή μεταναστεύουν αντί του πλήρους VM, όπως παραδείγματα βάσεων δεδομένων.

Παροχέας (Provider)

Αυτή η κατηγορία ταξινόμησης αναφέρεται στον αριθμό των παρόχων υποδομής του νέφους που υποστηρίζει το ελαστικό εργαλείο ταυτόχρονα. Οι πιθανές τιμές είναι (1) Μοναδιαία, που υποδηλώνει ότι υποστηρίζεται μόνο ένας πάροχος σύννεφων, (2) Μοναδιαία *, η οποία υποδηλώνει ότι υποστηρίζονται περισσότεροι από ένας πάροχοι όχι ταυτόχρονα και (3) Πολλαπλοί, όπου ο έλεγχος ελαστικότητας εξαπλώνεται σε πολλούς παρόχους συννέφου ταυτόχρονα.

Εκτίμηση (Evaluation)

Τέλος, η τελευταία πτυχή αναφέρεται στο είδος της αξιολόγησης της κάθε εργασίας. Οι πιθανές τιμές είναι: (1) Προσομοίωση, όπου τα αποτελέσματα λαμβάνονται με βάση υπολογισμούς σε προσομοιωμένο τεχνητό περιβάλλον (π.χ., OMNeT ++), (2) Εξομοίωση όπου τα αποτελέσματα των αξιολογήσεων λαμβάνονται σε ένα τεχνητό περιβάλλον που συμπεριφέρεται σύμφωνα με τον πραγματικό κόσμο και (3) Πραγματικό, όπου το ελαστικό εργαλείο εφαρμόζεται σε μια πραγματική υποδομή στο σύννεφο [6].

3 Πειραματική Προσέγγιση

Κατά την πειραματική προσέγγιση της εργασίας μας επικεντρωθήκαμε στην εφαρμογή των μηχανισμών ελαστικότητας στο περιβάλλον του Kubernetes. Το επιλέξαμε ως το εργαλείο εκείνο με το όποιο θα πραγματοποιήσουμε τα πειράματα μας καθώς είναι α) ευρέως χρησιμοποιούμενο και β) υποστηρίζει μηχανισμους ελαστικότητας μικρο-υπηρεσιών.

Στη συνέχεια του κεφαλαίου θα γίνει εισαγωγή στα εργαλεία που χρησιμοποιήθηκαν στην μελέτη της πτυχιακής καθώς και οι λεπτομέρειες υλοποίησης του πειραματικού περιβάλλοντος. Πιο συγκεκριμένα θα αναλύσουμε την πλατφόρμα του Kubernetes, τα εργαλεία με τα οποία πραγματοποιήσαμε τα πειραματικά μας σενάρια για να εφαρμόσουμε φόρτο στον επεξεργαστή στη μνήμη και στο δίκτυο όπως και τα εργαλεία με βάση τα οποία πραγματοποιήσαμε τις μετρήσεις που καταγράφονται στο 4ο κεφάλαιο. Ακόμη θα αναλύσουμε πως μεσω yamls αρχείων παραμετροποιήσαμε το περιβάλλον μας για να υλοποιηθεί η κάθετη και η οριζόντια ελαστικότητα στο περιβάλλον του Kubernetes.

3.1 Επισκόπηση Εργαλείων

Τα εργαλεία τα οποία χρησιμοποιήθηκαν στην παρούσα μελέτη είναι κατ’αρχήν οι υπηρεσίες με τις οποίες ελέγξαμε την αποδοτικότητα της οριζόντιας και κάθετης ελαστικότητας στο περιβάλλον του Kubernetes όπως και οι υπηρεσίες με τις οποίες

πήραμε τις μετρήσεις εκείνες που προσομοιώσαμε τους πελάτες και καταγράψαμε τελικά σε αποτελέσματα.

Πιο αναλυτικά χρησιμοποιήθηκε μια απαιτητική σε επεξεργαστικούς πόρους εφαρμογή (CPU-intensive) η οποία μέσω του apache server εκτελεί έναν αλγόριθμο υπολογισμών στο παρασκήνιο (back-end). Ο αλγόριθμος αυτός για κάθε αίτημα πραγματοποιεί μια αριθμητική πράξη επαναλαμβανόμενες φορές έτσι ώστε να δημιουργεί υπολογιστικό φόρτο στον επεξεργαστή. Επισής χρησιμοποιήθηκε μια διαδικτυακή εφαρμογή (web-stress-simulator) μέσω της οποίας προσομοιώνεται επεξεργαστικός φόρτος. Και αυτή η εφαρμογή τρέχει αλγόριθμους στο παρασκήνιο (back-end) οι οποίοι μπορούν να παραμετροποιηθουν για κάθε αίτημα να επιβάλλουν στο σύστημα διαφορετικό φόρτο εργασίας (π.χ. υπερφόρτωση δικτύου ή μνήμης RAM). Το εργαλείο συγκριτικής αξιολόγησης που χρησιμοποιούμε για την προσομοίωση του φόρτου εργασίας (αιτήματα πελατών) αλλα και για την καταγραφή των μετρήσεων για την αξιολόγηση των εφαρμογών (χρόνος εξυπηρέτησης αιμάτων, ολοκληρωμένα αιτήματα) είναι το Apache Benchmark.

Php-apache

Το Apache είναι ένας διακομιστής ιστού (web server). Εφαρμόζει το πρωτόκολλο HTTP και είναι σε θέση να εξυπηρετεί τους πελάτες του με περιεχόμενο (σε HTML) μέσω του πρωτοκόλλου.

Η PHP είναι μια γλώσσα πλευράς διακομιστή (server-side language) που εξάγει (συνήθως) αποτελέσματα HTML. Όταν καλείται μια διεύθυνση URL που ζητά ένα αρχείο .php, το Apache ρωτά την PHP (το runtime που είναι υπεύθυνο για την εκτέλεση του κώδικα που έχει υλοποιηθεί) να το επεξεργαστεί και αναμένει όποια έξοδο έρχεται από αυτό, στη συνέχεια το στέλνει πίσω (μέσω HTTP) στο λογισμικό του προγράμματος περιήγησης του πελάτη που το ζήτησε.

Χρησιμοποιούμε την συγκεκριμένη εφαρμογή γιατί είναι απαιτητική σε κατανάλωση επεξεργαστικής ισχύς.

Στην παρακάτω εικόνα περιγράφεται ένα παράδειγμα της λειτουργικότητας της συγκεκριμένης εφαρμογής:

```php
<?php
  $x = 0.0001;
  for ($i = 0; $i <= 1000000; $i++) {
    $x += sqrt($x);
  }
  echo "OK!";
?>
```

Εικόνα 3 – Λειτουργικότητα υπηρεσίας απαιτητικής σε CPU

Web-Stress-Simulator

Πρόκειται για μια Διαδικτυακή εφαρμογή (web app) προσομοίωσης διαφόρων συνθηκών, μέσω της οποίας μπορούμε να υποβάλλουμε τον υπολογιστή που την εκτελεί σε διαφορετικό φόρτο εργασίας (δλδ, υψηλό φορτίο cpu, υψηλή κατανάλωση μνήμης, υψηλή κατανάλωση εύρους ζώνης, υψηλές καθυστερήσεις backend, ασυνήθιστοι κωδικοί επιστροφής HTTP). Είναι συσκευασμένο ως docker container για απλότητα χρήσης [11].

Apache-Benchmark

Το Apache-Benchmark (ab) είναι ένα εργαλείο από τον οργανισμό Apache για τη συγκριτική αξιολόγηση ενός διακομιστή ιστού Hypertext Transfer Protocol (HTTP). Παρόλο που έχει σχεδιαστεί για τη μέτρηση της απόδοσης του διακομιστή ιστού Apache, μπορεί να χρησιμοποιηθεί και για τη δοκιμή οποιουδήποτε άλλου διακομιστή ιστού. Με αυτό το εργαλείο, μπορεί να γνωρίζει ο χρήστης γρήγορα πόσα αιτήματα ανά δευτερόλεπτο ο διακομιστής ιστού μπορεί να εξυπηρετήσει.

Χαρακτηριστικά του Apache Benchmark:

- Ως λογισμικό ανοιχτού κώδικα, είναι ελεύθερα διαθέσιμο.

- Πρόκειται για ένα απλό πρόγραμμα υπολογιστή γραμμής εντολών.
- Πρόκειται για ένα εργαλείο ανεξάρτητο από την πλατφόρμα. Σημαίνει ότι μπορεί να χρησιμοποιηθεί σε Linux / Unix ή σε διακομιστή Windows εξίσου καλά.
- Μπορεί να διεξάγει έλεγχο φορτίων και επιδόσεων μόνο για διακομιστή ιστού - HTTP ή HTTPS [10].

3.2 Kubernetes

Το Kubernetes είναι μια φορητή, επεκτάσιμη πλατφόρμα ανοιχτού κώδικα για τη διαχείριση των φόρτων εργασίας και των υπηρεσιών σε υποδοχείς, που διευκολύνει τόσο τη δηλωτική διαμόρφωση όσο και την αυτοματοποίηση. Έχει ένα μεγάλο, ταχέως αναπτυσσόμενο οικοσύστημα. Οι υπηρεσίες, η υποστήριξη και τα εργαλεία του Kubernetes είναι ευρέως διαθέσιμα.

Το Kubernetes έχει πολλά χαρακτηριστικά (features). Μπορεί να θεωρηθεί ως:

- μια πλατφόρμα δοχείων
- μια πλατφόρμα μικρο-υπηρεσιών
- μια φορητή πλατφόρμα υπολογιστικού νέφους και πολλά άλλα.

Το Kubernetes παρέχει ένα περιβάλλον διαχείρισης container-centric. Οργανώνει (orchestrates) τον υπολογισμό, την δικτύωση και την υποδομή της αποθήκευσης εξ ονόματος των φόρτων εργασίας των χρηστών. Αυτό παρέχει μεγάλο μέρος της απλότητας της πλατφόρμας ως υπηρεσίας (PaaS) με την ευελιξία της υποδομής ως υπηρεσίας (IaaS) και επιτρέπει τη φορητότητα μεταξύ των παρόχων υποδομής.

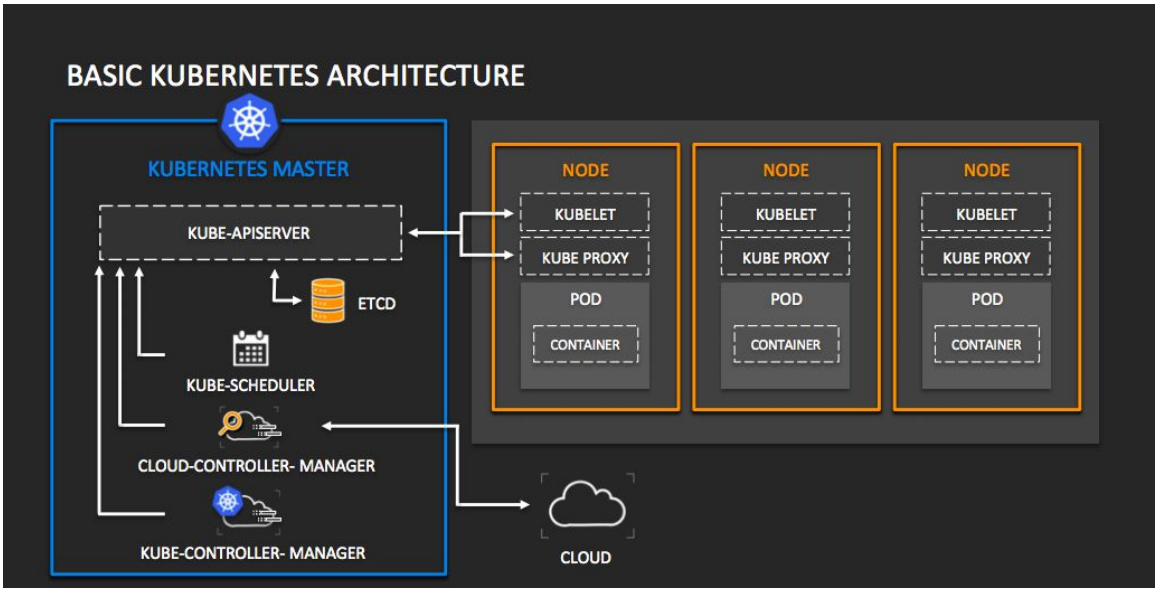
Παρόλο που η πλατφόρμα Kubernetes παρέχει πολύ λειτουργικότητα, υπάρχουν πάντα νέα σενάρια που μπορούν να επωφεληθούν από νέα χαρακτηριστικά (features). Οι ροές εργασιών που σχετίζονται με την εφαρμογή μπορούν να βελτιστοποιηθούν για να επιταχύνουν την ταχύτητα του προγραμματιστή. Η ad hoc ενορχήστρωση που είναι αποδεκτή αρχικά συχνά απαιτεί ισχυρή αυτοματοποίηση σε κλίμακα. Αυτός είναι ο λόγος για τον οποίο το Kubernetes σχεδιάστηκε επίσης για να χρησιμεύσει ως πλατφόρμα για την οικοδόμηση ενός οικοσυστήματος εξαρτημάτων (components) και

εργαλείων για να διευκολύνει την ανάπτυξη, την κλιμάκωση και τη διαχείριση εφαρμογών.

Οι ετικέτες (labels) επιτρέπουν στους χρήστες να οργανώνουν τους πόρους τους όπως αυτοί επιθυμούν. Οι σχολιασμοί (annotations) επιτρέπουν στους χρήστες να χαρακτηρίζουν τους πόρους με προσαρμοσμένες πληροφορίες για να διευκολύνουν τις ροές εργασίας τους και να παρέχουν έναν εύκολο τρόπο για την κατάσταση των σημείων ελέγχου στα εργαλεία διαχείρισης.

Επιπλέον, το επίπεδο ελέγχου του Kubernetes βασίζεται στα ίδια APIs που είναι διαθέσιμα και για προγραμματιστές και για χρήστες. Οι χρήστες μπορούν να γράψουν τους δικούς τους ελεγκτές (controllers), όπως χρονοπρογραμματιστές (schedulers), με τα δικά τους API που μπορούν να στοχευθούν από ένα εργαλείο γραμμής εντολών (command-line tool) γενικού σκοπού.

Αυτός ο σχεδιασμός επέτρεψε τη δημιουργία πολλών άλλων συστημάτων πάνω στο Kubernetes [7].



Εικόνα 4 – Αρχιτεκτονική Kubernetes [17]

3.2.1 Αντικείμενα του Kubernetes

Το Kubernetes ορίζει ένα σύνολο δομικών στοιχείων ("primitives"), τα οποία συλλογικά παρέχουν μηχανισμούς που αναπτύσσουν, συντηρούν και κλιμακώνουν εφαρμογές που βασίζονται σε CPU, μνήμη ή προσαρμοσμένες μετρήσεις. Το Kubernetes είναι χαλαρά συνδεδεμένο και επεκτάσιμο για να καλύπτει διαφορετικούς φόρτους

εργασίας. Αυτή η εκτασιμότητα παρέχεται σε μεγάλο βαθμό από το API του Kubernetes, το οποίο χρησιμοποιείται από εσωτερικά εξαρτήματα(components) καθώς και από επεκτάσεις(extensions) και containers που τρέχουν στο Kubernetes. Η πλατφόρμα ασκεί τον έλεγχο της στους υπολογιστικούς και τους αποθηκευτικούς πόρους με τον ορισμό των πόρων ως Αντικείμενα, τα οποία στη συνέχεια μπορούν να διαχειρίζονται ως τέτοια. Τα βασικά αντικείμενα είναι:

Pods

Η βασική μονάδα προγραμματισμού στο Kubernetes είναι ένα pod. Προσθέτει ένα υψηλότερο επίπεδο αφαίρεσης με την ομαδοποίηση components τα οποία περιέχουν containers. Ένα pod αποτελείται από ένα ή περισσότερους υποδοχείς οι οποίοι είναι εγγυημένο ότι βρίσκονται μαζί στο μηχάνημα υποδοχής (host machine) και μπορούν να μοιράζονται πόρους.

Κάθε pod στο Kubernetes διαθέτει μια μοναδική διεύθυνση Pod IP στο εσωτερικό της συστοιχίας (cluster), το οποίο επιτρέπει στις εφαρμογές να χρησιμοποιούν τις θύρες(ports) χωρίς κίνδυνο σύγκρουσης (conflict). Μέσα στο pod, όλοι οι υποδοχείς μπορούν να αναφέρονται μεταξύ τους στο localhost, αλλά ένας υποδοχέας μέσα σε ένα pod δεν έχει κανέναν τρόπο να απευθύνεται άμεσα σε έναν άλλο υποδοχέα σε άλλο pod, γι 'αυτό, πρέπει να χρησιμοποιήσει τη διεύθυνση Pod IP. Ένας προγραμματιστής εφαρμογών δεν πρέπει ποτέ να χρησιμοποιήσει την διεύθυνση Pod IP, για να αναφερθεί ή να καλέσει μια δυνατότητα σε άλλο pod, καθώς οι διευθύνσεις Pod IP είναι προσωρινές - το συγκεκριμένο pod που αντιστοιχούν μπορεί να αντιστοιχιστεί σε άλλη διεύθυνση Pod IP κατά την επανεκκίνηση. Αντ 'αυτού, θα πρέπει να χρησιμοποιούν μια αναφορά σε μια Υπηρεσία, η οποία περιέχει αναφορά στο target pod στη συγκεκριμένη διεύθυνση Pod IP.

Ένα pod μπορεί να ορίσει έναν τόμο (volume), όπως έναν τοπικό κατάλογο δίσκων ή έναν δίσκο δικτύου, και να το εκθέσει στους υποδοχείς του pod. Τα pods μπορούν να διαχειρίζονται χειροκίνητα μέσω του Kubernetes API ή η διαχείριση τους μπορεί να μεταβιβαστεί σε έναν ελεγκτή (controller). Οι τόμοι αυτοί αποτελούν επίσης τη βάση για τις λειτουργίες του Kubernetes των ConfigMaps (για την παροχή πρόσβασης στη διαμόρφωση μέσω του συστήματος αρχείων που είναι ορατό στον υποδοχέα) και

των Secrets (για την παροχή πρόσβασης σε διαπιστευτήρια που απαιτούνται για ασφαλή πρόσβαση σε απομακρυσμένους πόρους, παρέχοντας αυτά τα διαπιστευτήρια στο σύστημα αρχείων να είναι ορατά μόνο σε εγκεκριμένους υποδοχείς).

Services

Μια υπηρεσία Kubernetes είναι μια σειρά από pods που λειτουργούν μαζί, όπως μία βαθμίδα μιας πολυεπίπεδης εφαρμογής. Το σύνολο των pods που συνιστούν μια υπηρεσία καθορίζονται από έναν επιλογέα ετικετών (label selector). Το Kubernetes παρέχει δύο τρόπους ανίχνευσης των υπηρεσιών (services), χρησιμοποιώντας μεταβλητές περιβάλλοντος (environmental) ή χρησιμοποιώντας το DNS του Kubernetes. Η ανακάλυψη υπηρεσίας ορίζει μια σταθερή διεύθυνση IP και ένα όνομα DNS στην υπηρεσία και η εξισορρόπηση του φορτίου της κυκλοφορίας πετυχαίνεται με τον round-robbin τρόπο για τις δικτυακές συνδέσεις αυτής της διεύθυνσης IP μεταξύ των pods που ταιριάζουν με το label selector (ακόμη και όταν οι αστοχίες προκαλούν τη μετακίνηση των pods από μηχάνημα σε μηχάνημα). Από προεπιλογή, μια υπηρεσία εκτίθεται μέσα σε μια συστοιχία (cluster) (π.χ., τα back-end pods μπορεί να ομαδοποιηθούν σε μια υπηρεσία, ενώ τα αιτήματα από τα front-end pods κάνουν εξισορρόπηση φορτίου μεταξύ τους), αλλά μια υπηρεσία μπορεί επίσης να εκτεθεί έξω από μια συστοιχία (π.χ. για τους πελάτες που θέλουν να προσπελάσουν τα frontend pods).

Volumes

Τα συστήματα αρχείων στο Kubernetes παρέχουν προσωρινή αποθήκευση βραχείας διάρκειας, εξ ορισμού. Αυτό σημαίνει ότι η επανεκκίνηση του υποδοχέα θα εξαλείψει τυχόν δεδομένα σε αυτούς τους υποδοχείς και επομένως αυτή η μορφή αποθήκευσης είναι αρκετά περιοριστική σε οτιδήποτε άλλο παρά σε ασήμαντες εφαρμογές. Ένα Kubernetes volume παρέχει μόνιμη αποθήκευση που υπάρχει σε όλη τη διάρκεια ζωής του ίδιου του pod. Αυτή η αποθήκευση μπορεί επίσης να χρησιμοποιηθεί ως κοινόχρηστος χώρος στο δίσκο για υποδοχείς μέσα στο pod. Οι τόμοι (volumes) τοποθετούνται σε συγκεκριμένα σημεία τοποθέτησης μέσα στον υποδοχέα, τα οποία ορίζονται από τη διαμόρφωση του pod, και δεν μπορούν να τοποθετηθούν σε άλλους

τόμους ή να συνδεθούν με άλλους τόμους. Ο ίδιος τόμος μπορεί να τοποθετηθεί σε διαφορετικά σημεία του δέντρου συστήματος αρχείων από διαφορετικούς υποδοχείς.

Namespaces

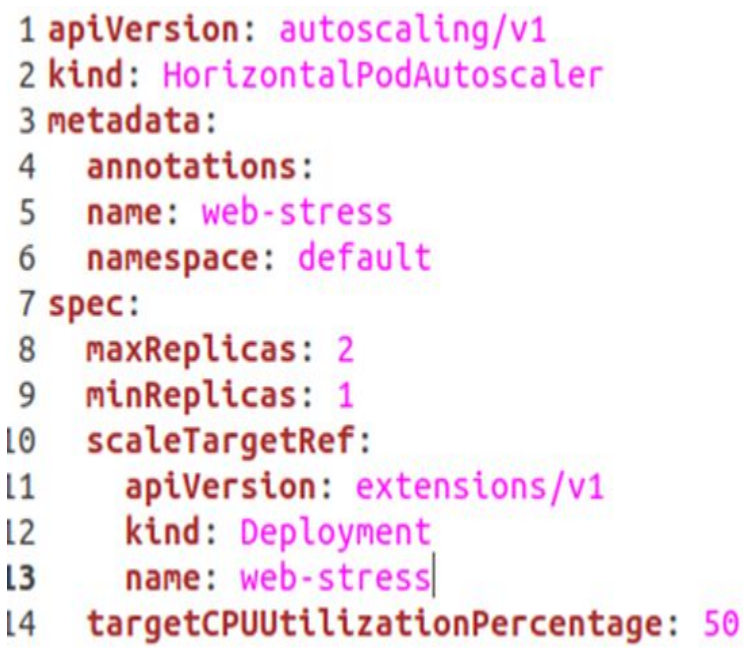
Το Kubernetes παρέχει ένα διαχωρισμό των πόρων, που διαχειρίζεται, σε μη αλληλεπικαλυπτόμενα σύνολα που ονομάζονται χώροι ονομάτων (namespaces). Προορίζονται για χρήση σε περιβάλλοντα με πολλούς χρήστες που διανέμονται σε πολλαπλές ομάδες ή έργα, ή ακόμα και διαχωρίζουν περιβάλλοντα όπως η ανάπτυξη, η δοκιμή και η παραγωγή [8].

3.3 Λεπτομέρειες Υλοποίησης

Στο περιβάλλον του Kubernetes υλοποιήθηκε η κάθετη και η οριζόντια ελαστικότητα μέσω της παραμετροποιήσης που υλοποιήσαμε στο περιβάλλον του και πιο συγκεκριμένα μέσω yamls αρχείων ορίσαμε τι ακριβώς θα γίνεται.

Οριζόντια Ελαστικότητα

Για την οριζόντια ελαστικότητα εφαρμόσαμε ένα ενδεικτικό yaml αρχείο όπως το παρακάτω.



Εικόνα 5 - yaml Οριζόντιας ελαστικότητας παραμετροποίηση CPU

Όπου στο παραπάνω αρχείο περιγράφεται η λειτουργία του Horizontal Pod Autoscaler (HPA) και πιο συγκεκριμένα δηλώνονται οι ελάχιστες ρεπλικες που στο συγκεκριμένο παράδειγμα είναι 1 και οι μέγιστες ρεπλικες που είναι 2. Επίσης ορίζεται το όριο εκείνο με το οποίο θα αυξάνονται οι ρεπλικες που ορίζεται η χρήση του επεξεργαστή στο μισό της ισχύος του (50%). Ακόμη ορίζεται η εφαρμογή στην οποία στοχεύουμε να κάνουμε κλιμάκωση δηλαδή ελαστικότητα όπου στο συγκεκριμένο παράδειγμα είναι η web-stress.

Για την παραμετροποίηση του περιβάλλοντος του Kubernetes στις περιπτώσεις εκείνες που ασχολούμασταν με το δίκτυο εφαρμόσαμε ένα ενδεικτικό yaml αρχείο όπως το παρακάτω.

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  annotations:
  name: hpa-network
  namespace: default
spec:
  maxReplicas: 10
  minReplicas: 1
  scaleTargetRef:
    apiVersion: extensions/v1
    kind: Deployment
    name: web-stress
  metrics:
  - type: Object
    object:
        target:
        kind: Service
        name: my-service11
        metricName: http_request
        targetValue: 100
  - type: Pods
    pods:
       metricName: packets-per-second
       targetValue: 100k
```

Εικόνα 6 - yaml Οριζόντιας ελαστικότητας παραμετροποίηση δικτύου

Όπου το παραπάνω αρχείο περιγράφει την παραμετροποίηση του περιβάλλοντός μας και πιο συγκεκριμένα αφού καθορίσει πάλι τις μέγιστες ρεπλικες και τις ελάχιστες στη συνέχεια περιγράφει αντίστοιχα όπως το προηγούμενο yaml τα όρια εκείνα με τα οποία θα κλιμακώνονται τα αντίγραφα και βλέπουμε ότι καθορίζονται διάφορες μετρικές για αυτόν τον σκοπό που έχουν να κάνουν με τα αιτήματα και τα πακέτα ανά δευτερόλεπτο.

Κάθετη Ελαστικότητα

Για την κάθετη ελαστικότητα στο περιβάλλον του Kubernetes εφαρμόσαμε ένα ενδεικτικό yaml αρχείο όπως το παρακάτω.

```yaml
apiVersion: autoscaling.k8s.io/v1beta2
kind: VerticalPodAutoscaler
metadata:
  name: my-vpa
spec:
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: web-stress
  updatePolicy:
    updateMode: "Auto"
```

Εικόνα 7 - yaml κάθετης ελαστικότητας

Όπου στο παραπάνω αρχείο καθορίζεται η εφαρμογή στην οποία στοχεύουμε να εφαρμόσουμε την κάθετη ελαστικότητα, στο συγκεκριμένο παράδειγμα η web-stress. Επίσης ορίζεται η πολιτική ανανέωσης των ρυθμίσεων, όπου στο συγκεκριμένο παράδειγμα είναι αυτόματη.

4 Πειραματικά Αποτελέσματα

Ο στόχος της εργασίας μας είναι η συγκριτική πειραματική ανάλυση εναλλακτικών επιλογών ενορχήστρωσης, σε διαφορετικά πειραματικά σενάρια με σκοπό την αποδοτικότερη αξιοποίηση των υπολογιστικών και δικτυακών πόρων των διακομιστών σε κέντρα δεδομένων αλλά και την βελτιστοποίηση της απόδοσης των παρεχόμενων υπηρεσιών. Πιο συγκεκριμένα θέλουμε να δούμε πως ανταποκρίνονται οι διάφοροι μηχανισμοί ελαστικότητας σε ένα υπολογιστικό περιβάλλον και να αναδειχθούν με αυτόν τον τρόπο τα υπέρ και τα κατά του κάθε μηχανισμού.

Προκειμένου να αναδείξουμε διαφορετικές πλευρές του προβλήματος, ορίσαμε τέσσερα διαφορετικά σενάρια, όπου το καθένα από αυτά στοχεύει στην απάντηση διαφορετικών επιστημονικών ερωτημάτων.

Πιο συγκεκριμένα, στόχος του πρώτου σεναρίου είναι να μελετήσουμε πως αποδίδει ο κάθε μηχανισμός ελαστικότητας σε χρονικά διαστήματα. Για να το πετύχουμε αυτό πραγματοποιήσαμε πειράματα με διαφορετικά χρονικά διαστήματα για μια συγκεκριμένη εφαρμογή εγκατεστημένη στο περιβάλλον του Kubernetes μετρώντας τον αριθμό των επιτυχημένων αιτημάτων.

Στόχος του δεύτερου σεναρίου είναι η αξιολόγηση των μηχανισμών ελαστικότητας σε διαφορετικού τύπου εφαρμογές στην βαθμιαία αύξηση του χρόνου. Οι τρεις εφαρμογές που εφαρμόσαμε στο περιβάλλον του Kubernetes είναι διαφορετικού τύπου δηλαδή CPU, Memory και Network intensive και στοχεύσαμε να παρατηρήσουμε το πως λειτουργούν οι διάφοροι μηχανισμοί ελαστικότητας σε αυτές τις εφαρμογές.

Στο τρίτο σενάριο επιχειρούμε να παρατηρήσουμε πως ανταποκρίνονται οι διάφοροι μηχανισμοί ελαστικότητας στην βαθμιαία αύξηση των πελατών και να αναλύσουμε την απόδοση των αλγορίθμων. Αυτό που υλοποιήσαμε στο συγκεκριμένο σενάριο ήταν να αυξάνουμε βαθμιαία τους πελάτες που αιτούνται από την εφαρμογή, την CPU intensive εφαρμογή, και να παρατηρούμε την απόδοση των μηχανισμών.

Στο τέταρτο σενάριο επιδιώκουμε να καταγράψουμε πως ανταποκρίνονται οι διάφοροι μηχανισμοί ελαστικότητας στην απότομη αύξηση του αριθμού των πελατών σε συγκεκριμένη εφαρμογή με σταθερό τον χρόνο. Αυτό που πραγματοποιήσαμε στο συγκεκριμένο πειραματικό σενάριο ήταν να δημιουργήσουμε ένα εκτελέσιμο αρχείο το

οποίο δημιουργούσε ξαφνικά στη μέση του πειράματος παραπάνω πελάτες άρα και παραπάνω requests στην εφαρμογή μας. Ο στόχος του πειράματος είναι να παρατηρήσουμε την ανταπόκριση κάθε μηχανισμού σε αυτήν την ξαφνική αύξηση.

Η μελέτη μας αυτή μας οδήγησε σε διάφορα συμπεράσματα τα οποία κατατίθενται στο τέλος του κεφαλαίου.

Στη συνέχεια του κεφαλαίου θα γίνει η εισαγωγή στην περιγραφή των πειραματικών σεναρίων τα οποία υλοποιήσαμε, τα ειδικά συμπεράσματα που προκύπτουν από το καθένα και στο τέλος θα κλείσουμε με τα γενικά συμπεράσματα που προκύπτουν.

Ερωτήματα που καλείται να απαντήσει το παρόν κεφάλαιο:

- Είναι πιο αποδοτικός μηχανισμός ο VPA ή ο HPA και σε ποιές περιπτώσεις χρήσης;
- Υπάρχουν κάποια όρια στη χρησιμοποίηση τους και αν ναι, ποιά;

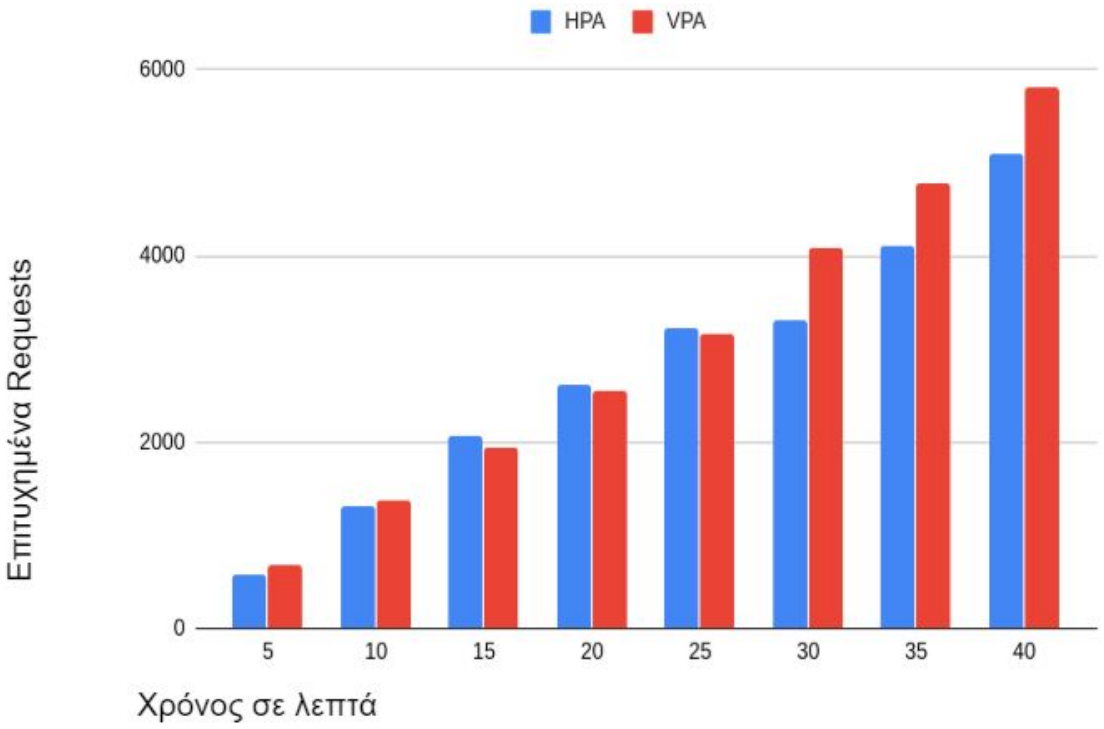
4.1 Σενάριο Α – Πειράματα με διαφορετικό χρόνο εκτέλεσης

Σε αυτό το πειραματικό σενάριο εφαρμόσαμε την περίπτωση χρήσης η οποία περιγράφεται ως εξής: Πως αποδίδει ο κάθε μηχανισμός ελαστικότητας σε πειράματα με διαφορετικό χρόνο εκτέλεσης. Κατα την διαρκεια των πειραμάτων πραγματοποιήσαμε προσομοίωση του φόρτου εργασίας με την χρήση του εργαλείου συγκριτικής αξιολόγησης ‘Apache Benchmark’, εκτελώντας αιτήματα προς την απαιτητική σε CPU (CPU-intensive) παρεχόμενη υπηρεσία. Αξιολογήσαμε πως αποδίδει ο κάθε μηχανισμός ελαστικότητας στην περίπτωση που έχουμε μια εφαρμογή εγκατεστημένη στο περιβάλλον του Kubernetes. Ως μέτρο για την επίδοση των μηχανισμών πήραμε των αριθμό των αιτημάτων που ολοκληρώθηκαν σε συγκεκριμένο χρονικό διάστημα (δλδ, χρόνος εκτέλεσης πειραμάτων).

Στο πειραματικό περιβάλλον της εργασίας δημιουργήσαμε ένα Deployment στο Kubernetes με ένα container image, το οποίο δημιουργεί μία εφαρμογή cpu-intensive η οποία όταν δέχεται αιτήματα πραγματοποιεί έναν επαναλαμβανόμενο υπολογισμό με

σκοπο να αυξήσει το φόρτο του επεξεργαστή. Με αυτόν τον τρόπο θέλαμε να δούμε πως ο κάθε μηχανισμός θα ανταποκριθεί στη βαθμιαία αύξηση του χρόνου των πειραμάτων.

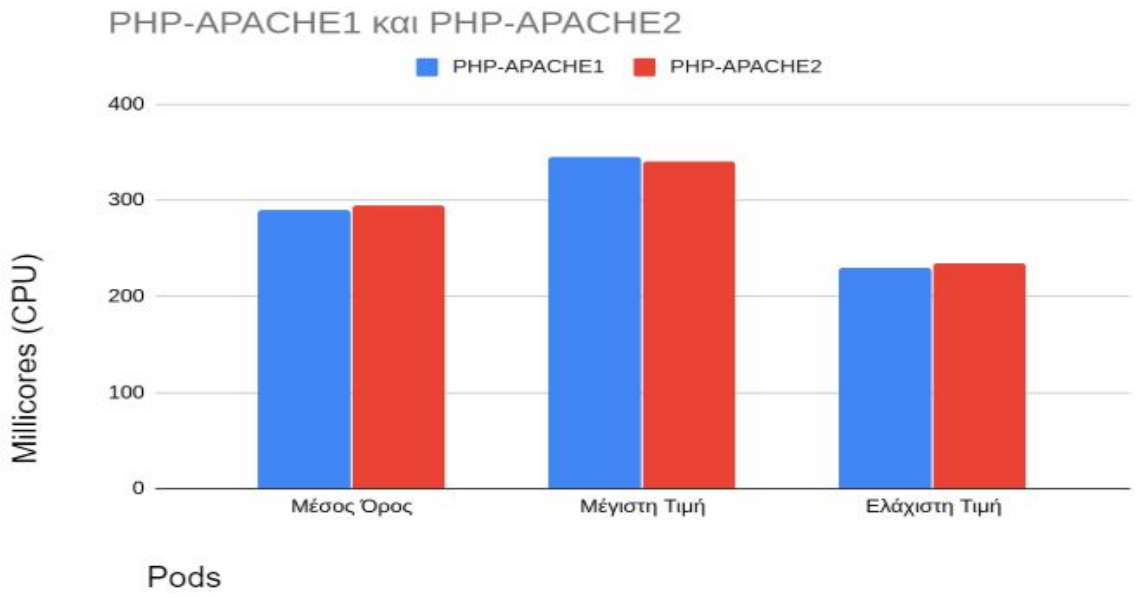
Τα αποτελέσματα των πειραμάτων παρατίθενται παρακάτω (όπου στον άξονα x έχουμε τον χρόνο των πειραμάτων σε λεπτά και στον άξονα y των αριθμό των ολοκληρωμένων αιτημάτων):



Γράφημα 1 - Αποτελέσματα πειραμάτων σενάριο Α

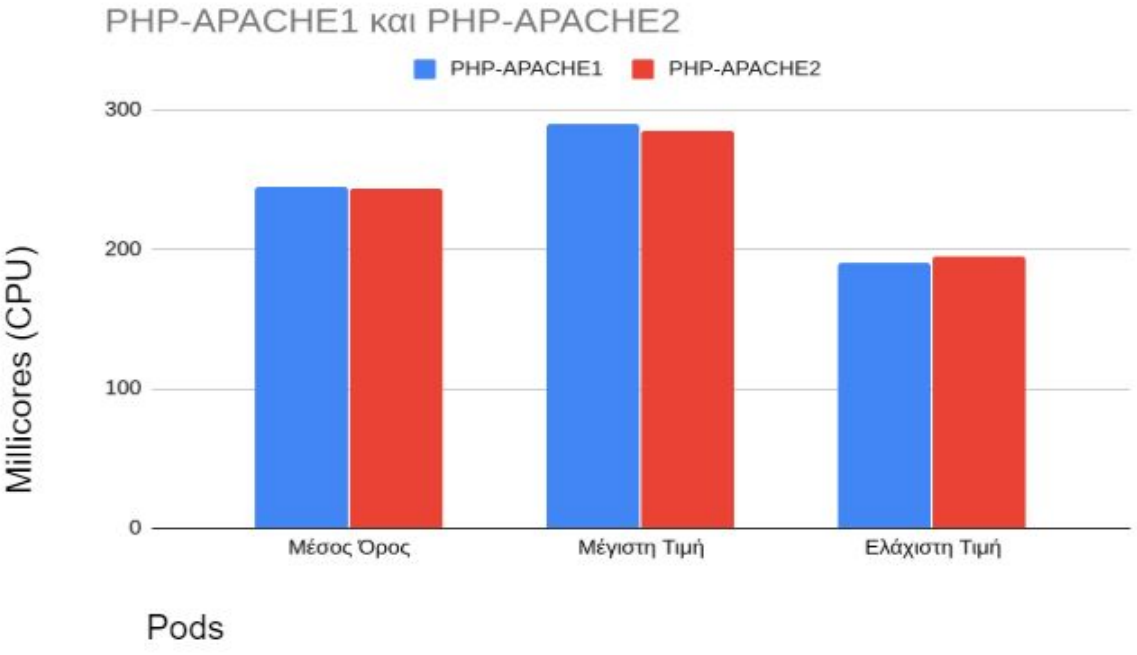
Μετά παραθέτουμε τις μετρήσεις που πάρθηκαν από κάθε Pod για την χρησιμοποίηση της CPU τους:

1. Μηχανισμός VPA:



Γράφημα 2 - Αποτελέσματα Pods για VPA σενάριο Α

2. Μηχανισμός HPA:



Γράφημα 3 - Αποτελέσματα Pods για HPA σενάριο Α

Απο τα αποτελέσματα συμπένουμε:

1. Τα 25 πρώτα λεπτά παρατηρούμε να αποδίδει ελαφρώς καλύτερα ο μηχανισμός HPA, καθως όπως παρατηρούμε εξυπηρετήθηκαν περίπου 300 αιτήματα περισσότερα σε σύγκριση με τα ελαφρώς λιγότερα του μηχανισμού VPA.
2. Από τα 25 λεπτά και έπειτα παρατηρούμε να λειτουργεί πιο αποδοτικά ο μηχανισμός VPA.

Ο VPA αυξομειώνει την διαθεσιμότητα των υπολογιστικών πόρων (συγκεκριμένα στο πειραμά μας την CPU) των PODs ανάλογα με το ποσοστό κατανάλωσης τους. Απο την αλλη πλευρα, ο HPA αυξομειώνει τα αντίγραφα των PODs άλογα με το ποσοστό κατανάλωσης των πόρων τους.

Σύμφωνα με τα αποτελέσματα αυτού του σεναρίου συμπεραίνουμε ότι ο μηχανισμός VPA αποδίδει καλύτερα μετά από ένα χρονικό διάστημα (περίπτωση 1). Παρόλο που και οι δύο μηχανισμοί είναι δυναμικοί δλδ. παρακολουθούν σε πραγματικό χρόνο την κατανάλωση των επεξεργαστικών πόρων (resources) ο VPA χρειάζεται περισσότερο χρόνο για την λήψη απόφασης από τον HPA. Συνεπώς, ο HPA λειτουργεί καλύτερα σε μικρά χρονικά διαστήματα. Σε μεγάλης διάρκειας πειράματα λειτουργεί αρκετά καλύτερα ο VPA, ο οποίος μέσω του σύνθετου μηχανισμού λειτουργίας που έχει, μπορεί να καθυστερεί να αποδώσει στην αρχή αλλά στη συνέχεια προσαρμόζεται αποδοτικότερα στις ανάγκες που υπάρχουν και προσθέτει με δυναμικό τρόπο ευελιξία στις απαιτήσεις.

4.2 Σενάριο Β – Βαθμιαία αύξηση του χρόνου για διαφορετικές εφαρμογές

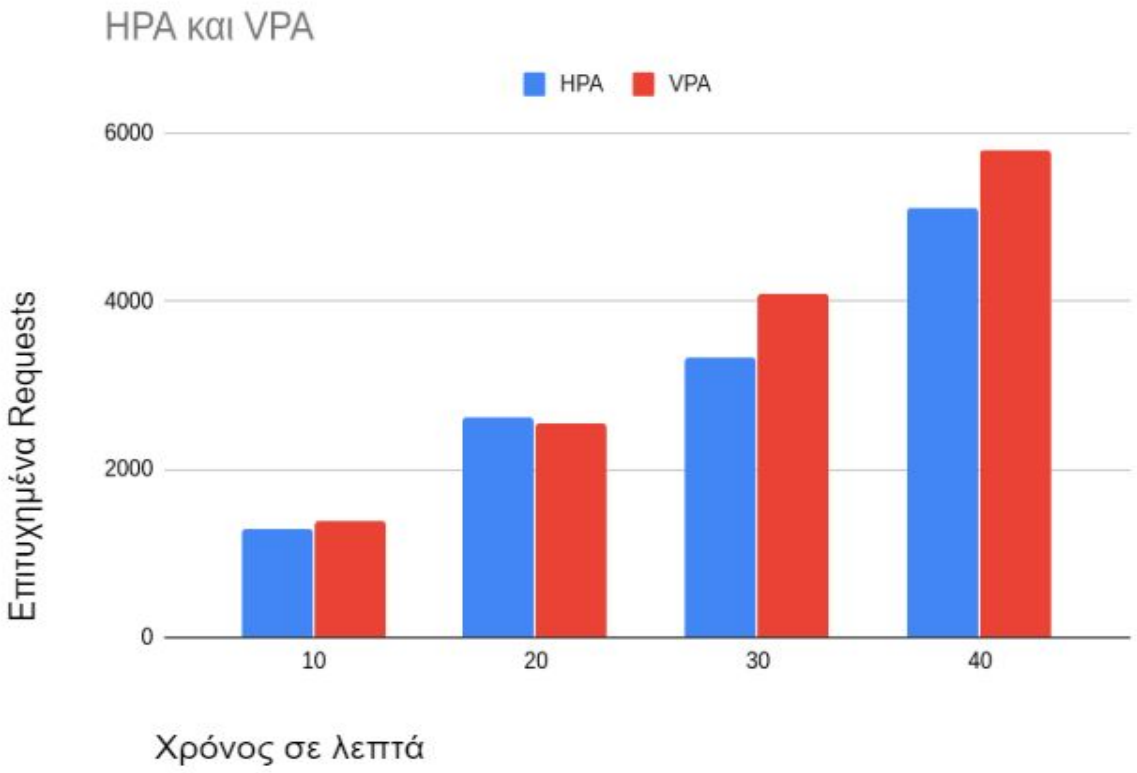
Στο συγκεκριμένο πειραματικό σενάριο προσομοιώσαμε την περίπτωση χρήσης η οποία απαντά στο εξής: Πως ανταποκρίνονται οι μηχανισμοί ελαστικότητας στην περίπτωση διαφορετικών εφαρμογών, με διαφορετικές απαιτήσεις σε πόρους, σε πειράματα με διαφορετικό χρόνο εκτέλεσης;

Αυτό που πραγματοποιήσαμε στο συγκεκριμένο σενάριο ήταν να δημιουργήσουμε 3 Deployments στο περιβάλλον του Kubernetes με το κάθε Deployment και από μία ξεχωριστή εφαρμογή, ένα container image ξεχωριστό δηλαδή. Το πρώτο

Deployment είναι η εφαρμογή του πρώτου μας σεναρίου και τα υπόλοιπα 2 είναι: το πρώτο μία εφαρμογή CPU και Memory intensive και το τελευταίο μία εφαρμογή Network intensive.

Τα αποτελέσματα των πειραμάτων παρατίθενται παρακάτω:

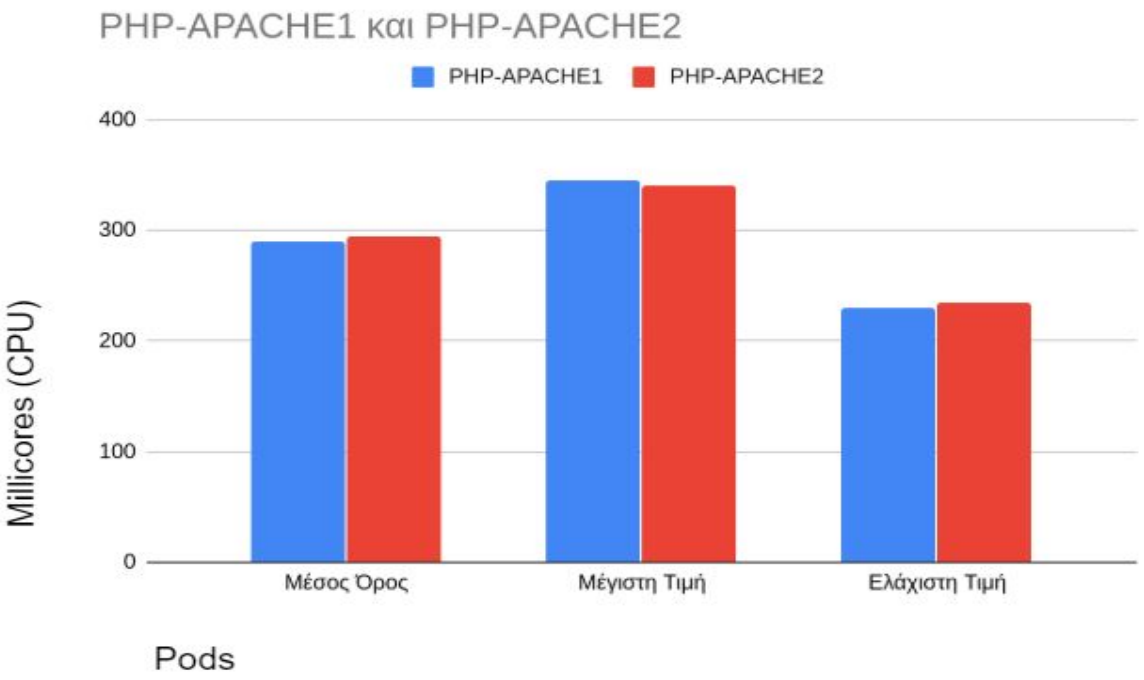
1) Εφαρμογή απαιτητική σε κατανάλωση CPU (CPU intensive):



Γράφημα 4 - Αποτελέσματα πειραμάτων σενάριο Β.1

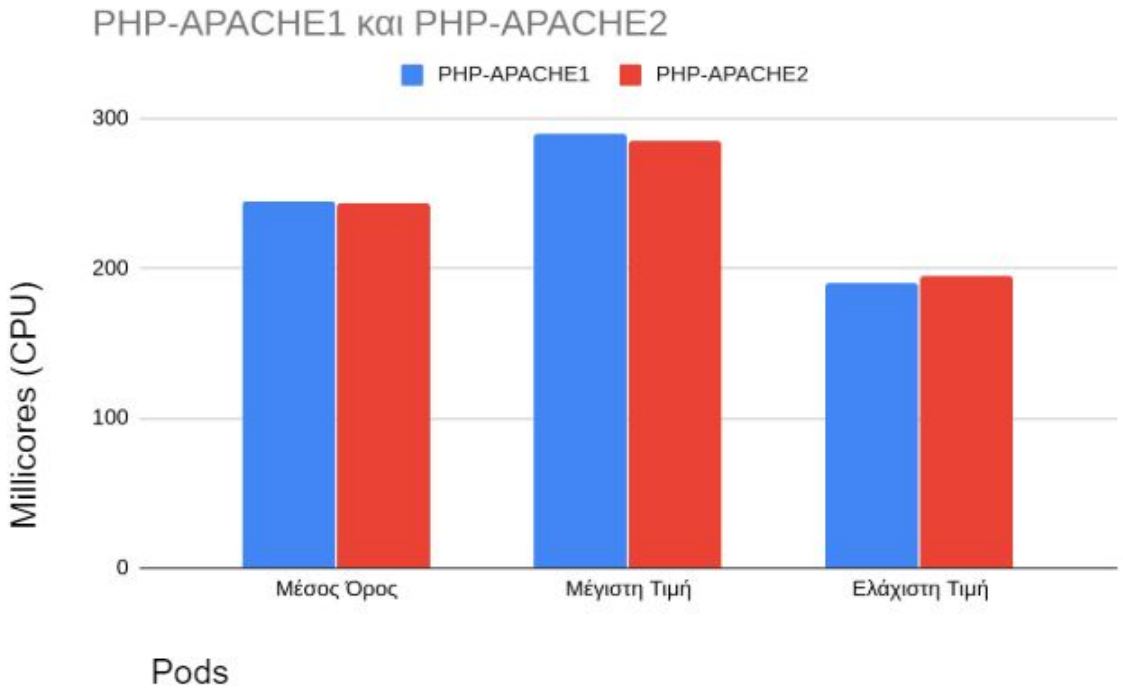
Μετά παραθέτουμε τις μετρήσεις που πάρθηκαν από κάθε Pod για την χρησιμοποίηση της CPU τους:

1. Μηχανισμός VPA:



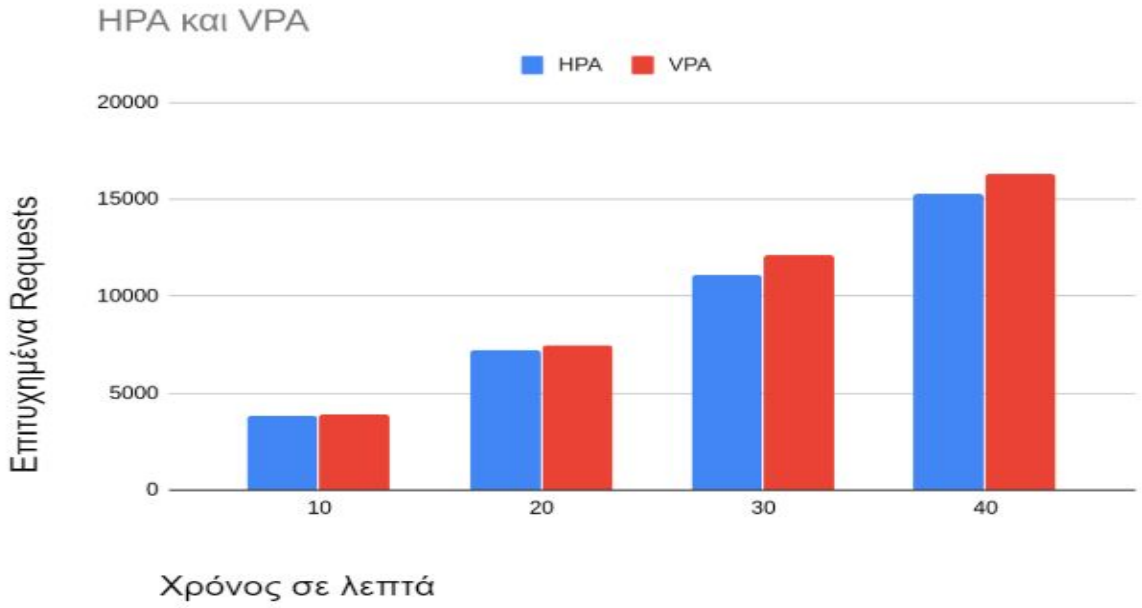
Γράφημα 5 - Αποτελέσματα Pods για VPA σενάριο B.1

2. Μηχανισμός HPA:



Γράφημα 6 - Αποτελέσματα Pods για HPA σενάριο B.1

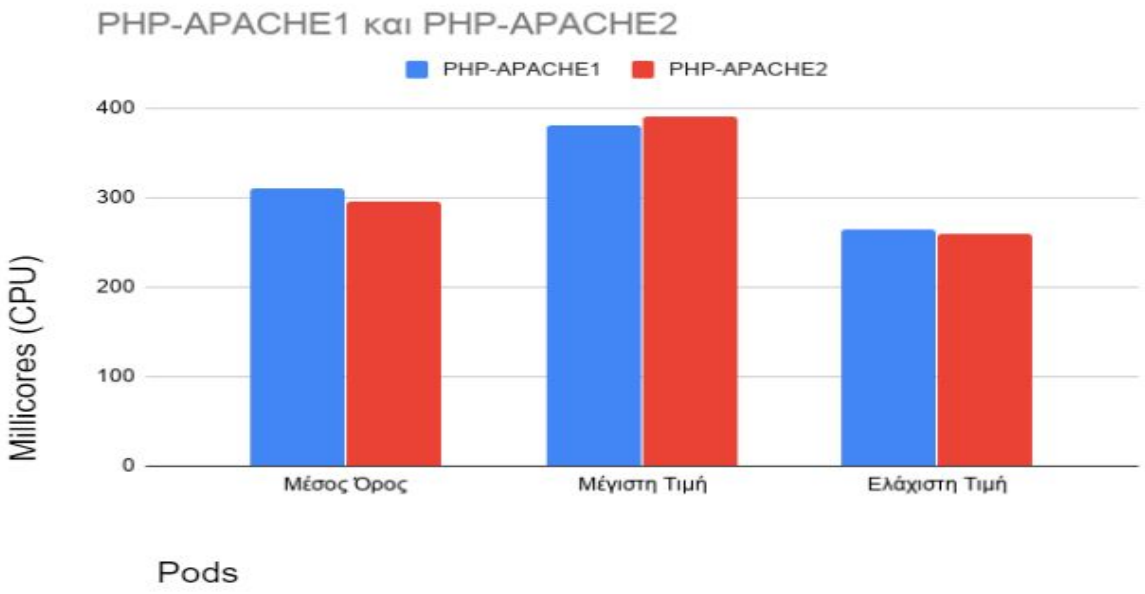
2) Εφαρμογή CPU and memory intensive:



Γράφημα 7 - Αποτελέσματα πειραμάτων σενάριο B.2 cpu & memory

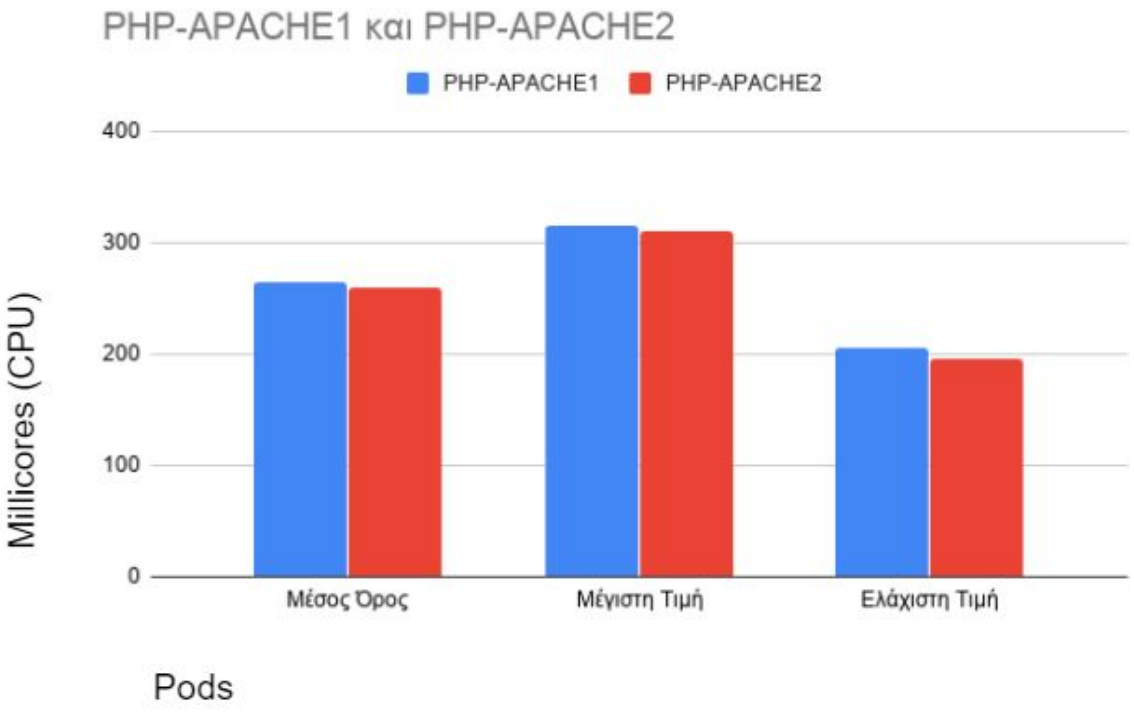
Μετά παραθέτουμε τις μετρήσεις που πάρθηκαν από κάθε Pod για την χρησιμοποίηση της CPU τους:

1. Μηχανισμός VPA:



Γράφημα 8 - Αποτελέσματα Pods για VPA σενάριο B.2 cpu

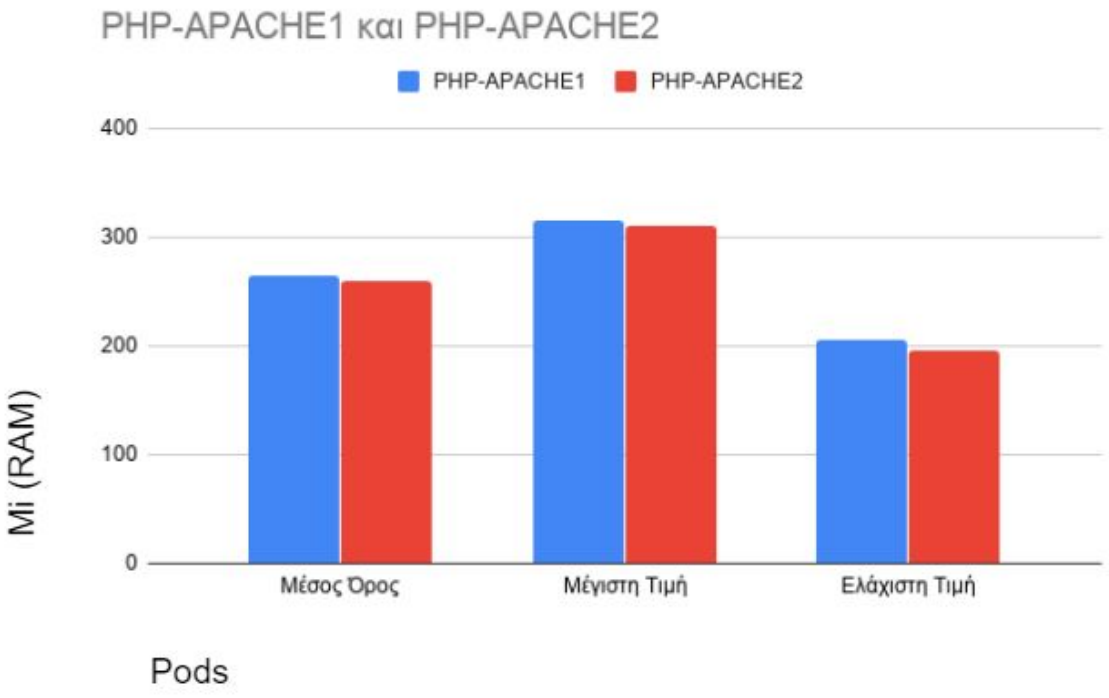
2. Μηχανισμός HPA:



Γράφημα 9 - Αποτελέσματα Pods για HPA σενάριο B.2 cpu

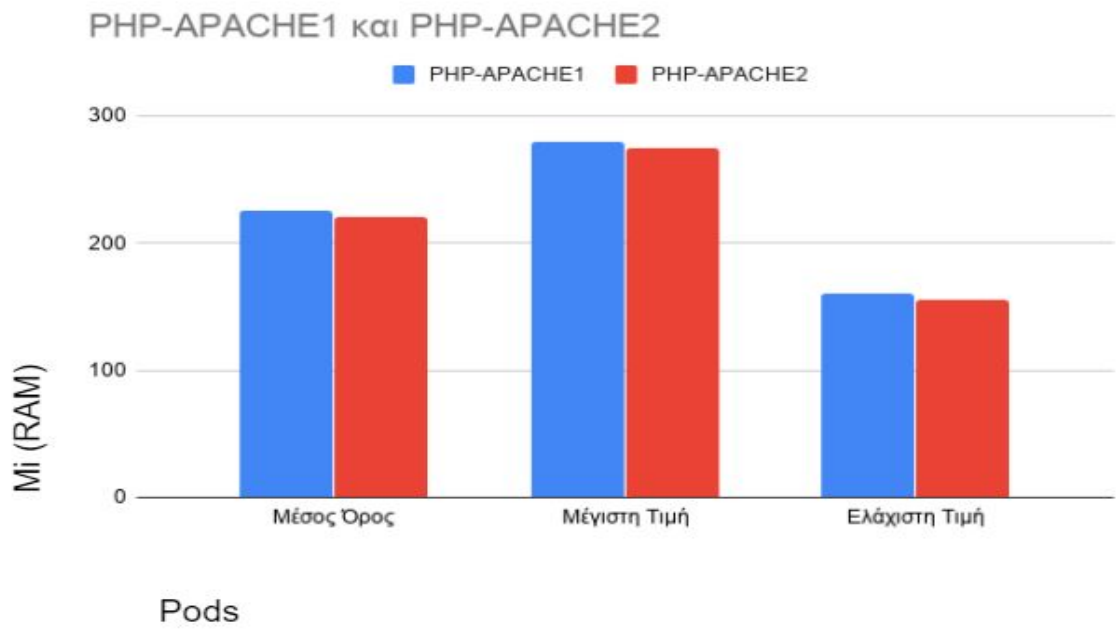
Και μετρήσεις για τη Memory τους:

1. Μηχανισμός VPA:



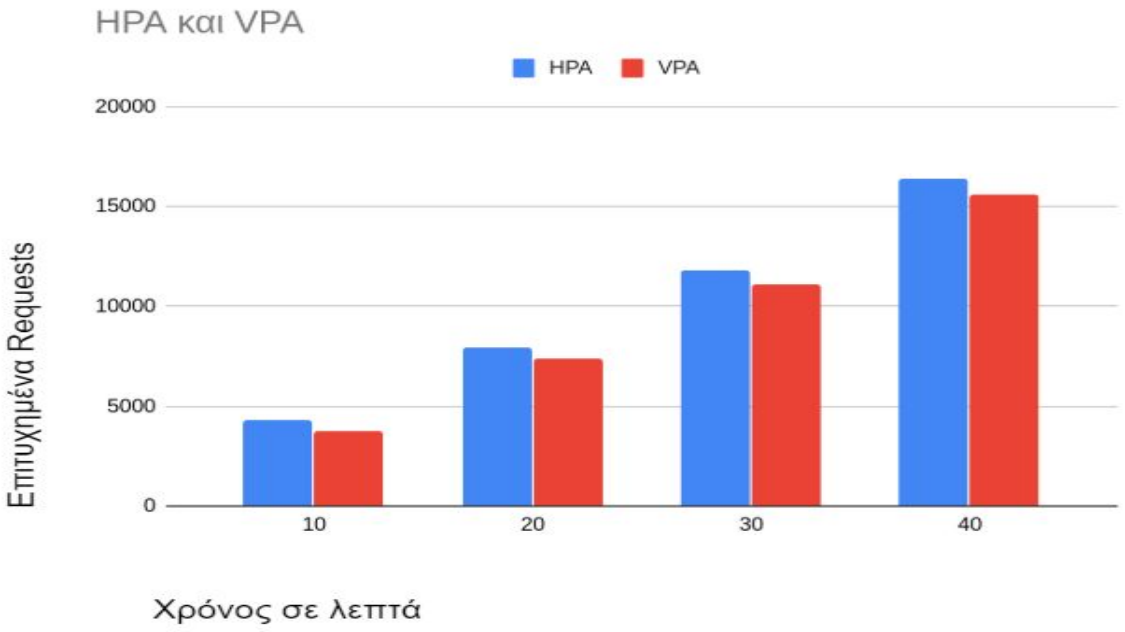
Γράφημα 10 - Αποτελέσματα Pods για VPA σενάριο B.2 memory

2. Μηχανισμός HPA:



Γράφημα 11 - Αποτελέσματα Pods για HPA σενάριο B.2 memory

3) Εφαρμογή απαιτητική σε εύρος ζώνης (Bandwidth intensive):



Γράφημα 12 - Αποτελέσματα πειραμάτων σενάριο B.2 network

Τα συμπεράσματα που προκύπτουν από την πραγματοποίηση αυτών των πειραμάτων μας δείχνουν ότι:

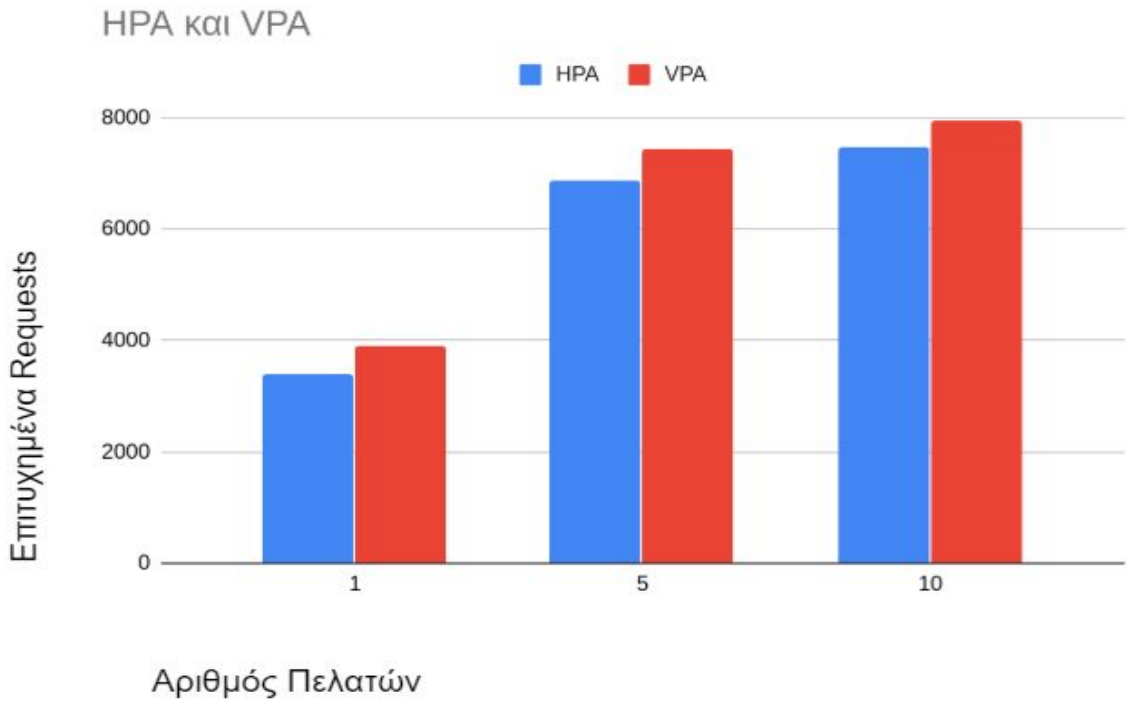
1. Στην περίπτωση 1 και 2 επιβεβαιώνονται τα αποτελεσματα του πρώτου σεναρίου. Δηλαδη και για την CPU και για την RAM εφαρμογή συνεχίζει να αποδίδει καλύτερα ο μηχανισμός VPA μετά την πάροδο ενός χρονικού διαστήματος.
2. Στην εφαρμογή που είναι απαιτητική σε εύρος ζώνης (bandwidth) θεωρούμε ότι τα αποτελέσματα δεν αντιπροσωπευτικά για την αξιολόγηση των μηχανισμών καθως VPA περιορίζεται από κατασκευής του να κοιτάει μόνο CPU & memory, και δεν έχει αναπτυχθεί ακόμη ώστε να κοιτάει δίκτυο. Αντιθέτως ο μηχανισμός HPA μπορεί να παραμετροποιηθεί και να λειτουργεί αντίστοιχα με βάση τον φόρτο που έχουμε για το δίκτυο.

4.3 Σενάριο Γ – Βαθμιαία αύξηση των αιτημάτων προς συγκεκριμένη εφαρμογή με σταθερό τον χρόνο

Στο συγκεκριμένο σενάριο προσομοιώσαμε την περίπτωση χρήσης, η οποία περιγράφεται ως εξής: Πως ανταποκρίνονται οι μηχανισμοί ελαστικότητας στην βαθμιαία αύξηση των πελατών;

Στο συγκεκριμένο σενάριο αξιολογήσαμε την αποδοση των μηχανισμων ελαστικοτητας ως προς τον αριθμό των αιτημάτων σε συγκεκριμένο χρονικό διάστημα. Δλδ, αυξήσαμε βαθμιαία τα αιτήματα προς CPU intensive εφαρμογή, παρατηρήσαμε την απόδοση των μηχανισμών με σταθερό χρόνο πειράματος 30 λεπτά

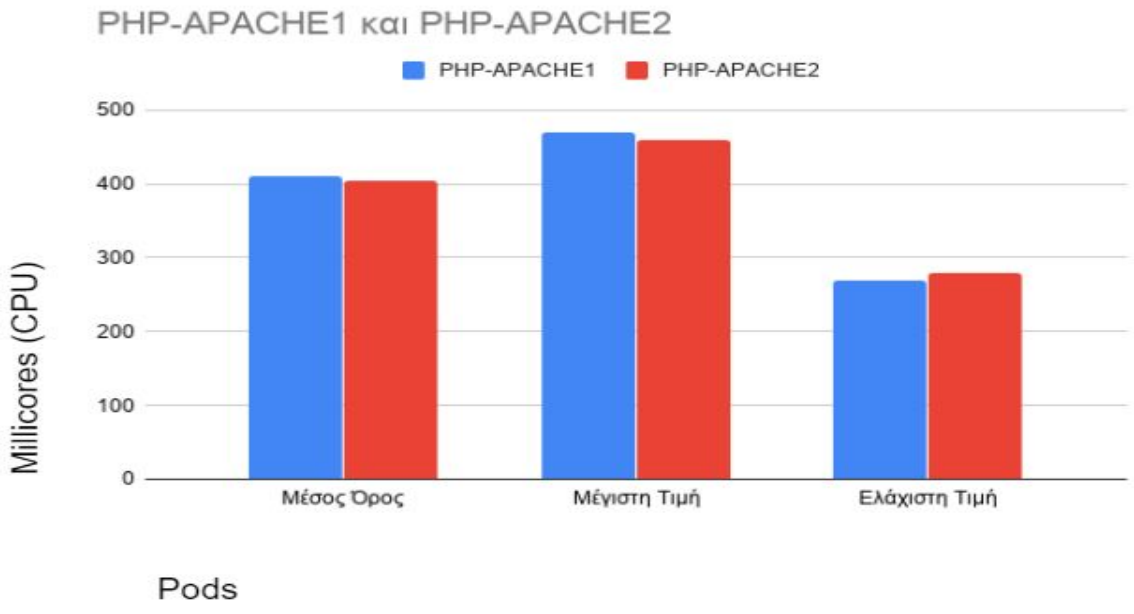
Τα αποτελέσματα των πειραμάτων φαίνονται παρακάτω (όπου στον άξονα x είναι οι πελάτες και στον άξονα y ο αριθμός των επιτυχημένων requests):



Γράφημα 13 - Αποτελέσματα πειραμάτων σενάριο Γ

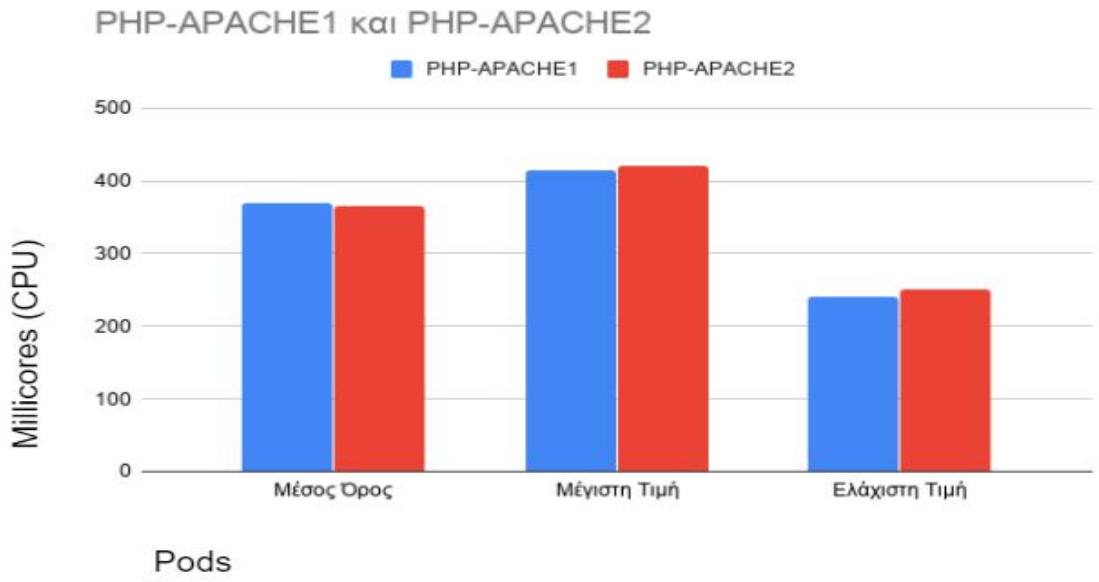
Μετά παραθέτουμε τις μετρήσεις που πάρθηκαν από κάθε Pod για την χρησιμοποίηση της CPU τους:

1. Μηχανισμός VPA:



Γράφημα 14 - Αποτελέσματα Pods για VPA σενάριο Γ

2. Μηχανισμός HPA:



Γράφημα 15 - Αποτελέσματα Pods για HPA σενάριο Γ

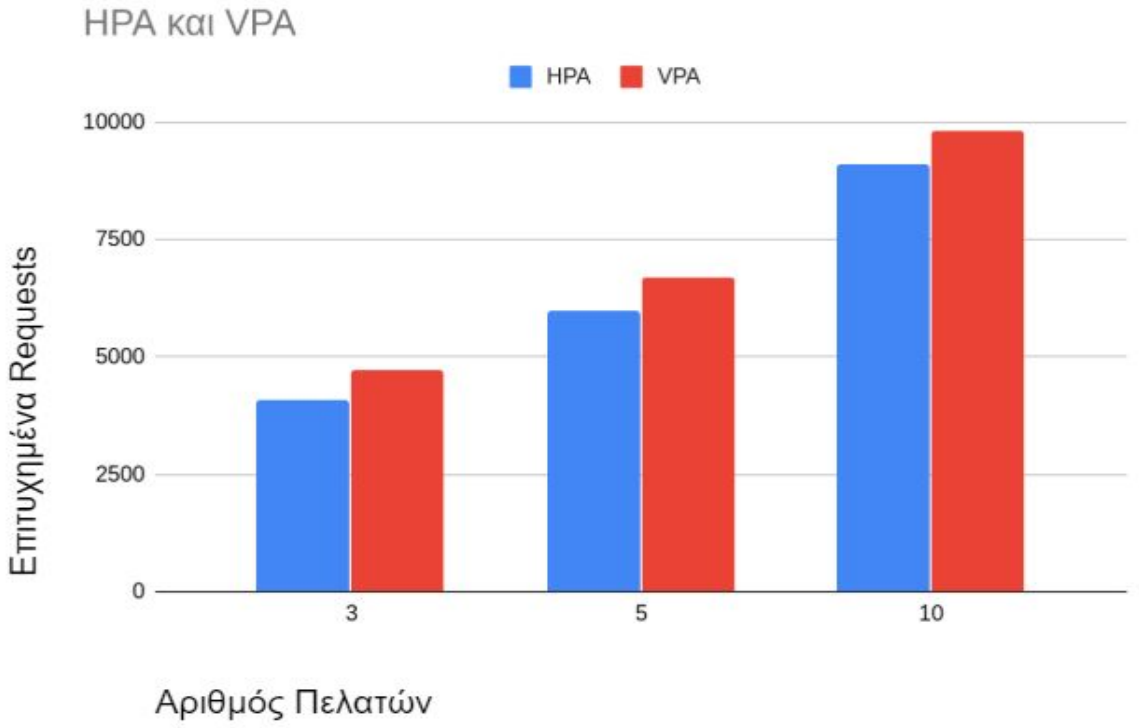
Συμπεραίνουμε ότι ο μηχανισμός VPA παραμένει και σε αυτή την περίπτωση χρήσης καλύτερος σε απόδοση σε σχέση με τον μηχανισμό HPA. Αυτό μπορεί να αιτιολογηθεί, από τις διαπιστώσεις που έχουμε κάνει στα προηγούμενα σενάρια σχετικά με τρόπο λειτουργίας των μηχανισμών.

4.4 Σενάριο Δ – Απότομη αύξηση των πελατών με σταθερό τον χρόνο σε συγκεκριμένη εφαρμογή

Σε αυτό το πειραματικό σενάριο προσομοιώσαμε εκείνη την περίπτωση χρήσης που διατυπώνεται ως εξής: Πως ανταποκρίνονται οι μηχανισμοί ελαστικότητας στην απότομη αύξηση του αριθμού των πελατών σε συγκεκριμένη εφαρμογή με σταθερό χρόνο;

Αυτό που πραγματοποιήσαμε στο συγκεκριμένο πειραματικό σενάριο ήταν να δημιουργήσουμε ένα εκτελέσιμο αρχείο το οποίο δημιουργούσε ξαφνικά στη μέση του πειράματος παραπάνω πελάτες άρα και παραπάνω requests στην εφαρμογή μας. Ο στόχος του πειράματος είναι να παρατηρήσουμε την ανταπόκριση κάθε μηχανισμού σε αυτήν την ξαφνική αύξηση.

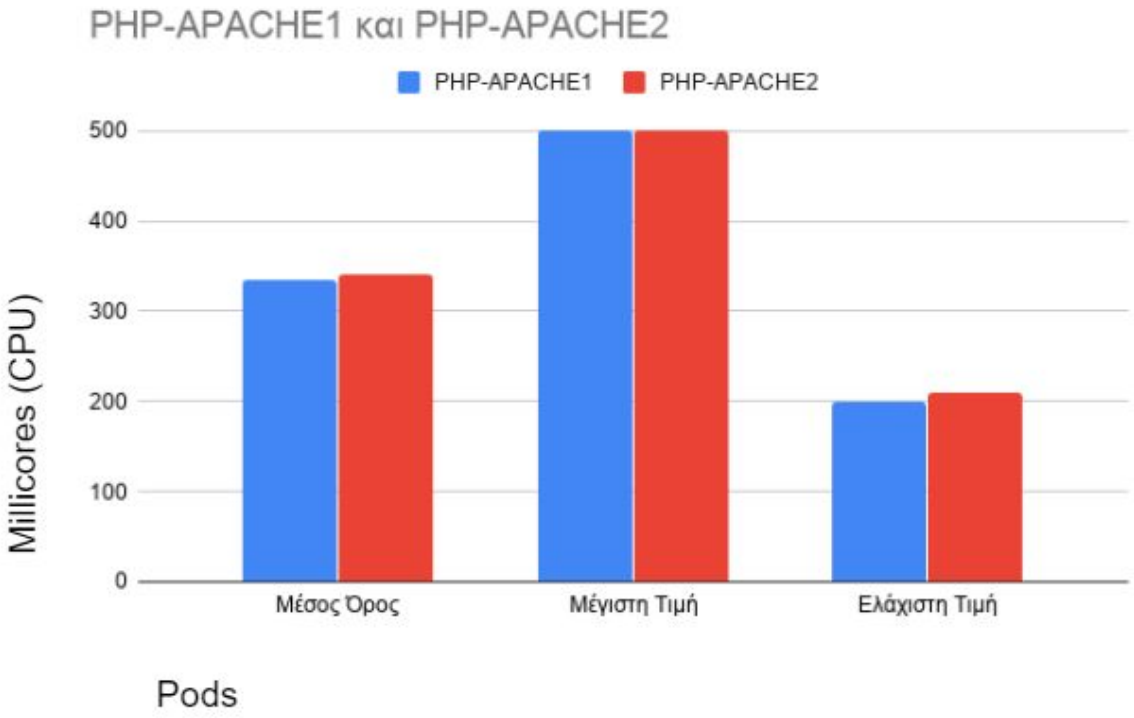
Τα αποτελέσματα των μετρήσεων του πειράματος φαίνονται παρακάτω (όπου στον άξονα του x είναι ο αριθμός των πελατών που αυξάνονται και στον άξονα των y ο αριθμός των επιτυχημένων requests):



Γράφημα 16 - Αποτελέσματα πειραμάτων σενάριο Δ

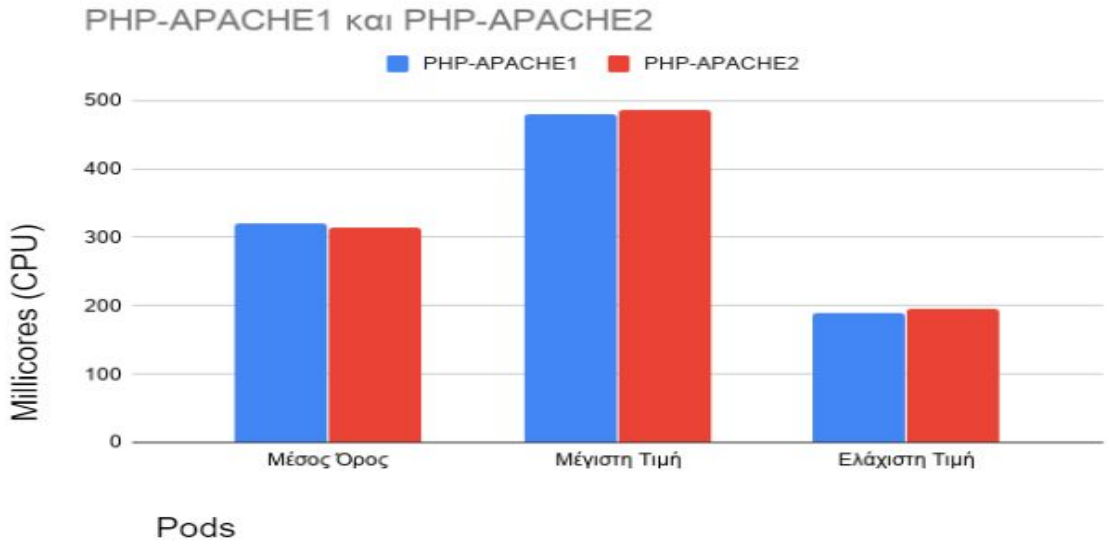
Μετά παραθέτουμε τις μετρήσεις που πάρθηκαν από κάθε Pod για την χρησιμοποίηση της CPU τους:

1. Μηχανισμός VPA:



Γράφημα 17 - Αποτελέσματα Pods για VPA σενάριο Δ

2. Μηχανισμός HPA:



Γράφημα 18 - Αποτελέσματα Pods για HPA σενάριο Δ

Αυτό στο οποίο μπορούμε να οδηγηθούμε ως συμπέρασμα είναι ότι: ο μηχανισμός VPA παραμένει πιο αποδοτικός σε σχέση με τα άλλα σενάρια, αν και πέφτει σχετικά η διαφορά της απόδοσής του σε σχέση με τα προηγούμενα σενάρια καθώς του πέρνει λίγο χρονικό διάστημα ώστε να αντιληφθεί τις αλλαγές στα αιτήματα και την αύξησή τους.

4.5 Συμπεράσματα

Τα συμπεράσματα που προκύπτουν από τα πειραματικά σενάρια μπορούμε να τα τοποθετήσουμε με βάση τα ερωτήματα τα οποία τέθηκαν στην αρχή του κεφαλαίου και τα οποία είναι:

1. Ο μηχανισμός VPA αποδίδει καλύτερα σε σχέση με τα άλλα σενάρια, ειδικά με την πάροδο ενός χρονικού διαστήματος το οποίο χρειάζεται με βάση την λειτουργία του για να αποδώσει.
2. Τα όρια που έχει ο κάθε μηχανισμός είναι για το VPA μόνο σε εφαρμογές που έχουν να κάνουν με CPU και memory λειτουργίες και ως εκ τούτου περιορίζεται μόνο στις συγκεκριμένες εφαρμογές, ενώ αντιθέτως ο HPA μπορεί να παραμετροποιηθεί και παραπάνω και σε εφαρμογές που έχουν να κάνουν με δικτυακό φόρτο.

Σύμφωνα με τα αποτελεσματα της πειραματικής αξιολόγησης των μηχανισμων ελαστικοτητας μπορουμε να προχωρήσουμε στις ακόλουθες προτάσεις για την χρήση τους.

- Σε περίπτωση που έχουμε υποδοχείς χωρίς περιορισμό στην κατανάλωση των επεξεργαστικών πόρων και θέλουμε να πραγματοποιήσουμε ελαστικότητα ανάμεσα σε πραγματικά μηχανηματα αναποφευκτα ο HPA είναι καλύτερος.
- Στην περιπτωση που εχουμε υποδοχείς με αρχικό περιορισμό στην κατανάλωση των επεξεργαστικών πόρων (με σκοπο την αποδοτικότερη διαχείριση τους) είναι καλύτερο να χρησιμοποιηθει ο VPA δηλαδη να αυξηθεί το αρχικό όριο της κατανάλωσης του υποδοχέα απο το ανοίξει ένας καινούριος σε ένα άλλο μηχάνημα.
- Ως καλύτερη λύση θα μπορουσε να είναι η εφαρμογή ενος υβριδικού μηχανισμου για την ελαστικότητα σε υποδοχείς με περιορισμό στην κατανάλωση των πόρων.

Πιο συγκεκριμένα θα μπορούσε να χρησιμοποιηθεί ο VPA μέχρι να φτάσει στο όριο των πόρων που μπορει να διαθέσει στην συνέχεια αν δεν καλύπτονται οι αναγκες ο HPA να ανοίξει έναν καινουριο σε έναν άλλο μηχάνημα. Όμως οι περιορισμοι της ερευνας μας δεν επέτρεψαν να αναπτυξουμε τέτοια πειράματα.

5 Επίλογος

Η συγκεκριμένη διπλωματική εργασία καταπιάνεται με την μελέτη των μηχανισμών ελαστικότητας σε περιβάλλοντα ενορχήστρωσης υποδοχέων. Δηλαδή πιο συγκεκριμένα, ασχοληθήκαμε με την ανάλυση του υπάρχοντος θεωρητικού υπόβαθρου στον τομέα των συστημάτων ενορχήστρωσης υποδοχέων και της ελαστικότητας σαν όρος του υπολογιστικού νέφους, της κατηγοριοποίησης και της εξειδίκευσής της. Στη συνέχεια εξομοιώνουμε διάφορα πειραματικά σενάρια με σκοπό να αναδείξουμε τα υπέρ και τα κατά του κάθε μηχανισμού ελαστικότητας στο περιβάλλον του Kubernetes και οδηγηθήκαμε σε κάποια συμπεράσματα.

5.1 Σύνοψη και συμπεράσματα

Τα συμπεράσματα στα οποία οδηγηθήκαμε είναι ότι ο οριζόντιος μηχανισμός ελαστικότητας αποδίδει καλύτερα σε αρχικά στάδια χρονικά και στη συνέχεια ο κατακόρυφος μηχανισμός ελαστικότητας, ο οποίος αποδίδει καλύτερα μετά την πάροδο ενός χρονικού διαστήματος που αυξομειώνει τον χώρο που μπορεί η εφαρμογή να χρησιμοποιεί από τους πόρους του συστήματος με βάση τον αριθμό των αιτημάτων και του υπολογιστικού φόρτου που έχει παρατηρήσει στο προαναφερθέν αρχικό διάστημα.

Πιο συγκεκριμένα, μετά από την πάροδο ενός χρονικού διαστήματος παρατηρήσαμε ότι υπερισχύει ο μηχανισμός ελαστικότητας ο οποίος πραγματοποιεί κατακόρυφη ή vertical αυξομείωση του χώρου που αποδίδεται σε κάθε στιγμιότυπο στο περιβάλλον του Kubernetes από τους πόρους του συστήματος με βάση τον υπολογιστικό φόρτο που πραγματοποιεί η εφαρμογή μας. Βέβαια ο συγκεκριμένος μηχανισμός έχει κάποια όρια, όπως ότι δεν μπορεί ακόμη τουλάχιστον να παραμετροποιηθεί για εφαρμογές οι οποίες έχουν δικτυακό φόρτο.

5.2 Όρια και περιορισμοί της έρευνας

Τα όρια και οι περιορισμοί της συγκεκριμένης έρευνας και μελέτης που πραγματοποιήσαμε είναι ότι περιορίζεται στο περιβάλλον ενός υπολογιστή και στα πειράματα που πραγματοποιήθηκαν σε αυτόν, οπότε δεν θα ήταν ασφαλές να οδηγηθούμε σε συμπεράσματα προς το παρόν τα οποία θα αφορούσαν μεγάλης κλίμακας υπολογιστικά συστήματα και τον τρόπο παραμέτροποίησης τους μέσω των αναφερόμενων μηχανισμών ελαστικότητας.

Ακόμη, ένα όριο της συγκεκριμένης μελέτης είναι ότι επικεντρωθήκαμε σε ένα εργαλείο ενορχήστρωσης υποδοχέων και τους μηχανισμούς ελαστικότητας που προσφέρονται σε αυτό, όπως επίσης ότι η πειραματική μελέτη πραγματοποιήθηκε μέσω της εξομοίωσης της συστοιχίας των υπολογιστικών συστημάτων μέσω ενός εικονικού μηχανήματος (virtual machine) σε ένα υπολογιστή.

5.3 Μελλοντικές Επεκτάσεις

Μελλοντικές επεκτάσεις της συγκεκριμένης έρευνας και μελέτης που πραγματοποιήσαμε θα μπορούσαν να αποτελέσουν η εφαρμογή ενός υβριδικού μηχανισμού χρησιμοποιώντας με αυτόν τον τρόπο τα υπέρ των δύο μηχανισμών σε διαφορετικές φάσεις της λειτουργίας του υπολογιστικού συστήματος. Δηλαδή για παράδειγμα, θα μπορούσε να υλοποιηθεί στον αρχικό χρόνο της λειτουργίας της εφαρμογής μας, στο περιβάλλον του Kubernetes, ο οριζόντιος μηχανισμός ελαστικότητας που αποδίδει καλύτερα σε μικρά χρονικά διαστήματα. Στην συνέχεια θα μπορούσε η υλοποίηση να μεταβεί στον κατακόρυφο μηχανισμό ελαστικότητας, ο οποίος αποδίδει καλύτερα μετά την πάροδο ενός χρονικού διαστήματος που αυξομειώνει τον χώρο που μπορεί η εφαρμογή να χρησιμοποιεί από τους πόρους του συστήματος με βάση τον αριθμό των αιτημάτων και του υπολογιστικού φόρτου που έχει παρατηρήσει στο προαναφερθέν αρχικό διάστημα.

Ακόμη θα μπορούσε να πραγματοποιηθεί η αντίστοιχη μέλετη που πραγματοποιήσαμε στο περιβάλλον ενός υπολογιστή με ισχυρότερους υπολογιστικούς πόρους ή ακόμα καλύτερα αναμεσα σε πολλά φυσικά μηχανήματα. Η εκτέλεση

πειραμάτων σε τέτοια περιβάλλοντα θα μπορούσαν να οδηγήσουν σε ασφαλέστερα συμπεράσματα.

Βιβλιογραφία

[1] P. Heidari, Y. Lemieux, and A. Shami, “QoS Assurance with Light Virtualization - A Survey,” in *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2016, pp. 558–563.

[2] M. A. Rodriguez and R. Buyya, “Container-based cluster orchestration systems: A taxonomy and future directions,” *Softw. Pract. Exp.*, vol. 49, no. 5, pp. 698–719, 2019.

[3] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, “Elasticity in Cloud Computing: State of the Art and Research Challenges,” *IEEE Trans. Serv. Comput.*, vol. 11, no. 2, pp. 430–447, Mar. 2018.

[4] “Microservices,” *Wikipedia*. 21-Oct-2019.

[5] “Why should you use microservices and containers?,” *IBM Developer*, 01-Nov-2018.

[6] A. Naskos, A. Gounaris, and S. Sioutas, “Cloud Elasticity: A Survey,” in *Algorithmic Aspects of Cloud Computing*, Cham, 2016, pp. 151–167.

[7]“What is Kubernetes.” [Online]. Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/.

[8] “Kubernetes,” *Wikipedia*. 10-Oct-2019.

[9] “Horizontal Pod Autoscaler.” [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/. [10] “kubernetes/autoscaler,” *GitHub*. [Online]. Available: https://github.com/kubernetes/autoscaler.

[10] “Apache Bench - Overview - Tutorialspoint.” [Online]. Available: https://www.tutorialspoint.com/apache_bench/apache_bench_overview.htm.

[11] F. Stutz, *flaviostutz/web-stress-simulator*. 2019.

[12] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: state-of-the-art and research challenges,” Journal of Internet Services and Applications, vol. 1, no. 1, pp. 7–18, May 2010.

[13] Luong, Nguyen Cong, et al. "Resource management in cloud networking using economic analysis and pricing models: A survey." IEEE Communications Surveys & Tutorials 19.2 (2017): 954-1001.

[14] Pahl, Claus. "Containerization and the paas cloud." IEEE Cloud Computing 2.3 (2015): 24-31.

[15] Thönes, Johannes. "Microservices." IEEE software 32.1 (2015): 116-116.

[16] Morabito, Roberto, Jimmy Kjällman, and Miika Komu. "Hypervisors vs. lightweight virtualization: a performance comparison." 2015 IEEE International Conference on Cloud Engineering. IEEE, 2015.

[17] A. Mittal, "Kubernetes Architecture Diagram." [Online]. Available: http://www.unixcloudfusion.in/2018/02/kubernetes-architecture-diagram.html.