ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σχολή Ηλεκτρολόγων και Μηχανικών Υπολογιστών

Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής

Δημιουργία και κατηγοριοποίηση προφίλ φόρτου εργασίας εφαρμογών στο υλικό υπολογιστικών συστημάτων σε περιβάλλον Docker

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Φοίβος Αστέριος Νταντάμης

Επιβλέπουσα Καθηγήτρια

Θεοδώρα Βαρβαρίγου

Καθηγήτρια Ε.Μ.Π

ΑΘΗΝΑ ΜΑΪΟΣ 2022

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σχολή Ηλεκτρολόγων και Μηχανικών Υπολογιστών

Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής

Δημιουργία και κατηγοριοποίηση προφίλ φόρτου εργασίας εφαρμογών στο υλικό υπολογιστικών συστημάτων σε περιβάλλον Docker

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Φοίβος Αστέριος Νταντάμης

Επιβλέπουσα Καθηγήτρια

Θεοδώρα Βαρβαρίγου

Καθηγήτρια Ε.Μ.Π

Εγκρίθηκε από την εξεταστική τριμελή επιτροπή στις 16 Απριλίου 2021

.....	.....	.....
Θ. Βαρβαρίγου	Ε. Βαρβαρίγος	Σ. Παπαβασιλείου
Καθηγήτρια Ε.Μ.Π	Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π

ΑΘΗΝΑ ΜΑΪΟΣ 2022

.....

Φοίβος Αστέριος Νταντάμης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Το cloud computing έχει πια εδραιωθεί ως τεχνολογία. Παράλληλα, τα containers προσφέρουν ευελιξία, απομόνωση, και δυνατότητα scaling ώστε να χρησιμοποιούνται όλο και περισσότερο σε cloud περιβάλλοντα. Με εκατοντάδες διαφορετικά πλέον διαθέσιμα instances(στιγμιότυπα) από τους providers(παρόχους) (Amazon Web Services, Microsoft Azure, Google Cloud κ.τ.λ) προκύπτει η ανάγκη βέλτιστης επιλογής αυτών. Σε αυτήν τη διπλωματική εργασία παρουσιάζουμε ένα εργαλείο και μία offline μεθοδολογία δημιουργίας και πρόβλεψης προφίλ εφαρμογών που εκτελούνται μέσα σε docker containers.

Η πρόβλεψη αναφέρεται στην αντιστοίχιση με ήδη γνωστά προφίλ που έχουν προκύψει από benchmarks για τα οποία ο διαχειριστής ενός συστήματος γνωρίζει ήδη τις ανάγκες σε πόρους. Μέσω αυτής της αντιστοίχισης θα μπορεί να επιλέξει βέλτιστα το κατάλληλο instance και να μειώσει το κόστος διαχείρισης, αφού δεν θα πληρώσει για παραπάνω πόρους από ό,τι χρειάζεται. Από τη στιγμή που όλη αυτή η διαδικασία μπορεί να γίνει offline, μειώνεται το κόστος επίσης διότι δεν προκύπτει η ανάγκη ενοικίασης πόρων για τον χαρακτηρισμό της συμπεριφοράς μιας εφαρμογής.

Παρατίθεται λεπτομερώς η υλοποίηση του προαναφερθέντος εργαλείου σε Node-red flows. Αναλύεται όλη η μεθοδολογία και ο τρόπος που καταλήξαμε σε αυτήν. Τέλος παρουσιάζεται μία σειρά πειραμάτων βάση των οποίων ελέγχουμε την ορθότητα της μεθοδολογίας και αξιολογούμε τα αποτελέσματα των προβλέψεων.

Λέξεις-κλειδιά: Resource management, profiling, containerised application, Docker, Node-red

Abstract

Cloud computing is now established as technology. At the same time, containers offer flexibility, isolation and scaling ability which results in becoming increasingly used in cloud environments. Hundreds of different instances available from providers (Amazon Web Services, Microsoft Azure, Google Cloud, etc.) create the need for optimal choice. In this thesis we present a tool and an offline methodology for creating and predicting applications profiles executed in docker containers.

The prediction refers to the matching with already known profiles that have emerged from benchmarks for which the administrator of a system already knows the needs of resources. Through this match he will be able to optimally choose the appropriate instance and reduce the management costs, since he will not pay for more resources than he needs. Since all this process can be done offline, costs are also reduced because there is no need to rent resources to characterize the behavior of an application.

The implementation of the aforementioned tool in Node-Red Flows is detailed. All the methodology and the way we came up with it is analyzed. Finally, a series of experiments are presented on the basis of which we control the correctness of the methodology and evaluate the results of the forecasts.

Keywords: Resource management, profiling, containerised application, Docker, Node-red

Ευχαριστίες

Θα ήθελα αρχικά να ευχαριστήσω την κα. Θεοδώρα Βαρβαρίγου που μου έδωσε την ευκαιρία να ασχοληθώ με ένα εξαιρετικά ενδιαφέρον ερευνητικό θέμα στα πλαίσια του εργαστηρίου Distrubuted Knowledge and Media Systems.

Συνεχίζοντας, θα ήθελα να ευχαριστήσω θερμά τον διδάκτορα πλέον Αλέξανδρο Ψύχα για την αμέριστη βοήθεια και πολύτιμη υποστήριξη που μου παρείχε καθόλη τη διάρκεια εκπόνησης της παρούσας διπλωματικής εργασίας.

Τέλος, ευχαριστώ τους γονείς μου, τα αδέρφια μου, τους φίλους μου και όλους τους κοντινούς μου ανθρώπους, για την έμπρακτη στήριξη από την αρχή μέχρι το τέλος αυτού του ταξιδιού.

Η παρούσα διπλωματική εργασία αφιερώνεται στην οικογένειά μου.

Φοίβος Αστέριος Νταντάμης

27 Μαϊου 2022

Περιεχόμενα

Λίστα σχημάτων

Λίστα Γραφημάτων

Λίστα Πινάκων

1 Εισαγωγή

Η συνεχής άνοδος των τεχνολογιών υπολογιστικού νέφους (Cloud computing) και κατανεμημένων αρχιτεκτονικών [1][2][3] έχει δημιουργήσει νέες προκλήσεις ως προς την επιλογή των κατάλληλων υποδομών υλικού που οι providers διαθέτουν [4]. Η πληθώρα των διαφορετικών κατηγοριών των εκτελούμενων εφαρμογών, οδηγούν στην ανάγκη χαρακτηρισμού τους ώστε ο διαχειριστής να μπορεί μέσα σε κάποια πλαίσια να προβλέψει τους απαιτούμενους πόρους πριν από την εγκατάσταση και εκκίνηση τους. Οι λόγοι αφορούν προφανώς την επίδοση αλλά και το κόστος, αφού η ενοικίαση μη χρησιμοποιούμενων πόρων έχει ως αποτέλεσμα έξοδα τα οποία μπορούν να αποφευχθούν, ενώ η έλλειψη πόρων μπορεί να οδηγήσει ακόμα και σε μη διαθεσιμότητα της υπηρεσίας. Επιπροσθέτως, ο χαρακτηρισμός των αναγκών μιας εφαρμογής μέσα από την εκτέλεση της σε ένα περιβάλλον cloud απαιτεί την ενοικίαση των πόρων για όλη τη διάρκεια της εκτέλεσης και επομένως όσο πιο ακριβές ζητάμε να είναι το προφίλ της εφαρμογής, τόσο αυξάνεται και το κόστος. Προκύπτει λοιπόν η ανάγκη παραγωγής των προφίλ αυτών τοπικά και όσο το δυνατόν γίνεται ανεξάρτητα από το hardware(υλικό), αφού σε αντίθετη περίπτωση θα χρειαζόταν να ελέγχουμε όλους τους διαφορετικούς διαθέσιμους συνδυασμούς τεχνικών χαρακτηριστικών.

Η επικρατέστερη μέθοδος χαρακτηρισμού των instances που παρέχουν οι εταιρείες που δραστηριοποιούνται στον χώρο του cloud computing είναι μέσω των benchmarks. Τα benchmarks υλοποιούν την απλή ιδέα του σημείου αναφοράς. Κάθε benchmark είναι ένα αυτοτελές πρόγραμμα που εκτελεί βασικές λειτουργίες ή υπολογισμούς χαρακτηριστικών εφαρμογών. Συγκρίνοντας τις επιδόσεις των εκτελέσεων αυτών μπορούμε να εξάγουμε συμπεράσματα σχετικά με την αναμενόμενη επίδοση άλλων εφαρμογών που μοιάζουν. Δημιουργώντας ένα μεγάλο σύνολο προφίλ από εκτελέσεις διαφορετικής φύσης benchmarks που προσομοιάζουν πραγματικές λειτουργίες πραγματικών εφαρμογών και υπηρεσιών όπως για παράδειγμα, εκτέλεση queries σε μία βάση δεδομένων, εκτέλεση αιτημάτων σε έναν εξυπηρετητή, συμπίεση αρχείων, εκτέλεση παράλληλων νημάτων που εκτελούν απαιτητικούς υπολογισμούς κ.τ.λ, μπορούμε να υποθέσουμε ότι θα υπάρχει μία αντιστοιχία με κάποιο από τα προφίλ αυτά με οποιαδήποτε τυχαία

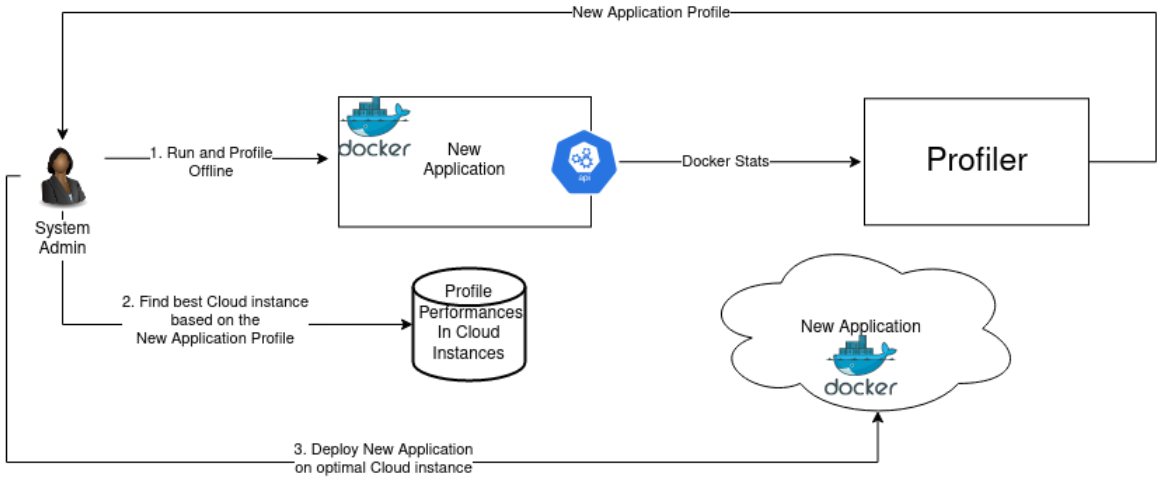
εφαρμογή. Ακόμα και αν η αντιστοιχία δεν προσομοιάζει ακριβώς την φύση της εφαρμογής, θα εντοπιστεί μία μεγάλη συνάφεια σε ό,τι αφορά την επίδοση.

Σε όλο αυτό το περιβάλλον του cloud computing, σημαντικό ρόλο έχει αποκτήσει η έννοια των containers. Πρόκειται για μία μορφή εικονικοποίησης όπως τα virtual machines, με τη διαφορά ότι είναι πιο “ελαφριά”, δεσμεύουν μόνο τους απολύτως απαραίτητους πόρους και είναι μεταφέρσιμα μεταξύ διαφορετικών λειτουργικών συστημάτων και διαφορετικών μηχανημάτων χωρίς καμία επιπλέον παραμετροποίηση. Μέσα σε ένα container μπορούμε να τρέξουμε οποιουδήποτε είδους εφαρμογή και συνεπώς και benchmarks. Μπορούμε με λίγα λόγια να μεταφέρουμε το ίδιο περιβάλλον εκτέλεσης σε διαφορετικούς στόχους και να εκτελέσουμε τα ίδια benchmarks σε μία κατάσταση όπου η επίδοση θα εξαρτάται μόνο από τα υποκείμενα τεχνικά χαρακτηριστικά, πραγματικά ή εικονικά.

Το αντικείμενο μελέτης της παρούσας διπλωματικής εργασίας αφορά στον τρόπο ταξινόμησης τυχαίων εφαρμογών σε συγκεκριμένες κατηγορίες που εξάγονται από το προφίλ τους. Συνδυάζοντας λοιπόν όσα αναφέρθηκαν μέχρι τώρα αναπτύξαμε ένα εργαλείο υπολογισμού των προφίλ αυτών για εκτελέσεις containerised εφαρμογών και συγκεκριμένα σε docker containers. Αναπαράγοντας τις εκτελέσεις μιας πληθώρας benchmarks σε διαφορετικές υποδομές μέσω των containers, δημιουργούμε χαρακτηριστικά διανύσματα βασισμένα σε μετρικές επίδοσης που μας παρέχει το docker stats API. Στη συνέχεια, δημιουργούμε ένα μοντέλο προβλέψεων, με αξιοποίηση του αλγορίθμου random forest, που προκύπτει από τα διανύσματα αυτά. Ένας διαχειριστής συστήματος θα πρέπει ακολούθως να τρέξει όλα τα benchmarks του μοντέλου στις δικές του διαθέσιμες υποδομές και να καταγράψει τη συμπεριφορά και την απόδοση τους. Έχοντας στη διάθεση του τα αποτελέσματα αυτά και το μοντέλο που του παρέχουμε, το μόνο που χρειάζεται να κάνει ώστε να προβλέψει τη συμπεριφορά μίας νέας τυχαίας εφαρμογής είναι να παράξει το χαρακτηριστικό της διάνυσμα και να το τροφοδοτήσει στο σύστημα πρόβλεψης. Αυτό θα έχει ως αποτέλεσμα την αντιστοίχισή της με κάποιο από τα γνωστά benchmarks με αποτέλεσμα να μπορεί να προβλέψει τη συμπεριφορά της και να επιλέξει με τον βέλτιστο τρόπο που θα γίνει το initial deployment(αρχική εγκατάσταση). Το πιο σημαντικό είναι ότι θα μπορέσει να παράξει το απαιτούμενο διάνυσμα σε οποιοδήποτε μηχάνημα διαθέτει αφού όπως

θα δείξουμε παρακάτω, το αποτέλεσμα δεν εξαρτάται από διαφοροποιήσεις στο υλικό.

Η συνέχεια της διπλωματικής εργασίας διαρθρώνεται ως εξής. Στο κεφάλαιο 2 παρουσιάζεται η μέχρι τώρα σχετική έρευνα και αναφέρονται οι διαφορετικές προσεγγίσεις και ροές εργασιών που έχουν ακολουθηθεί στον χώρο της διαχείρισης πόρων και των προβλέψεων. Στο κεφάλαιο 3 παρουσιάζεται η υλοποίηση μας για το εργαλείο προβλέψεων. Πιο συγκεκριμένα, στο υποκεφάλαιο 3.1, παρουσιάζεται το τεχνολογικό stack που χρησιμοποιήθηκε, ενώ στο 3.2 αναλύεται η υλοποίηση. Στη συνέχεια, στο κεφάλαιο 4, παραθέτουμε ενδελεχώς τις συνθήκες και τα αποτελέσματα των πειραματισμών μας. Αναλυτικότερα, στο υποκεφάλαιο 4.1 περιγράφουμε το περιβάλλον και τις υποδομές των πειραματισμών. Στο υποκεφάλαιο 4.2 αναφέρουμε τις παραδοχές που κάναμε και τον τρόπο με τον οποίο ορίστηκαν τα διανύσματα που απαρτίζουν τα προφίλ των εφαρμογών. Στο υποκεφάλαιο 4.3 παραθέτουμε τη διαδικασία επιλογής του αλγορίθμου μηχανικής μάθησης. Στο υποκεφάλαιο 4.4 παραθέτουμε και σχολιάζουμε τις προκύπτουσες γραφικές παραστάσεις και τέλος στο υποκεφάλαιο 4.5 συμπερασματολογούμε και σχολιάζουμε τα αποτελέσματα. Καταλήγουμε με το κεφάλαιο 5 της σύνοψης της εργασίας και ακολουθεί η λίστα με όλες τις αναφορές.



Σχήμα 1: Παράδειγμα χρήσης του Profiler

2 Βιβλιογραφική και τεχνολογική επισκόπηση ερευνητικού τομέα

Έχουν γίνει διάφορες προσπάθειες και έχουν υπάρξει διαφορετικές προσεγγίσεις σε ό,τι αφορά στο πρόβλημα του χαρακτηρισμού της συμπεριφοράς μιας εφαρμογής και ως εκ τούτου της πρόβλεψης και βελτιστοποίησης των επιδόσεών της. Για παράδειγμα, μία προσπάθεια δημιουργίας ενός συστήματος βελτιστοποίησης απόδοσης μέσω κατηγοριοποίησης των εισερχόμενων εργασιών έχει γίνει στο πεδίο των MapReduce εφαρμογών [5]. Η κατηγοριοποίηση απαίτησε δύο στάδια. Πρώτον τον ορισμό τριών κατηγοριών που προέκυψαν από μία παραλλαγή του αλγορίθμου k-means++ με είσοδο από ένα σύνολο εργασιών που είναι κοινώς αποδεκτά ως benchmarks, με άλλα λόγια τη δημιουργία του μοντέλου προβλέψεων μέσω τεχνικών μηχανικής μάθησης. Δεύτερον, την εκτέλεση ενός υποσυνόλου της εισόδου των εισερχόμενων εργασιών και στην συνέχεια της αντιστοίχισης με μία από τις γνωστές κατηγορίες του πρώτου βήματος. Τα διανύσματα που χαρακτηρίζουν τις εκτελέσεις προκύπτουν από την καταγραφή του χρόνου υπολογισμού που αφιερώνεται στον επεξεργαστή, τη μνήμη και το δίσκο. Για κάθε μία από τις κατηγορίες, μέσω αναζήτησης έχει οριστεί ένας συνδυασμός κάποιων διαθέσιμων παραμέτρων που βελτιώνει την απόδοση των εκτελέσεων. Τέλος, η εφαρμογή όλης της διαδικασίας σε ένα Amazon EC2 testbed φανέρωσε τα επίπεδα βελτιστοποίησης που προέκυψαν σε σχέση με την προεπιλεγμένη παραμετροποίηση. Σε μία άλλη περίπτωση [6], επιχειρήθηκε η δημιουργία προφίλ για τα ίδια τα benchmarks ώστε να αποκτηθεί ενδότερη κατανόηση της λειτουργίας και της φύσης τους, αφού συνήθως η τεκμηρίωσή τους περιορίζεται σε μία πολύ απλή περιγραφή. Χρησιμοποιήθηκαν για τον σκοπό αυτό τα execution traces που καταγράφει το εργαλείο LLTng και αναλύθηκαν low level μετρικές όπως CPU utilization, parallelization, stability, memory usage.

Φυσικά, εφόσον μιλάμε για προβλέψεις, έχουν γίνει αρκετές προσπάθειες που εμπλέκουν πιο σύνθετες μεθόδους μηχανικής μάθησης. Στο [7], τα data traces από servers(εξυπηρετητές) που φιλοξενούν διαφορετικού είδους υπηρεσίες χρησιμοποιούνται σε ένα νευρωνικό δίκτυο και μία παραλλαγή του black hole αλγορίθμου ώστε να προβλεφθεί το μελλοντικό workload και επομένως να μπορούν να οργανωθούν αναλόγως οι απαιτούμενοι πόροι. Συνεχίζοντας, βλέπουμε στο [8]

μία διαδικασία που επιχειρεί τη δημιουργία του μοντέλου πρόβλεψης offline και βελτιώνει τα αποτελέσματα του online profiling χρησιμοποιώντας transfer learning. Στη δική μας προσέγγιση, η διαδικασία του profiling λαμβάνει χώρα εξ ολοκλήρου σε offline περιβάλλον.

Το κόστος αποτελεί έναν από τους κύριους λόγους που προσπαθούμε να βελτιστοποιήσουμε την αξιοποίηση των διαθέσιμων πόρων. Λαμβάνοντας λοιπόν υπόψιν τον παράγοντα αυτό, στο [9] βλέπουμε μία αυτοματοποίηση όλης της διαδικασίας εγκατάστασης, παραμετροποίησης και δημιουργίας προφίλ στο cloud. Επιλέχθηκε να δημιουργηθούν πολύ λεπτομερή προφίλ συνδυάζοντας την πληροφορία του κόστους όπως αυτά προκύπτουν μέσα από τις εκτελέσεις στα cloud instances που επιλέγει ο χρήστης. Στοιχεία των προφίλ αποτελούν το ύψος αξιοποίησης του επεξεργαστή, του δίσκου, του δικτύου αλλά και ο χρόνος εκτέλεσης και φυσικά το κόστος όπως προκύπτει από την τιμολόγηση του provider και το οποίο προφανώς εξαρτάται από όλα τα προηγούμενα. Για την αξιοποίηση των προφίλ αυτών θα χρειαστεί ο χρήστης να έχει εξειδικευμένες γνώσεις πάνω στις εφαρμογές ώστε να πάρει αποφάσεις. Αντιθέτως, στην προσέγγισή μας οι αποφάσεις στηρίζονται εξ ολοκλήρου στην αντιστοίχιση προφίλ και benchmarks και δεν είναι απαραίτητη πουθενά η κρίση του χρήστη.

Το DocLite [10] είναι ένα containerised εργαλείο που προσπαθεί να αξιολογήσει VMs(εικονικές μηχανές) αξιοποιώντας containerised benchmarks. Τα αποτελέσματά του δείχνουν ότι εκτελώντας τα benchmarks ακόμα και σε περιορισμένος πόρους των VMs, μπορούμε να εξάγουμε σε ποσοστό έως και 91%, την ίδια πληροφορία από τα προφίλ με την περίπτωση να χρησιμοποιούσαμε ολόκληρο το VM. Επιπροσθέτως, η αξιολόγηση γίνεται σχεδόν σε πραγματικό χρόνο. Ισχυρές είναι επίσης οι ενδείξεις ότι οι μετρήσεις μέσω “ελαφριών” benchmarks, μέχρι και microbenchmarks αποτελούν έναν καθόλα έγκριτο τρόπο εξαγωγής συμπερασμάτων για την απόδοση των διάφορων cloud configurations [11]. Στη δική μας περίπτωση αυτό που περιορίζεται είναι ο χρόνος εκτέλεσης, ενώ οι διαθέσιμοι πόροι αξιοποιούνται σε επίπεδα που εξαρτώνται από τα benchmarks.

Το RUBiS [12] είναι μία Profiling-as-a-Service Web εφαρμογή που προσεγγίζει τη διαχείριση των cloud εφαρμογών μέσω ταυτόχρονου scaling και profiling. Ο χρήστης ορίζει περιορισμούς σχετικά με την απόδοση και το κόστος και με προτεραιότητα το δεύτερο, καθοδηγείται η διαδικασία του scaling, ενώ

παράλληλα παράγονται λεπτομερή προφίλ για όσο περισσότερους κόμβους που απαρτίζουν την εφαρμογή γίνεται. Στο [13], το auto-scaling επιτυγχάνεται μέσω του offline profiling. Δημιουργήθηκαν χαρακτηριστικά workloads για διαφορετικούς τύπους χρηστών τα οποία δόθηκαν ως είσοδος στις εφαρμογές ώστε να αποκτηθεί πιο λεπτομερής γνώση για την εξάρτηση της επίδοσης. Στη συνέχεια μετρήθηκε η επίδοση για κάθε περίπτωση και έγινε μια προσπάθεια πρόβλεψης του μελλοντικού workload έτσι ώστε ο διαχειριστής να μπορεί να καταστρώσει μια στρατηγική για το scalability λαμβάνοντας υπόψιν το QoS(Quality of Service) και το κόστος.

Συμπεριλαμβάνοντας τον παράγοντα του background workload, σε διαμοιραζόμενα VMs, στο [14] μία canonical correlation ανάλυση επιχειρεί να συμπεράνει τους παράγοντες του συστήματος που επηρεάζουν την απόδοση περισσότερο. Δεδομένα σχετικά με την χρησιμοποίηση των πόρων μέσω online profiling επιτρέπουν την πρόβλεψη των αναγκών υλικού για ένα συγκεκριμένο επίπεδο υπηρεσιών της εφαρμογής. Μία πολύ παρεμφερής περίπτωση [15], αξιοποιεί τους Hardware Performer Counters ώστε να συμπεράνει τα ζητούμενα προφίλ. Δε συνυπολογίσαμε τον συγκεκριμένο παράγοντα, αφού προϋποθέτει ότι ο χρήστης κατέχει βαθιά γνώση για τον τρόπο της παράλληλης αξιοποίησης πόρων των εφαρμογών. Εξάλλου μελετάμε μόνο containerised εφαρμογές που εξ ορισμού είναι απομονωμένες.

Η πιο συγγενική προσέγγιση με τη μέθοδο μας είναι [16], που παρουσιάζει μία offline μεθοδολογία βασισμένη στη μηχανική μάθηση για την πρόβλεψη της επίδοσης HPC εφαρμογών. Μετρικές που δεν εξαρτώνται από το υλικό συλλέγονται με τη χρήση ενός LLVM plugin ώστε να δημιουργηθούν τα προφίλ. Για να επιτευχθεί το παραπάνω, ο κώδικας των εφαρμογών πρέπει να ενορχηστρωθεί κατά τη μεταγλώττιση και συνεπώς είναι προαπαιτούμενο ο πηγαίος κώδικας να είναι διαθέσιμος. Παρομοίως στη μέθοδο μας εφαρμόζουμε offline αλγόριθμους μηχανικής μάθησης για να πετύχουμε το ίδιο πράγμα, με τη διαφορά ότι βασιζόμαστε σε μετρικές που συλλέγονται κατά την εκτέλεση οποιασδήποτε εφαρμογής μέσα σε docker containers και αποδεικνύουμε μέσω των πειραματισμών ότι τα αποτελέσματά τείνουν να μην εξαρτώνται από το υλικό, ακόμα και αν οι μετρικές αυτές καθεαυτές εξαρτώνται.

3 Υλοποίηση συστήματος εξαγωγής προφίλ Εφαρμογών

Η υλοποίηση του profiler βασίζεται σε Node-red flows. Όταν έχει ξεκινήσει και τρέχει ένα Docker container, η διαδικασία του profiling μπορεί να πυροδοτηθεί μέσω των endpoints που κάνει expose ο profiler. Για την αποθήκευση των δεδομένων χρησιμοποιείται η βάση δεδομένων MongoDB, ενώ για τη δημιουργία του μοντέλου μηχανικής μάθησης και τις προβλέψεις αξιοποιείται το εργαλείο Weka. Τα benchmarks βάση των οποίων δημιουργήσαμε το μοντέλο μας αντλήθηκαν από τη σουίτα Phoronix Test Suite, αλλά και την οικογένεια banchmakrs που ονομάζεται sysbench.

Ο profiler χωρίζεται σε 5 flows. Δύο από αυτά είναι υπέυθυνα για τη δημιουργία των προφίλ. Το πρώτο υπολογίζει τα προφίλ των benchmarks σύμφωνα με τα οποία θα προκύψει το μοντέλο προβλέψεων. Το δεύτερο είναι υπεύθυνο για τα προφίλ των νέων εφαρμογών τα οποία θέλουμε να αντιστοιχισούμε με τα ήδη γνωστά και συνεπώς πέρα απο τη σύνθεση του προφίλ εκκινεί και τη διαδικασία της πρόβλεψης. Όλες οι καταγεγραμμένες μετρικές αποθηκεύονται σε ένα ενδιάμεσο στάδιο στη βάση δεδομένων. Το ίδιο συμβαίνει και με τα τελικά προφίλ που εξάγονται με χρήση των ενδιάμεσων δεδομένων με τη διαφορά ότι αποθηκεύονται και σε αρχεία csv ώστε να τροφοδοτηθούν στους αλγορίθμους μηχανικής μάθησης. Τα επόμενα 2 flows είναι υπεύθυνα για το κομμάτι του machine learning. Το ένα από αυτά δημιουργεί το μοντέλο προβλέψεων ενώ το άλλο πραγματοποιεί τις προβλέψεις με χρήση αυτού του μοντέλου. Τέλος, το τελευταίο flow δίνει τη δυνατότα της ταυτόχρονης εξυπηρέτησης και εκτέλεσης πολλαπλών αιτημάτων κάνοντας έτσι εφικτή τη παραλληλοποίηση τους. Επιτρέπει με αυτόν τον τρόπο και το scaling(κλιμάκωση) ώστε να επιτευχθεί ελάφρυνση και αποσυμφόρηση σε ενδεχόμενη πλήρης αξιοποίηση των διαθέσιμων πόρων.

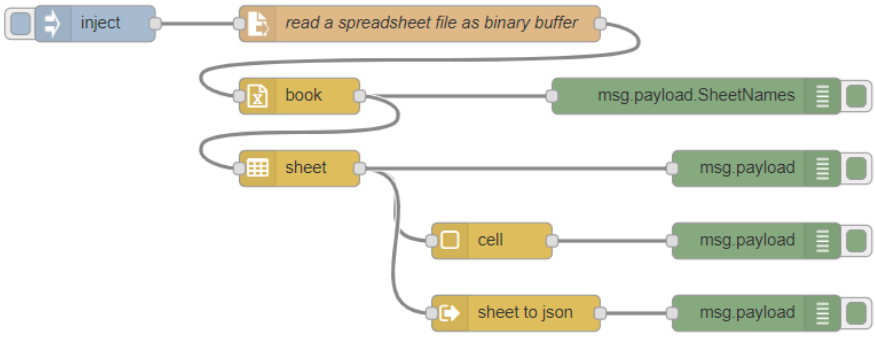
Θα αναφερθούμε πιο αναλυτικά στο τεχνολογικό stack και τα εργαλεία που χρησιμοποιήθηκαν και στη συνέχεια θα παρουσιάσουμε λεπτομερώς τα στοιχεία της υλοποίησης.

3.1 Τεχνολογίες και πλατφόρμες υλοποίησης λογισμικού

Node-red [1]

Σχήμα 2: Node-RED logo

Το Node-red είναι ένα προγραμματιστικό εργαλείο βασισμένο στο Node.js, με το οποίο μεταξύ άλλων, μπορεί κανείς να δημιουργήσει APIs. Ακολουθεί το υπόδειγμα του flow-based προγραμματισμού και διαθέτει έναν γραφικό editor, προσβάσιμο μέσω του browser. Παρέχει διάφορα nodes(κόμβους) που λειτουργούν σαν black boxes τα οποία συνδέονται και ανταλλάζουν μηνύματα και μέσω των ροών μηνυμάτων που προκύπτει υλοποιείται η λογική του προγράμματος. Για πιο εξεζητημένες λειτουργίες υπάρχει η δυνατότητα δημιουργίας των δικών μας κόμβων, μέσα στους οποίους μπορούμε να υλοποιήσουμε οτιδήποτε χρειαζόμαστε με τη χρήση javascipt. Τα προγράμματα-flows αποθηκεύονται σε μορφή JSON και είναι εύκολα μεταφέρσιμα.



Σχήμα 3: Παράδειγμα Node-RED flow

[1] https://nodered.org/

Docker [2]

Σχήμα 4: Docker logo

Το Docker είναι μία open-source πλατφόρμα που επιτρέπει το containerization εφαρμογών. Τα containers είναι αυτοτελή εκτελέσιμα που περικλείουν ό,τι μπορεί να χρειαστεί ένα πρόγραμμα, δηλαδή όλες τις εξαρτήσεις του. Η καρδιά του συστήματος Docker είναι το Docker Engine, το οποίο εμπεριέχει τρία κύρια μέρη. Τον server του Docker που ονομάζεται dockerd, ένα Rest API μέσω του οποίου μπορεί κανείς να δώσει εντολές στο dockerd και ένα CLI που δέχεται docker commands. Το dockerd χρησιμοποιείται για τη διαχείριση των containers όπως και των images βάσει των οποίων αυτά δημιουργούνται. Σημαντικό για εμάς ήταν το API του docker και συγκεκριμένα το stats endpoint το οποίο παρέχει τις low-level μετρικές κατά τη διάρκεια της εκτέλεσης, με τις οποίες θα παράγουμε τα προφίλ των containerized εφαμρογών. Παράλληλα, στον ιστότοπο του dockerhub [3] υπάρχουν αποθετήρια για πολλών ειδών έτοιμα images, τα οποία μπορεί κανείς να κατεβάσει μέσω του dockerd και να δημιουργήσει τα ανάλογα containers, στην περίπτωση μας αυτά που θα τρέχουν τα benchmarks.

Weka [4]

Σχήμα 5: Weka logo

[2] https://www.docker.com/
[3] https://hub.docker.com/
[4] https://www.cs.waikato.ac.nz/ml/weka/

Το Weka είναι ένα open-source εργαλείο γραμμένο σε Java, για την επεξεργασία και οπτικοποίηση δεδομένων που περιέχει μία ευρεία συλλογή με υλοποιημένους αλγορίθμους μηχανικής μάθησης.

MongoDB [5]

Σχήμα 6: mongoDB logo

Η MongoDB είναι το δημοφιλέστερο NoSQL πρόγραμμα βάσεων δεδομένων. Τα δεδομένα αποθηκεύονται ως documents, η δομή των οποίων είναι παρόμοια με αυτή των JSON αρχείων. Παρέχει ένα API για την εκτέλεση των queries και αναλυτικό documentation για όλες τις λειτουργίες που υποστηρίζει.

Phoronix Test Suite (PTS)[6]

Σχήμα 7: Phoronix Test Suite logo

Η σουίτα Phoronix χρησιμοποιείται για την αυτοματοποίηση της εκτέλεσης των τεστ σε διάφορες πλατφόρμες. Αναλαμβάνει δηλαδή το κατέβασμα, την εγκατάσταση, τη διαχείριση των εξαρτήσεων την εκτέλεση και την αναφορά των αποτελεσμάτων. Τραβάει τα διαθέσιμα τεστ από την ιστοσελίδα openbenchmarking.org, όπου υπάρχει πληθώρα από τεστ για πολλές κατηγορίες. Από αυτά προέρχονται και τα benchmark μας ώστε να διευκολύνουμε την εκτέλεση πολλών τελείως διαφορετικών μεταξύ τους bechmarks.

[5] https://www.mongodb.com/
[6] https://www.phoronix-test-suite.com/

Sysbench [7]

Σχήμα 8: Sysbench logo

Το sysbench είναι μία συλλογή από benchmarks για μηχανήματα Linux. Υποστηρίζει το έλεγχο των επιδόσεων της CPU, της μνήμης, των διαδικασιών εισόδου/εξόδου, των mutexes, και των βάσεων MySQL. Υποστηρίζει επίσης πολλαπλά threads και δέχεται παραμετροποιήσεις για τον όγκο του φορτίου που θα εκτελέσει καθώς και για άλλα πράγματα, όπως τον χρόνο της εκτέλεσης που μας ενδιαφέρει. Αυτή η ευελιξία στην παραμετροποίηση είναι και ο λόγος που επιλέχθηκε να συνοδέψει τα benchmarks του PTS.

3.2 Node-red flows

Η υλοποίηση ολόκληρου του εργαλείου μοιράζεται στα 5 παρακάτω flows:

- **Docker Benchmark Profile Exctractor**: Παράγει τα profiles των benchmarks εξάγοντας τις αναγκαίες μετρικές και φτιάχνει ένα csv αρχείο από το οποίο θα προκύψει το μοντέλο προβλέψεων.
- **Random Forest Weka Trainer**: Δημιουργεί το μοντέλο προβλέψεων χρησιμοποιώντας το εργαλείο Weka.
- **Application Profile Exctractor**: Παράγει τα profiles των νέων containerized εφαρμογών για τις οποίες ζητάμε την αντιστοίχιση με κάποιο από τα benchmark profiles και εκκινεί τη διαδικασία της πρόβλεψης.
- **Random Forest Weka Classifier**: Χρησιμοποιεί το μοντέλο προβλέψεων ώστε να παράξει τις προβλέψεις για τα προφίλ των

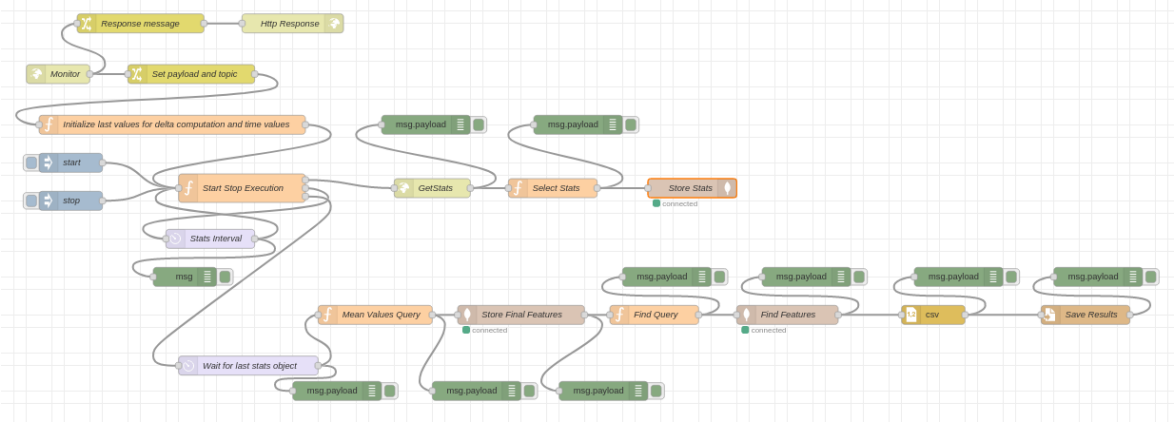
[7] https://github.com/akopytov/sysbench

containerized εφαρμογών που ζητάμε κάνοντας επίσης χρήση του Weka.

- **Requests Handler**: Επιτρέπει την ταυτόχρονη εξυπηρέτηση πολλαπλών αιτημάτων για profiling.

Ξεκινώντας την περιγραφή των κυριότερων σημείων κάθε flow, θα παραθέτουμε αρχικά ολόκληρο το flow και θα εμβαθύνουμε στη συνέχεια.

3.2.1 Docker Benchmark Profile Exctractor



Σχήμα 9: Docker Benchmark Profile Exctractor flow

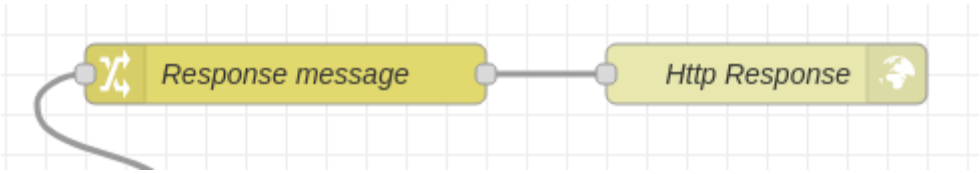
Το flow χωρίζεται πολύ γενικά σε 3 μέρη, την αρχικοποίηση, τη συλλογή δεδομένων και την παραγωγή και αποθήκευση των προφίλ

Ο πρώτος κόμβος με τον οποίο και ξεκινάει το flow είναι αυτός που δημιουργεί το endpoint του profiler. Το NODE-red “ακούει” στην πόρτα 1880 και στέλνοντας ένα POST request στο endpoint /monitor ξεκινάει το profiling μιας εφαρμογής σε κάποιο docker container που τρέχει εκείνη τη στιγμή. Για να προσδιοριστεί ποιο container ζητάμε να γίνει profiled, θα πρέπει να συμπεριλάβουμε στο σώμα του POST request το container id, που είναι ένα μοναδικό αναγνωριστικό για το συγκεκριμένο container. Επίσης πρέπει να ορίσουμε ένα όνομα το οποίο το επιλέγουμε εμείς και θα λειτουργήσει ως αναγνωριστικό του συγκεκριμένου προφίλ που προκύπτει από τη συγκεκριμένη εκτέλεση όταν αποθηκευτεί στη βάση δεδομένων.

```
{
    "cid": "428e715bf771",
    "bname": "sysbench-cpu-1000"
}
```

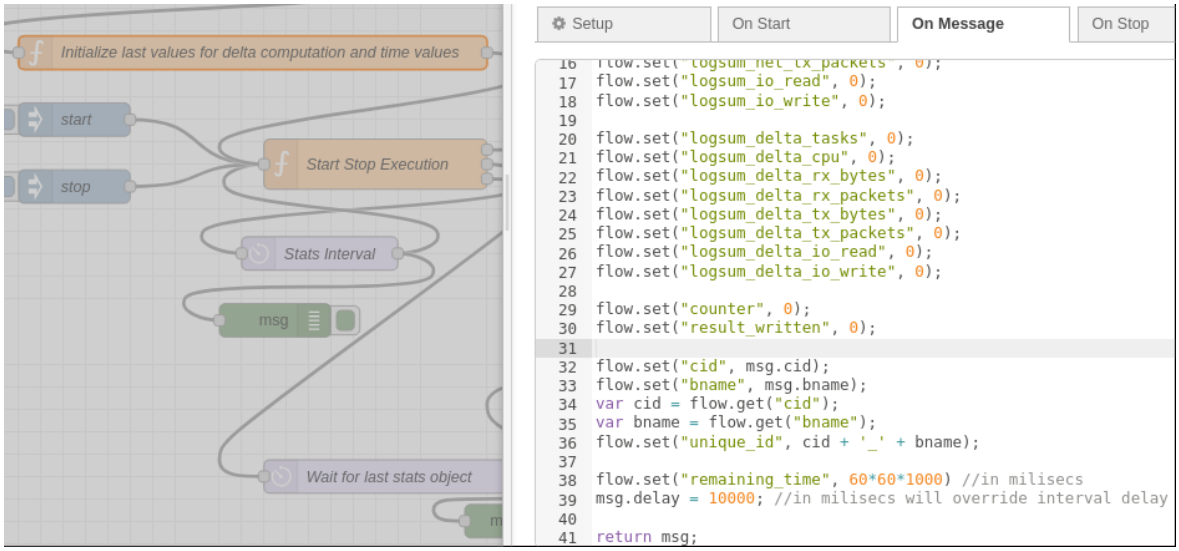
Σχήμα 10: POST request body example

Αφού ο profiler λάβει το POST request, θα επιστρέψει ένα μήνυμα ώστε να ενημερώσει αυτόν που τον κάλεσε ότι η διαδικασία έχει ξεκινήσει.



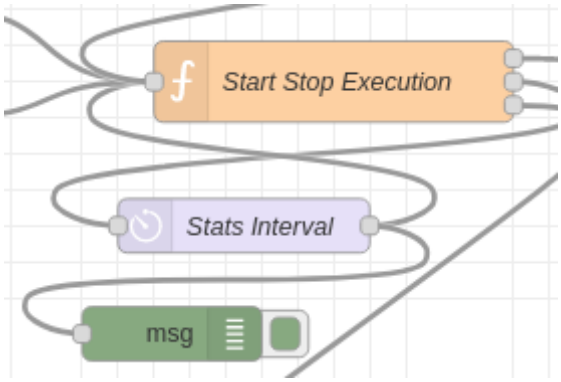
Σχήμα 11: Response nodes

Ακολουθεί η αρχικοποίηση όλων των αναγκαίων μεταβλητών. Οι μεταβλητές last_*, logsum_* και logsum_delta_*, μία για κάθε μετρική που θα κρατήσουμε, χρησιμεύουν για τον υπολογισμό της αυξομείωσης σε κάθε στιγμή μέτρησης σε σχέση με την προηγούμενη τιμή αλλά και τον υπολογισμό των γεωμετρικών μέσων όρων όπως θα αναλύσουμε και παρακάτω. Εξάγουμε επίσης τις τιμές των μεταβλητών που λάβαμε με το post request και δημιουργούμε ένα μοναδικό id που προκύπτει ως συνδυασμός του container id και του ονόματος που έδωσε ο χρήστης. Τέλος ορίζουμε το χρονικό διάστημα για το οποίο θέλουμε να διαρκέσει το profiling (αν η εκτέλεση του container τελειώσει πιο νωρίς προφανώς θα σταματήσει και το profiling) αλλά και το διάστημα ανάμεσα σε δύο διαδοχικές μετρήσεις. Όλα τα παραπάνω αποθηκεύονται σε μεταβλητές με εμβέλεια όλο το flow και συνεπώς είναι προσβάσιμες από όλους του κόμβους.



Σχήμα 12: Initiliazation code

Με τους κόμβους Start Stop Execution και Stats Interval υλοποιείται μία λούπα η οποία κάθε φορά που περνάει το ορισμένο χρονικό διάστημα εκκινεί τη διαδικασία αίτησης των μετρικών και αποθήκευσής τους. Όταν τελειώσει η εκτέλεση ή ολοκληρωθεί το διάστημα του profiling, εκκινείται το υπόλοιπο μέρος του flow που αναλαμβάνει το aggregation των αποτελεσμάτων και την αποθήκευση των τελικών προφίλ.



Σχήμα 13: Loop nodes

Με τον http requests κόμβο μπορούμε να στείλουμε αιτήματα για να ζητήσουμε τις μετρικές του container μία δεδομένη στιγμή μέσω του docker API. Το url που είναι γραμμένο μέσα στον κόμβο είναι το εξής:

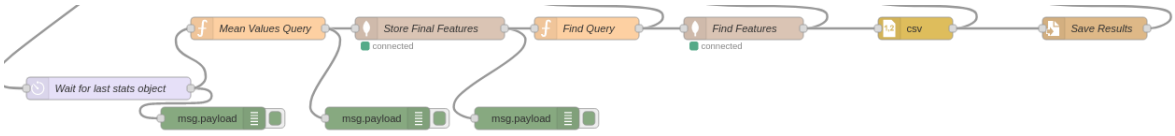
localhost:2375/containers/{{msg.cid}}/stats?stream=false

Η μεταβλητή cid περιέχει το container id του container για το οποίο ζητάμε τις μετρικές. Η παράμετρος stream=false, ορίζει ότι αφού λάβουμε την απάντηση από το request, θα κλείσει η σύνδεση και θα λάβουμε τα επόμενα δεδομένα όταν ξαναστείλουμε αίτημα. Τα δεδομένα έρχονται σε μορφή json και περιέχουν πολλές πληροφορίες οι οποίες δεν μας αφορούν και θα φιλτραριστούν στη συνέχεια. Η μορφή που έχουν είναι εξής.

Σχήμα 14: Docker stats response

Στον επόμενο κόμβο Select Stats επιλέγουμε συγκεκριμένα ποιες μετρικές θέλουμε να κρατήσουμε. Επίσης υπολογίζουμε τις μεταβολές σε σχέση με τις προηγούμενη μέτρηση για όσες μετρικές έχει νόημα και τις ορίζουμε σαν ξεχωριστή μετρική. Τέλος, κρατάμε το λογαριθμικό άθροισμα σε μεταβλητές που θα χρησιμοποιήσουμε για να υπολογίσουμε γεωμετρικούς όρους. Το κάνουμε αυτό διότι η MongoDB δεν έχει κάποια εντολή στο στάδιο του aggregation που να τους υπολογίζει αυτόματα. Τελικά όλα όσα μας ενδιαφέρουν αποθηκεύονται σε ένα collection της βάσης μας.

Αφού ολοκληρωθούν τα παραπάνω οι υπόλοιποι κόμβοι δημιουργούν το τελικό προφίλ της εκτέλεσης, το αποθηκεύουν σε διαφορετικό collection αλλά και σε μορφή csv, ώστε να μπορεί να τροφοδοτηθεί αργότερα στο Weka. Μέσα στον κόμβο Mean Values Query υπολογίζουμε τους γεωμετρικούς μέσους όρους με χρήση javascript και τους αριθμητικούς μέσους όρους με aggreagation pipeline[8] της Mongo.



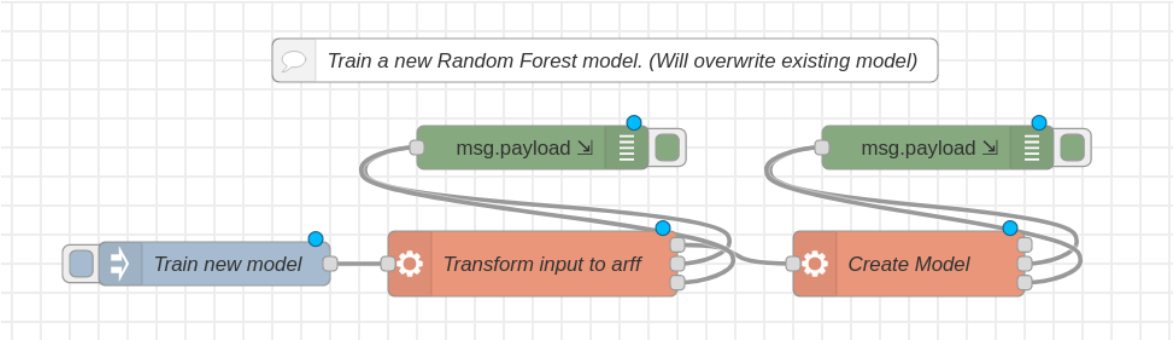
Σχήμα 15: Profile creation nodes

Αφού έχουμε καλέσει τον profiler για όλες τις εκτελέσεις των benchmarks που θέλουμε, είμαστε πλέον σε θέση να δημιουργήσαμε το μοντέλο πρόβλεψής μας. Χρειαζόμαστε αρχικά το αρχείο csv που έχει όλα τα προφίλ που θα ληφθούν υπόψιν και έχουν παραχθεί με το προηγούμενο flow. Το δεύτερο που χρειαζόμαστε είναι το Weka, για το λόγο αυτό χρησιμοποιούμε το jar αρχείο που έχουμε κατεβάσει και μέσω του option -cp της java προσθέτουμε τις κλάσεις του εργαλείου στο classpath της java.

[8] https://www.mongodb.com/docs/manual/aggregation/#std-label-aggregation-pipeline-intro

java -cp /home/pserver/Desktop/weka-3-8-5-azul-zulu-linux/weka-3-8-5/weka.jar weka.core.converters.CSVLoader /home/pserver/Desktop/weka_input/PromModel.csv > /home/pserver/Desktop/weka_input/PromModel.arff

3.2.2 Random Forest Weka Trainer



Σχήμα 16: Random Forest Weka Trainer flow

Το flow αποτελείται από δύο κόμβους που εκτελούν εντολές συστήματος. Ο πρώτος κόμβος μετατρέπει την είσοδο από csv μορφή σε arff ώστε να μπορεί να χρησιμοποιηθεί στους αλγορίθμους μηχανικής μάθησης. Ο δεύτερος κόμβος δημιουργεί το ζητούμενο μοντέλο.

Η εντολή του πρώτου κόμβου είναι η εξής:

java -cp /home/pserver/Desktop/weka-3-8-5-azul-zulu-linux/weka-3-8-5/weka.jar weka.core.converters.CSVLoader /home/pserver/Desktop/weka_input/PromModel.csv> /home/pserver/Desktop/weka_input/PromModel.arff

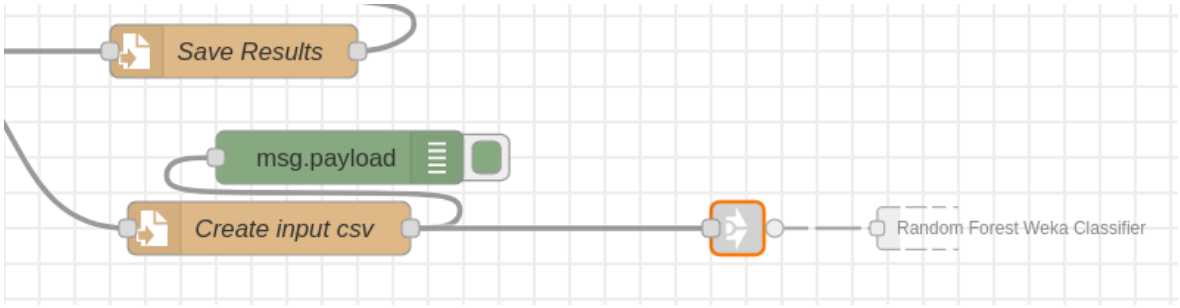
Χρησιμοποιούμε τις κλάσεις του weka για τη μετατροπή και ανακατευθύνουμε την έξοδο στο σημείο που έχουμε ορίσει για να την διαβάσουμε στο επόμενο βήμα

Η εντολή του δεύτερου κόμβου είναι η παρακάτω:

java -cp /home/pserver/Desktop/weka-3-8-5-azul-zulu-linux/weka-3-8-5/weka.jar weka.classifiers.trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1 -t /home/pserver/Desktop/weka_input/PromModel.arff -d /home/pserver/Desktop/weka_input/Prom.model

Με την ίδια λογική παράγουμε το μοντέλο και το αποθηκεύουμε σε ένα αρχείο με επέκταση .model. Τα διάφορα options έχουν οριστεί είτε στην αναζήτηση της βέλτιστης επιλογής από το Weka είτε στην προεπιλογή αφού δεν παρατηρήθηκε κάποια βελτίωση κάνοντας κάποια αλλαγή.

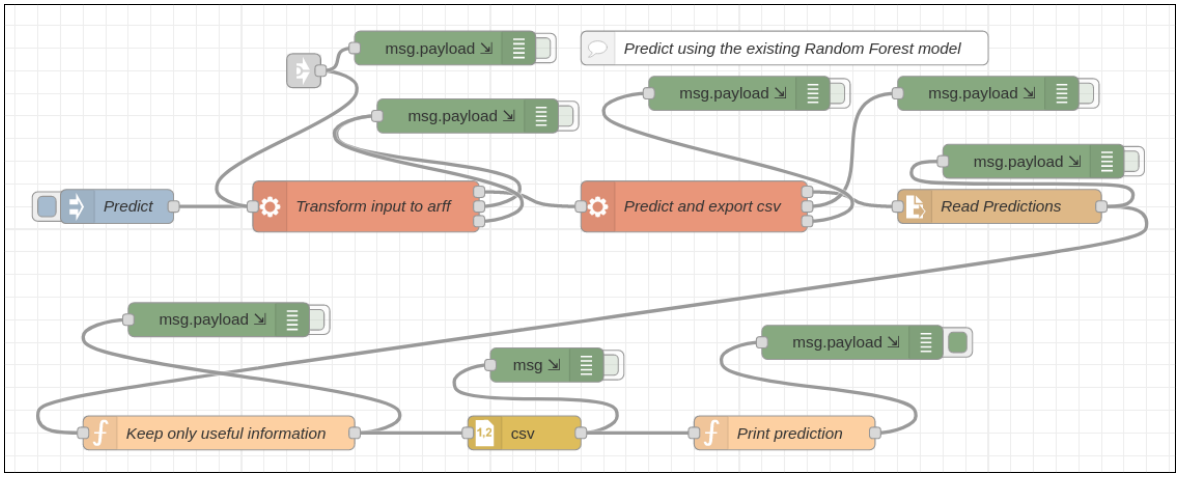
3.2.3 Application Profile Exctractor



Σχήμα 17: Input creation nodes

Το flow αυτό είναι μία επέκταση του Docker Benchmark Profile Exctractor. Είναι ακριβώς το ίδιο με τη διαφορά ότι στο τέλος δημιουργεί επιπλέον ένα αρχείο csv με το προφίλ που υπολόγισε, το οποίο θα χρησιμοποιηθεί ως είσοδος στο μοντέλο μας για να γίνει η πρόβλεψη. Στη συνέχεια πυροδοτεί την έναρξη του επόμενου flow στο οποίο θα πραγματοποιηθεί η πρόβλεψη.

3.2.4 Random Forest Weka Classifier



Σχήμα 18: Random Forest Weka Classifier flow

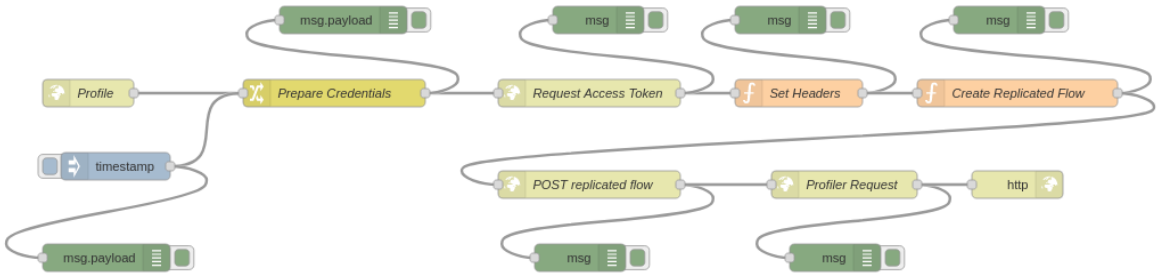
Στο flow αυτό πραγματοποιούνται οι προβλέψεις μας. Αρχικά γίνεται η μετατροπή της εισόδου όπως και προηγουμένως σε μορφή arff με την εντολή

java -cp /home/pserver/Desktop/weka-3-8-5-azul-zulu-linux/weka-3-8-5/weka.jar weka.core.converters.CSVLoader /home/pserver/Desktop/weka_input/PromIn.csv > /home/pserver/Desktop/weka_input/PromIn.arff

Στη συνέχεια, παράγουμε τις προβλέψεις χρησιμοποιώντας το μοντέλο του προηγούμενου βήματος και αποθηκεύουμε το αποτέλεσμα σε ένα αρχείο csv

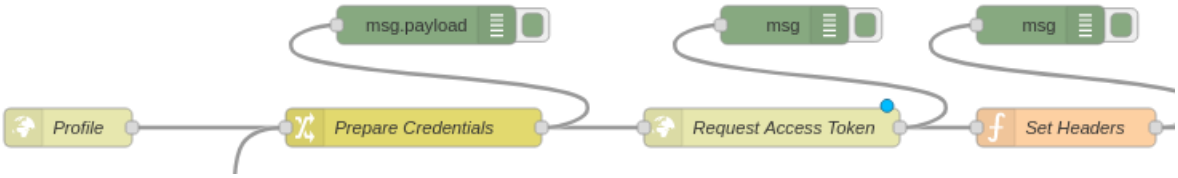
java -cp /home/pserver/Desktop/weka-3-8-5-azul-zulu-linux/weka-3-8-5/weka.jar weka.classifiers.misc.InputMappedClassifier –I -trim -t /home/pserver/Desktop/weka_input/PromModel.arff -T /home/pserver/Desktop/weka_input/PromIn.arff -L /home/pserver/Desktop/weka_input/Prom.model -classifications weka.classifiers.evaluation.output.prediction.CSV > /home/pserver/Desktop/weka_input/PromOut.cs

3.2.5 Requests Handler



Σχήμα 19: Requests Handler flow

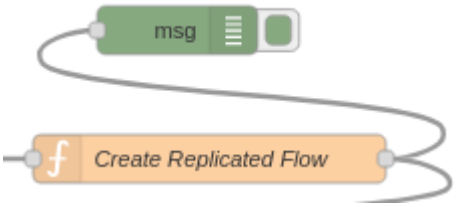
Το τελευταίο flow δίνει τη δυνατότητα της ταυτόχονης εκτέλεσης πολλών αιτημάτων για τη δημιορουργία profiles για πολλές εφαρμογές. Για το σκοπό αυτό αξιοποιεί το Admin API του Node-RED[9] ώστε να τροποποιήσει το flow που παράγει τα προφίλ και να δημιουργήσει ένα ξεχωριστό αντίγραφο που θα εκτελείται παράλληλα. Προκειμένου να χρησιμοποιηθεί το flow αυτό θα πρέπει να αλλάξει το endpoint του flow το οποίο ζητάμε να παραλληλοποιηθεί ώστε να μην ταυτίζεται με αυτό του Request Handler.



Σχήμα 20: Node-RED Admin API usage preparation nodes

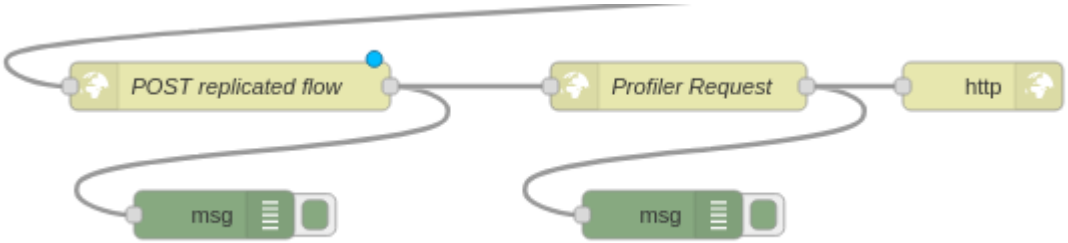
Οι πρώτοι κόμβοι προετοιμάζουν όλη τη διαδικασία. Εφόσον θα χρησιμοποιηθεί το Admin API χρειάζεται η ταυτοποίηση ως admin που σημαίνει ότι πρέπει να παραμετροποιηθούν το όνομα χρήστη και οι αντίστοιχοι κωδικοί.

[9] https://nodered.org/docs/api/admin/methods/



Σχήμα 21: Flow replication node

Στον επόμενο κόμβο υλοποιείται η αντιγραφή του flow παραγωγής των profiles. Όπως έχουμε ήδη αναφέρει τα flows αποθηκεύονται σε μορφή json. Κάθε flow συνοδεύεται από ένα μοναδικό id το οποίο χαρακτηρίζει και τους κόμβους που περιλαμβάνει. Παρέχοντας λοιπόν το json string που περιγράφει το flow που ζητάμε να παραλληλοποιηθεί, αντικαθίσταται το id με ένα καινούριο και ορίζεται ένα καινούριο προσωρινό endpoint.



Σχήμα 22: Add replicated flow new request nodes

Στο τελευταίο κομμάτι του flow πρώτον προστίθεται στα διαθέσιμα το καινούριο flow που ορίσαμε με το καινούριο endpoint. Δεύτερον πραγματοποιείται το request για profiling στο νέο flow που μόλις δημιουργήθηκε. Τα flows είναι τελείως ανεξάρτητα και μπορούμε να φτιάξουμε τόσα όμοια όσα μας επιτρέπουν οι διαθέσιμοι πόροι.

3.2.6 Περιγραφή API

Αναφέρουμε συνοπτικά τις λεπτομέρειες των δύο endpoints όπως προκύπτουν από τα παραπάνω

Endpoint
/monitor
Περιγραφή
Εξάγει το προφίλ από ένα docker container που εκτελεί μία εφαρμογή benchmark
Τύπος request
POST
Σώμα του request

{
"cid": το id του container που τρέχει το benchmark για το οποίο θέλουμε το προφίλ
"bname": ένα όνομα που αναφέρεται στο benchmark και θα συνοδεύει το προφίλ
}

Απάντηση
200, Profiling of the application started successfully

Πίνακας 1: /monitor endpoint details

Endpoint
/profile
Περιγραφή
Εξάγει το προφίλ από ένα docker container που εκτελεί μία εφαρμογή και πυροδοτεί τη διαδικασία της πρόβλεψης
Τύπος request
POST
Σώμα του request

{
"add": η διεύθυνση στην οποία εκτελείται η εφαρμογή για την οποία ζητάμε το προφίλ
"port": η πόρτα στην οποία ακούει το dockerd για την εφαρμογή για την οποία ζητάμε το προφίλ
"cid": το id του container που τρέχει το benchmark για το οποίο θέλουμε το προφίλ
"bname": ένα όνομα που αναφέρεται στο benchmark και θα συνοδεύει το προφίλ
}

Απάντηση
200, Profiling of the application started successfully

Πίνακας 2: /profile endpoint details

4 Πειραματισμός και αξιολόγηση αλγορίθμων κατηγοριοποίησης

Η διαδικασία του πειραματισμού ώστε να μελετηθεί η ακρίβεια των προβλέψεων απαιτεί δύο πράγματα. Πρώτον να οριστεί η μορφή των προφίλ, δηλαδή ποιες μετρικές θα συνιστούν το διάνυσμα που αντιπροσωπεύει μία εφαρμογή. Δεύτερον, να επιλεχθεί ποιος αλγόριθμος μηχανικής μάθησης θα χρησιμοποιείται κατά τη διαδικασία της πρόβλεψης.

4.1 Περιγραφή πειραμάτων

Για να καταφέρουμε να δημιουργήσουμε το μοντέλο προβλέψεων χρειαζόμαστε ένα επαρκές σύνολο δεδομένων. Η διαδικασία του πειραματισμού που ακολουθήθηκε είναι η εξής. Αρχικά επιλέξαμε ένα μεγάλο σύνολο από διαφορετικής φύσης benchmarks από την σουίτα Phoronix Test Suite. Πιο συγκεκριμένα, επιλέξαμε περίπου 40 benchmarks, τα οποία προσομοιάζουν χαρακτηριστικές λειτουργίες ενός υπολογιστή, όπως για παράδειγμα είναι η συμπίεση αρχείων, η μεταγλώττιση προγραμμάτων, η κωδικοποίηση αρχείων ήχου κ.ά. Επίσης επιλέξαμε 4 benchmarks από τη συλλογή sysbench για linux τα οποία επιδέχονται παραμετροποίηση ως προς τον φόρτο εργασίας. Προέκυψαν έτσι από τη συλλογή αυτή 21 διαφορετικοί συνδυασμοί benchmark και φόρτου εργασίας, κάθε ένας εκ των οποίων αντιμετωπίζεται αρχικά σαν ξεχωριστό benchmark.

Εν συνεχεία, ορίσαμε 5 διαφορετικά περιβάλλοντα εκτέλεσης, εκ τω οποίων τα 3 διαφοροποιούνται από την υλική τους υποδομή, ενώ τα υπόλοιπα 2 ήταν εικονικές μηχανές. Η μία εικονική μηχανή δημιουργήθηκε σε ένα από τα 3 μηχανήματα, ενώ η δεύτερη φιλοξενήθηκε στην ακαδημαϊκή υπηρεσία για cloud computing Okeanos [10] Στους πίνακες 3, 4, 5, 6 και 7 παρατίθενται λεπτομερώς οι τεχνικές λεπτομέρειες για κάθε περίπτωση. Σε κάθε μηχάνημα, πραγματικό ή

[10] https://okeanos.grnet.gr/home/

εικονικό, εγκαταστήσαμε το λειτουργικό σύστημα Ubuntu LTS (20.04) [11]και την πλατφόρμα του Docker.

Configuration A

cpu	Intel® Core™ i7-3820 CPU @ 3.6GHz
RAM	32 GB (4x Nanya NT8GC64B8HB0N)
Disk	256 GB (ADATA SU800)

Πίνακας 3: Hardware Configuration A

Configuration B

cpu	AMD FX-8370 Eight-Core Processor
RAM	16 GB (2x Mushkin 992125R)
Disk	256 GB (Samsung SSD 850)

Πίνακας 4: Hardware Configuration B

Configuration C

VM hosted on academic cloud service	
cpu	2 cores with 1 thread from an Intel® Xeon® CPU E5-2650 v3 @ 2.30GHz
RAM	4 GB

Πίνακας 5: Hardware Configuration C

[11] https://releases.ubuntu.com/20.04/

Configuration D

cpu	Intel® Core™ i5-3570 CPU @ 3.4GHz
RAM	16 GB (4x AMI CMZ8GX3M2A1600C9)
Disk	256 GB (Samsung SSD 850)

Πίνακας 6: Hardware Configuration D

Configuration E

VM created in configuration	
cpu	2 out of 4 physical cores
RAM	8 GB out of 16 GB RAM
Disk	100 GB out of 256 GB Disk

Πίνακας 7: Hardware Configuration E

Ακολούθως, προβήκαμε στις εκτελέσεις των benchmarks, την καταγραφή των μετρικών μέσω του docker stats και την παραγωγή των προφίλ. Κάναμε πολλαπλές εκτελέσεις για όλα τα benchmarks σε όλα μηχανήματα μέσα σε απλά docker containers που περιλαμβάνουν μία πιο μικρή έκδοση του Ubuntu Linux, τη σουίτα με τα benchmark και τυχόν dependencies. Από το περιβάλλον A, B και C σχηματίσαμε το training dataset μας με λίγο περισσότερες από 1000 εγγραφές. Σε αυτό το σύνολο δεδομένων, περίπου το 70% προέρχεται από το configuration A, το 15% από το B και το υπόλοιπο 15% από το C. Από τα configurations D και E με ποσοστά 80% και 20% προέκυψε ένα δεύτερο dataset το οποίο χρησιμοποιήσαμε για την φάση του testing, με περίπου 250 εγγραφές. Τα παραπάνω φαίνονται και στον πίνακα 8. Κάθε εκτέλεση ορίστηκε να διαρκεί 10 λεπτά. Τα δεδομένα μέσω του docker stats τραβήχτηκαν ανά 10 δευτερόλεπτα. Στις περιπτώσεις που η εκτέλεση του benchmark ολοκληρωνόταν πριν το ορισμένο χρονικό διάστημα, ρυθμίστηκε η

επανάληψη του υπό τις ακριβώς ίδιες συνθήκες. Η λούπα των εκτελέσεων αυτή τερματιζόταν μετά το πέρας των 10 λεπτών. Τέλος, για όποιο benchmark παρείχε ειδικές επιλογές για την εκτέλεσή του, πάντα επιλέχθηκαν οι προεπιλεγμένες.

Training Dataset	**External Testing Dataset**
Configuration A: 70%	Configuration D: 80%
Configuration B: 15%	Configuration E: 20%
Configuration C: 15%	
Συνολικός αιρθμός εγγραφών 1050 από 63 benchmarks	Συνολικός αιρθμός εγγραφών 250 από 63 benchmarks

Πίνακας 8: Training and Testing Dataset

4.2 Ορισμός διανύσματος προφίλ

Το API του docker με το endpoint /stats επιστρέφει έναν πάρα πολύ μεγάλο αριθμό από μετρικές για το container για το οποίο τις ζητάμε. Το πρώτο βήμα λοιπόν, ήταν να επιλέξουμε ένα υποσύνολο από αυτές που αφορούν στην συμπεριφορά της εκτέλεσης ως προς τη CPU, τη μνήμη, τις διαδικασίες εισόδου/εξόδου και το δίκτυο. Ταυτόχρονα προσπαθήσαμε να κρατήσουμε μόνο τις μετρικές που οι ίδιες ή η μεταβολή τους θα μπορούσαν να χαρακτηριστούν με μία γενικότερη προσέγγιση, ως ανεξάρτητες του υλικού. Έτσι για παράδειγμα ο χρόνος πραγματικής χρησιμοποίησης της CPU από την εφαρμογή προφανώς εξαρτάται από το μοντέλο της, ωστόσο η μεταβολή στην χρησιμοποίηση, δηλαδή το πότε και πόσο απαιτητική σε υπολογισμούς γίνεται η εφαρμογή θα πρέπει να εμφανίζει κάποια ομοιότητα ανεξαρτήτως του που τρέχει, υπό την προϋπόθεση ότι οι αναγκαίοι πόροι επαρκούν. Προκύπτει έτσι ένας πολλαπλασιασμός των διαθέσιμων μετρικών, αφού με την παραπάνω λογική, δημιουργήσαμε για όσες μετρικές έχει νόημα, τον αριθμητικό μέσο όρο, το γεωμετρικό μέσο όρο, τον αριθμητικό μέσο όρο της μεταβολής και το γεωμετρικό μέσο όρο της μεταβολής, οι οποίοι συμβολίζονται ως

avg_(metric_name), geo_avg_(metric_name), avg_delta_(metric_name) και geo_avg_delta_(metric_name) αντίστοιχα. Πέρα από τους αριθμητικούς μέσους όρους παρηγάγαμε και τους γεωμετρικούς διότι η αξία τους προκύπτει από τη δυνατότητα σύγκρισης και με αυτόν τον τρόπο καταφέρνουμε να μετριάσουμε την επίδραση των outliers, δηλαδή των τιμών που δεν φαίνεται να ακολουθούν το γενικότερο μοτίβο της μεταβλητής και απλώς επηρεάζουν σε μεγάλο βαθμό το αποτέλεσμα. Οι τιμές αυτές μπορεί να προκύπτουν από κάποιο τυχαίο και μη προβλέψιμο γεγονός κατά τη διάρκεια της εκτέλεσης ή απλώς να μην έχουν αξία για την πρόβλεψη. Το σύνολο των μετρικών που αποθηκεύσαμε για κάθε προφίλ είναι το παρακάτω:

avg_delta_cpu geo_avg_delta_cpu	Η τιμή της μετρικής cpu ισούται κάθε στιγμή με το συνολικό χρόνο σε nanoseconds τον οποίο χρησιμοποιήθηκε η cpu από τον container.
avg_tasks geo_avg_tasks avg_delta_tasks	Η τιμή της μετρικής tasks ισούται κάθε στιγμή με τον αριθμό των νημάτων που λειτουργούν παράλληλα εντός του container.
avg_memory avg_delta_memory geo_avg_memory	Η τιμή της μετρικής memory ισούται κάθε στιγμή με τον αριθμό των bytes που χρησιμοποιούνται στη μνήμη από τον container.
rx_bytes avg_rx_bytes geo_avg_rx_bytes avg_delta_rx_bytes geo_avg_delta_rx_bytes	Η τιμή της μετρικής rx_bytes ισούται κάθε στιγμή με τον τον αριθμό των bytes που έχει λάβει ο container.

tx_bytes avg_tx_bytes avg_delta_tx_bytes geo_avg_tx_bytes geo_avg_delta_tx_bytes	Η τιμή της μετρικής tx_bytes ισούται κάθε στιγμή με τον τον αριθμό των bytes που έχει στείλει ο container.
rx_packets avg_rx_packets avg_delta_rx_packets geo_avg_rx_packets geo_avg_delta_rx_packets	Η τιμή της μετρικής rx_packets ισούται κάθε στιγμή με τον τον αριθμό των πακέτων που έχει λάβει ο container.
tx_packets avg_tx_packets avg_delta_tx_packets geo_avg_tx_packets geo_avg_delta_tx_packets	Η τιμή της μετρικής tx_packets ισούται κάθε στιγμή με τον τον αριθμό των πακέτων που έχει στείλει ο container.
io_read avg_io_read avg_delta_io_read geo_avg_io_read geo_avg_delta_io_read	Η τιμή της μετρικής io_read ισούται κάθε στιγμή με το σύνολο των bytes που έχουν γραφτεί στο δίσκο από τον container.
io_write avg_io_write geo_avg_io_write avg_delta_io_write geo_avg_delta_io_write	Η τιμή της μετρικής io_read ισούται κάθε στιγμή με το σύνολο των bytes που έχουν διαβαστεί από το δίσκο από τον container.

Πίνακας 9: Profile Metrics

Οι μετρικές που θα επιλέξουμε θα πρέπει να πληρούν τα εξής τρία κριτήρια ώστε να μπορούν να αξιοποιηθούν από τον μοντέλο μας

Α) Σταθερότητα

Αν θέλουμε να έχει νόημα να αναφερόμαστε σε προβλέψεις, θα πρέπει οι μετρικές στις οποίες αυτές βασίζονται, να έχουν μία όσο γίνεται πιο σταθερή συμπεριφορά ανάμεσα στις διαφορετικές εκτελέσεις των ίδιων benchmarks και προγραμμάτων.

Β) Διαφοροποίηση

Σε συνέχεια του προηγούμενου, θα πρέπει οι εκτελέσεις των διαφορετικών benchmarks και προγραμμάτων να μπορούν να διαφοροποιούνται. Θα προτιμήσουμε δηλαδή τις μετρικές εκείνες για τις οποίες παρατηρείται μία σταθερότητα στην τιμή για τις εκτελέσεις των ίδιων benchmarks, αλλά και σημαντική διαφορά για τις εκτελέσεις των διαφορετικών. Όσο πιο “διακριτή” είναι η διαφορά αυτή, τόσο πιο “εύκολο” θα είναι για τον αλγόριθμο να αποφασίσει ανάμεσα σε δύο διαφορετικά υποψήφια προφίλ πρόβλεψης.

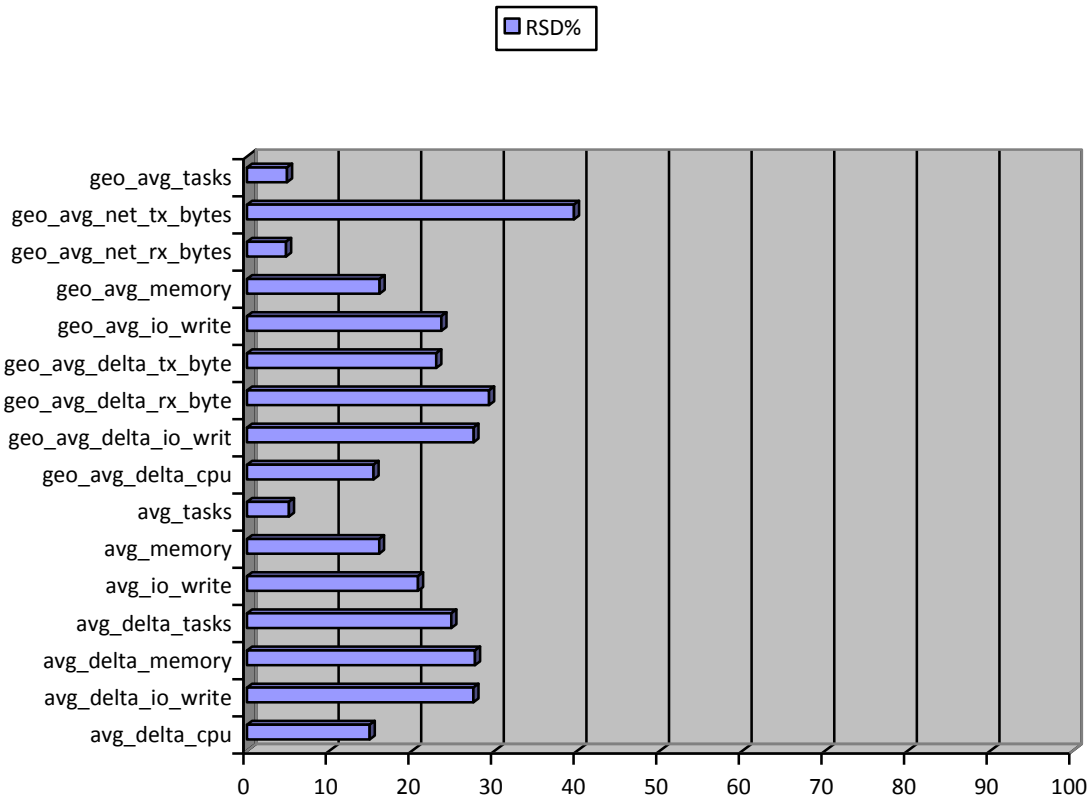
Γ) Ανεξαρτησία

Ζητάμε να κρατήσουμε τις μετρικές εκείνες για τις οποίες τα προηγούμενα δύο κριτήρια ικανοποιούνται ακόμα και σε εκτελέσεις διαφορετικών μηχανημάτων. Με τον τρόπο αυτό θα μπορούμε να αποφανθούμε για την αντιστοίχιση του προφίλ της εφαρμογής χωρίς να νοιαζόμαστε για το που αυτή εκτελείται.

Προκειμένου να μελετήσουμε όσα προαναφέρθηκαν, συγκεντρώσαμε όλα τα προφίλ όπως προέκυψαν από τις εκτελέσεις σε 3 διαφορετικά μηχανήματα όπως περιγράφονται στο κεφάλαιο 4.1. Χωρίζοντας σε ομάδες τα αποτελέσματά ανά benchmark, υπολογίσαμε αρχικά την σχετική τυπική απόκλιση (RSD) η οποία ορίζεται ως εξής:

$$\text{RSD} = \frac{Τυπική\ Απόκλιση}{Μέση\ Τιμή} * 100\%$$

Το μέγεθος αυτό εκφράζει το κατά πόσο οι τιμές μιας μεταβλητής βρίσκονται κοντά στη μέση τιμή, ή απλώνονται σε ένα ευρύτερο φάσμα τιμών. Μία μικρή σχετική τυπική απόκλιση υποδηλώνει ότι τα σημεία των δεδομένων είναι “μαζεμένα” και συνεπώς τείνουν πιο πολύ στο να ικανοποιούν το πρώτο κριτήριο της σταθερότητας. Θα θέλαμε ιδανικά οι σχετικές τυπικές αποκλίσεις όλων των μετρικών για όλες τις ομάδες των benchmarks να είναι όσο χαμηλότερες γίνεται. Προφανώς οι διαφορετικές εκτελέσεις των ίδιων προγραμμάτων δεν μπορούν να ταυτίζονται απολύτως. Σύμφωνα με αυτήν τη λογική, για κάθε μετρική υπολογίζουμε τον αριθμητικό μέσο όρο των RSD που υπολογίσαμε για κάθε ομάδα. Ένας χαμηλός μέσος όρος φανερώνει ότι οι τιμές μίας μετρικής για κάθε πρόγραμμα κυμαίνονται γύρω από το μέσο όρο της ομάδας και συνεπώς τείνει να είναι γενικά πιο σταθερή. Παραθέτουμε τις επιλεγμένες μετρικές που θα συνθέτουν το διάνυσμα των προφίλ μαζί με τα ποσοστά των υπολογισμένων σχετικών τυπικών αποκλίσεων.



Γράφημα 1:RSDs επιλεγμένων μετρικών

Το διάνυσμά μας λοιπόν θα έχει 16 συνιστώσες. Παρατηρούμε ότι τα RSD κυμαίνονται γενικά μεταξύ του 10 με 30%. Σε μία περίπτωση φτάνει και μέχρι το 40%. Οι τιμές αυτές μπορεί να φαίνονται αρκετά υψηλές σύμφωνα με όσα περιγράψαμε μέχρι τώρα, ωστόσο μέσω του πειραματισμού και την αξιολόγηση των προβλέψεων, θα φανερωθεί ότι είναι αποδεκτά ποσοστά. Αυτό οφείλεται και στη λειτουργία του αλγορίθμου που τελικά επιλέχθηκε (Random Forest), όπως αναλυτικά θα επεξηγηθεί στο επόμενο κεφάλαιο. Ιδιαίτερη σημασία έχει ότι τα αποτελέσματα από τα οποία προέκυψαν τα RSD προέρχονται από ετερογενή συστήματα και εξ ορισμού δεν θα μπορούσαν να είναι κοντά στο 0. Η σταθερότητα μεταξύ τους όμως είναι αρκετή ώστε να αποκτά αξία για τη διαδικασία της πρόβλεψης.

4.3 Επιλογή αλγορίθμου πρόβλεψης

Η μηχανική μάθηση είναι μία κατηγορία αλγορίθμων, οι οποίοι προσπαθούν να “μάθουν” από τα δεδομένα που τους δίνουμε. Δημιουργούν έτσι μοντέλα βασισμένα σε ένα σύνολο δεδομένων, με στόχο να είναι σε θέση να κάνουν προβλέψεις για νέα δεδομένα τα οποία δεν έχουν ξαναδεί. Στη δική μας περίπτωση αυτό που προσπαθούμε να προβλέψουμε είναι τα προφίλ των εφαρμογών. Θέλουμε για ένα τυχαίο προφίλ να είμαστε σε θέση να προβλέψουμε ή καλύτερα να αντιστοιχίσουμε με ποιο προφίλ από αυτά του μοντέλου παρουσιάζει την πιο κοντινή συμπεριφορά σε ό,τι αφορά τις απαιτήσεις σε πόρους. Προφανώς δεν θα μπορούσαμε να το ελέγξουμε αυτό, για το λόγο αυτό όπως προαναφέρθηκε έχουμε δημιουργήσει δύο σύνολα δεδομένων, το training dataset και το test dataset. Με το πρώτο, γνωρίζοντας τα προφίλ, θα δημιουργήσουμε το μοντέλο μηχανικής μάθησης. Κρύβοντας προσωρινά τα προφίλ από το δεύτερο dataset, θα πραγματοποιήσουμε προβλέψεις και έτσι θα μπορούμε επαναφέροντας τα κρυμμένα προφίλ να υπολογίσουμε σε τι ποσοστό οι προβλέψεις ήταν σωστές. Η κατηγορία των αλγορίθμων αυτών ονομάζεται επιβλεπόμενη μάθηση.

Υπάρχουν πολλοί αλγόριθμοι που λειτουργούν με τον τρόπο που θέλουμε και προσπαθούν να κάνουν προβλέψεις. Ο αλγόριθμος που τελικά επιλέχθηκε είναι ο Random Forest. Η επιλογή αυτή βασίστηκε σε δύο κριτήρια, ένα θεωρητικό το οποίο

θα αναπτύξουμε στη συνέχεια και ένα πειραματικό, σύμφωνα με το οποίο δοκιμάσαμε μία πληθώρα διαφορετικών αλγορίθμων και συγκρίναμε τα αποτελέσματα των ποσοστών ορθών προβλέψεων. Για να δικαιολογήσουμε την επιλογή θεωρητικά θα περιγράψουμε τον αλγόριθμο, ωστόσο ο Random Forest βασίζεται στον αλγόριθμο Decision Trees, οπότε αρχικά θα περιγράψουμε αυτόν.

4.3.1 Decision Trees

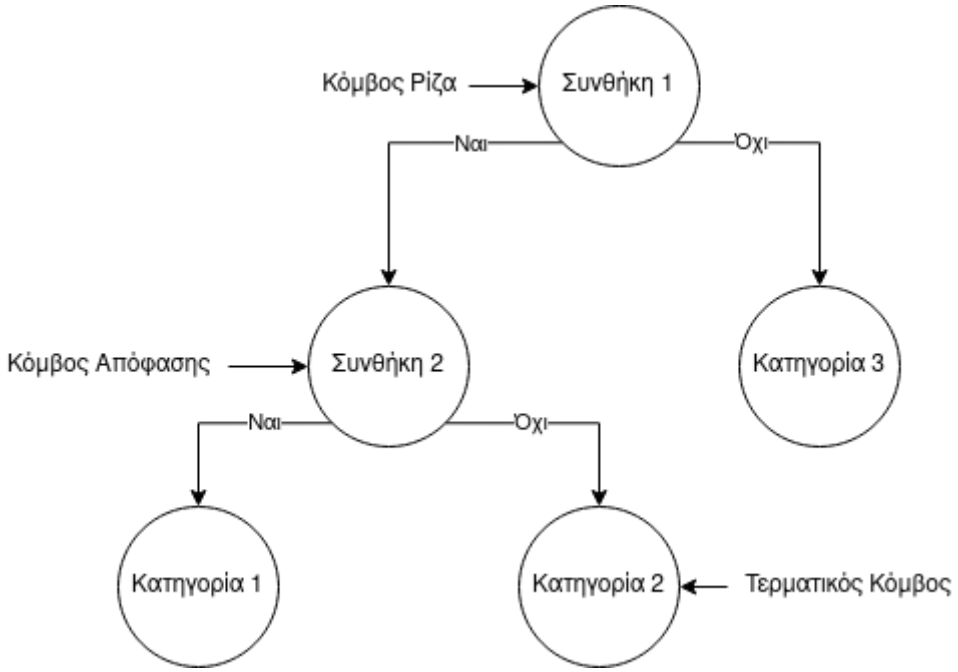
Ένα Decision Tree είναι στην πραγματικότητα ένα απλό δυαδικό δέντρο μέσω του οποίου μπορεί να ταξινομηθεί σε κάποια κατηγορία κάποιο εισερχόμενο δεδομένο. Χωρίζουμε τους κόμβους ενός τέτοιου δέντρου σε 3 κατηγορίες.

Κόμβος – Ρίζα: Αποτελεί τη ρίζα του δέντρου και λειτουργεί σαν σημείο εισόδου του αλγορίθμου. Είναι δηλαδή ο πρώτος κόμβος που πρέπει να επισκεφτεί ένα νέο δεδομένο για να κατηγοριοποιηθεί.

Κόμβος απόφασης: Κάθε κόμβος που ελέγχει ένα κριτήριο του δεδομένου και έχει δύο παιδιά που ακολουθούνται ανάλογα με την απάντηση που έδωσε.

Τερματικός κόμβος: Είναι τα φύλλα του δέντρου και αντιστοιχίζουν το εισερχόμενο δεδομένο σε μία κατηγορία.

Η διαδικασία που ακολουθείται είναι εξής. Ένα καινούριο δεδομένο εισέρχεται στον κόμβο-ρίζα σε μορφή διανύσματος, στην περίπτωση μας αυτό θα ήταν ένα προφίλ μιας εφαρμογής που αναπαρίσταται ως διάνυσμα των μετρικών του. Από τον κόμβο αυτόν και σε κάθε επόμενο, εξετάζεται μία συνθήκη για ένα χαρακτηριστικό του διανύσματος. Ελέγχεται δηλαδή αν η τιμή μιας συνιστώσας είναι μεγαλύτερη ή μικρότερη από ένα κρίσιμο σημείο. Συνεχίζεται η διαδικασία αυτή μέχρι να καταλήξουμε σε ένα τερματικό κόμβο, ο οποίος αντιστοιχεί σε μία κατηγορία, η οποία είναι και η πρόβλεψη του αλγόριθμου.



Σχήμα 23: Decision Tree example

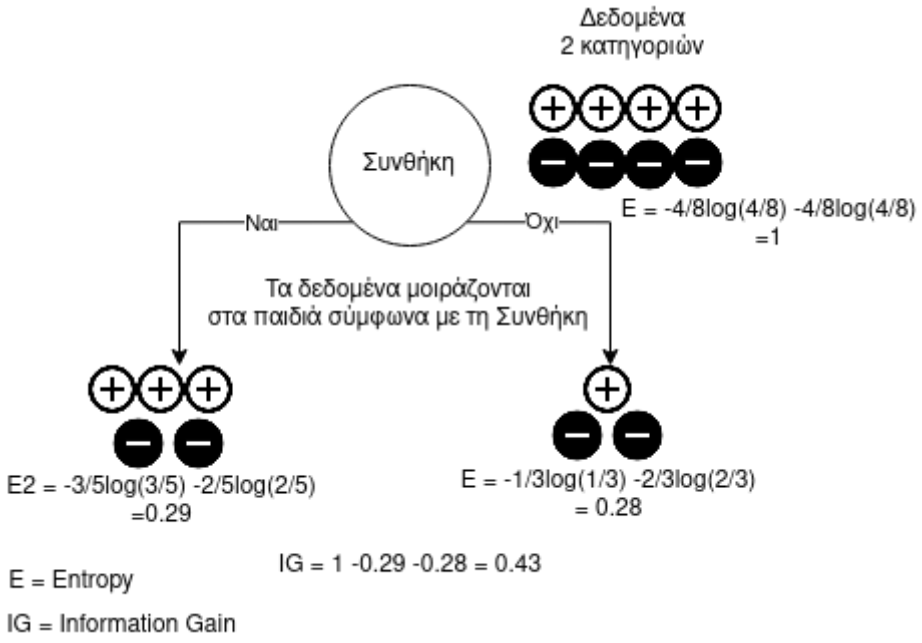
Μέχρι αυτό το σημείο ο αλγόριθμος φαίνεται μάλλον αρκετά απλός και δεν περιέχει κάποιο στοιχείο μηχανικής μάθησης. Το σημαντικό στοιχείο που διαφοροποιεί ένα Decision Tree από μία απλή αλληλουχία από if και else είναι η επιλογή των σημείων των κόμβων απόφασης σύμφωνα με τον οποίο γίνεται ο διαχωρισμός. Η βασική ιδέα είναι ότι θέλουμε από το αποτέλεσμα της απόφασης κάθε κόμβου να αποκομίζουμε όσο περισσότερη πληροφορία για την κατηγοριοποίηση γίνεται. Για να το πετύχουμε αυτό ορίζουμε το μέγεθος της εντροπίας (entropy) το οποίο εκφράζει την ποσότητα της πληροφορίας που χρειαζόμαστε σε κάποιον κόμβο, ώστε να κωδικοποιήσουμε την κατηγοριοποίηση κάθε μέλους του συνόλου που βρίσκεται σε αυτό, ξεκινώντας αρχικά στη ρίζα με το training dataset. Πιο τυπικά η εντροπία ορίζεται ως εξής.

$$Εντροπία = \sum (p_i * \log(p_i))$$

όπου p_i είναι η πιθανότητα της κατηγοριοποίησης στην κλάση i, στα σύνολα όπως προκύπτει από τον διαχωρισμό που επιβάλλει ο κόμβος. Ένας τερματικός κόμβος έχει εντροπία 0.

Τελικά το πληροφοριακό κέρδος (Information Gain) ορίζεται ως τη μείωση της εντροπίας. Πιο συγκεκριμένα ισούται με τη διαφορά του αθροίσματος των εντροπιών των παιδιών ενός κόμβου από την εντροπία του κόμβου αυτού. Με άλλα λόγια, όσο πιο πολύ καταφέρνουμε να μειώσουμε την εντροπία, τόσο περισσότερη πληροφορία κερδίζουμε. Σε κάθε κόμβο λοιπόν, ο αλγόριθμος δοκιμάζει όλες τις πιθανές τιμές για το σημείο διαχωρισμού για όλες τις συνιστώσες τους διανύσματος εισόδου και επιλέγει τελικά εκείνο για το οποίο το πληροφοριακό κέρδος μεγιστοποιείται. Ο τύπος για το πληροφοριακό κέρδος είναι:

$$Πληροφορικό\ Κέρδος = Εντροπία(Κόμβου) - \sum(w_i * Εντροπία(Παιδί_i))$$



Σχήμα 24: Information Gain calculation example

4.3.2 Random Forest

Το πρόβλημα των Decision Trees είναι ότι πολύ εύκολα υπερεκπαιδεύονται (overfitting), είναι ευαίσθητα σε έστω και σε μικρές αλλαγές στα δεδομένα που δημιουργούν το μοντέλο και συνεπώς δεν καταφέρνουν να πετύχουν καλά ποσοστά προβλέψεων για εξωτερικά νέα δεδομένα. Προκειμένου να λυθεί το πρόβλημα αυτό εισάγεται η έννοια του Bagging που είναι ένας συνδυασμός των Bootstrpping και Aggregating. Οι τεχνικές αυτές εισάγουν τον παράγοντα της τυχαιότητας ώστε να υπερβούν το πρόβλημα που περιγράψαμε. Ο αλγόριθμος Random Forest αξιοποιώντας το Bagging δημιουργεί πολλά διαφορετικά Decision Trees προκειμένου να φτάσει στην πρόβλεψη.

Η διαδικασία δημιουργίας ενός Decision Tree στα πλαίσια του Random Forest είναι η εξής. Αρχικά δημιουργούμε ένα Bootstrapped σύνολο δεδομένων με βάση το training dataset μας. Δημιουργούμε δηλαδή ένα ίδιου μεγέθους σύνολο δεδομένων το οποίο γεμίζουμε με τυχαίες εγγραφές επιλεγμένες από το αρχικό σύνολο δεδομένων μας, χωρίς να απαγορεύεται να επιλέξουμε την ίδια εγγραφή παραπάνω από μία φορές. Δημιουργείται έτσι ένα σύνολο δεδομένων στο οποίο προφανώς κάποιες εγγραφές θα απουσιάζουν και δεν θα ληφθούν υπόψιν. Στη συνέχεια, κατασκευάζουμε το Decision Tree από το σύνολο αυτό που προέκυψε, με τη διαφορά ότι σε κάθε βήμα όπου εξετάζουμε την βέλτιστη επιλογή για τη συνθήκη μέσα στον κόμβο, λαμβάνουμε υπόψιν μόνο ένα υποσύνολο από στήλες, δηλαδή συνιστώσες του διανύσματος που στην περίπτωση μας είναι οι μετρικές που απαρτίζουν τα προφίλ των εφαρμογών. Θα επαναλάβουμε αυτήν τη διαδικασία περίπου 100 φορές, αριθμός που εξαρτάται από το εκάστοτε πρόβλημα καταλήγοντας σε Decision Trees κάθε φορά διαφορετικά. Στη συνέχεια αφού έχουμε φτιάξει όλα τα δέντρα, πραγματοποιούμε μία πρόβλεψη με σε κάθε ένα από αυτά. Η τελικά πρόβλεψη προκύπτει με μία απλή ψηφοφορία μεταξύ των δέντρων, όποια πρόβλεψη πάρει τις περισσότερες ψήφους, αυτή θα είναι και η πρόβλεψη του Random Forest.

Με αυτές τις παρατηρήσεις βλέπουμε ότι ο αλγόριθμος φαίνεται θεωρητικά να ταιριάζει στο πρόβλημα μας, αφού όταν εκπαιδευτεί το μοντέλο την πρώτη φορά, όλες οι ακόλουθες προβλέψεις θα αφορούν σε εξωτερικά δεδομένα που θα προέρχονται και από διαφορετικά μηχανήματα. Εξαλείφοντας τον παράγοντα του overfitting σε κάποιον βαθμό αναμένουμε να πετύχουμε σίγουρα καλύτερα αποτελέσματα από το Decision Tree. Προκειμένου σε κάθε κόμβο να μπορούν να παίρνονται αποφάσεις που θα οδηγούν σε σωστή πρόβλεψη, οι μετρικές θα πρέπει να έχουν τα κριτήρια που προαναφέραμε. Ήδη δείξαμε ότι οι εκτελέσεις παραμένουν σταθερές για τα ίδια benchmarks. Αυτό που μένει είναι να εξετάσουμε αν οι εκτελέσεις των διαφορετικών benchmarks διαφοροποιούνται με τέτοιον τρόπο ώστε να μπορεί να αξιοποιηθεί η διαφοροποίηση αυτή μέσω των συνθηκών των κόμβων απόφασης των δέντρων. Αυτό φάνηκε μέσα από τα πειράματα.

4.3.3 Δοκιμές Αλγορίθμων

Με στόχο την επιβεβαίωση της επιλογής του κατάλληλου αλγορίθμου, αλλά και τη σύγκριση με διαφορετικούς αλγορίθμους και ενδεχομένως την επιλογή κάποιου άλλου δημιουργήσαμε ξεχωριστά μοντέλα πρόβλεψης για κάθε έναν από αυτούς. Χρησιμοποιήσαμε τόσο την τεχνική 10-fold cross validation όσο και testing με εξωτερικά δεδομένα όπως περιγράψαμε στην ενότητα 4.1. Σύμφωνα με τη μέθοδο 10-fold cross validation χωρίζουμε το training dataset σε 10 ίδιου μεγέθους υποσύνολα. Σε κάθε βήμα, με τη σειρά κρατάμε στην άκρη το ένα από τα 10 αυτά σύνολα και τα χρησιμοποιούμε ως training dataset για να δημιουργήσουμε ένα μοντέλο πρόβλεψης. Στη συνέχεια χρησιμοποιούμε το υποσυνόλο που κρατήσαμε ως testing dataset και υπολογίζουμε την ακρίβεια των προβλέψεων. Αφού το κάνουμε αυτό και για τις 10 φορές, υπολογίζουμε τη συνολική ακρίβεια ως το μέσο όρο των αποτελεσμάτων των 10 επαναλήψεων. Παράλληλα με την ακρίβεια, υπολογίζουμε το μέσο απόλυτο σφάλμα (MAE), το μέσο τετραγωνικό σφάλμα (RMSE) και το σχετικό απόλυτο σφάλμα (RAE). Τα σφάλματα αυτά ορίζονται με τον κλασικό τρόπο για κατηγορικές μεταβλητές. Παραθέτουμε τη λίστα όλων των αλγορίθμων που δοκιμάσαμε μαζί με τα αντίστοιχα σφάλματα και τις ακρίβειες που προέκυψαν από το 10-fold cross validation.

Algorithm	MAE	RMSE	RAE	Accuracy
Random Forest	0.0056	0.0464	17.8356%	91.8762%
ForestPA	0.0071	0.0526	22.802%	88.8781%
Bayes Network	0.004	0.0498	12.7839%	88.7814%
Random Tree	0.0037	0.0612	11.996%	88.2012%
SPAARC	0.0042	0.0568	13.5238%	87.0406%
PART Decision List	0.0047	0.0619	15.1034%	85.5899%
REPTree	0.0054	0.0586	17.2878%	84.5261%
MODLEM	0.0051	0.0716	16.4207%	83.8491%
HoeffdingTree	0.007	0.0765	22.3548%	77.853%
Naive Bayes	0.0083	0.0719	26.615%	74.8549%
K-nearest neighbours	0.0106	0.0931	33.88%	71.0832%
Functional Trees	0.0109	0.0899	34.7492%	69.5358%
Decision Table	0.0291	0.1167	93.0452%	65.8607%
Logistic Model Trees	0.0189	0.109	60.3713%	55.4159%
Simple Logistic	0.0214	0.1069	68.5321%	54.0619%
Locally Weighted Learning	0.0302	0.1223	96.8212%	26.5957%
DecisionStump	0.0307	0.1239	98.3123%	4.2553%

KStar	0.0311	0.1251	99.4887%	2.5145%

Πίνακας 10: Algorithms cross validation with workloads results

Ακολουθεί η ίδια λίστα με τα αποτελέσματα για τα μοντέλα που προέκυψαν από ολόκληρο το training dataset και τροφοδοτήθηκαν με το εξωτερικό testing dataset.

Algorithm	**MAE**	**RMSE**	**RAE**	**Accuracy**
Random Forest	0.0147	0.078	46.9251%	84.739%
Random Tree	0.0139	0.1179	44.4885%	56.2249%
REPTree	0.0151	0.1156	48.1956%	53.8153%
DecisionStump	0.0307	0.124	98.3726%	2.8112%
ForestPA	0.0149	0.0842	47.7124%	70.6827%
Functional Trees	0.0144	0.1088	46.1002%	57.8313%
HoeffdingTree	0.0179	0.1292	57.1927%	43.3735%
Logistic Model Trees	0.0188	0.126	60.2482%	46.988%
SPAARC	0.0138	0.1157	44.1845%	57.0281%
Bayes Network	0.0142	0.1032	45.4912%	56.6265%
Naive Bayes	0.0173	0.1216	55.2328%	45.7831%
Simple Logistic	0.0223	0.1097	71.2751%	50.2008%

WiSARD	0.4826	0.5617	1544.9795%	53.012%
K-nearest neighbours	0.4832	0.5623	1547.0171%	42.1687%
Kstar	0.0312	0.125	100.0165%	1.6064%
Locally Weighted Learning	0.0303	0.1227	97.1418%	21.2851%
Decision Table	0.0306	0.1228	97.8559%	11.6466%
MODLEM	0.0144	0.12	46.1211%	54.6185%
PART Decision List	0.0166	0.1258	52.9936%	47.7912%

Πίνακας 11: Algorithms external dataset testing with workloads results

Προσπαθώντας να βελτιώσουμε τα παραπάνω αποτελέσματα προβήκαμε στην ομογενοποίηση των datasets μας ως εξής. Για όλα τα προφίλ που παρήχθησαν από τη συλλογή sysbench, δώσαμε το ίδιο όνομα στα προφίλ που προέρχονται από τα ίδια benchmarks με διαφορετικό φόρτο εργασίας. Θεωρήσαμε δηλαδή ότι ένα προφίλ χαρακτηρίζει τη φύση μιας εφαρμογής και δεν διαφοροποιείται από την παραμετροποίηση. Για τα προφίλ που προέρχονται από τη σουίτα Phoronix με την ίδια λογική ομογενοποιήσαμε εφαρμογές που επιτελούν παρόμοιες λειτουργίες, όπως για παράδειγμα συμπίεση αρχείου rar και συμπίεση αρχείου zip ή κωδικοποίηση αρχείου mp3 και κωδικοποίηση αρχείου flac. Παραθέτουμε τα νέα αποτελέσματά για 10 cross fold-validation.

Algorithm	Accuracy
Random Forest	97.4855%
ForestPA	95.8414%
Bayes Network	96.8085%
Random Tree	92.4565%
SPAARC	95.0677%
PART	94.3907%
REPTree	94.5841%
MODLEM	91.0058%
HoeffdingTree	89.1683%
Naive Bayes	92.5532%
K-nearest neighbours	84.1393%
Functional Trees	88.3946%
Decision Table	78.9168%
Logistic Model Trees	72.5338%
Simple Logistic	72.147%
Locally Weighted Learning	28.6267%
DecisionStump	18.5687%
KStar	7.06%

Πίνακας 12: Algorithms cross validation without workloads accuracies

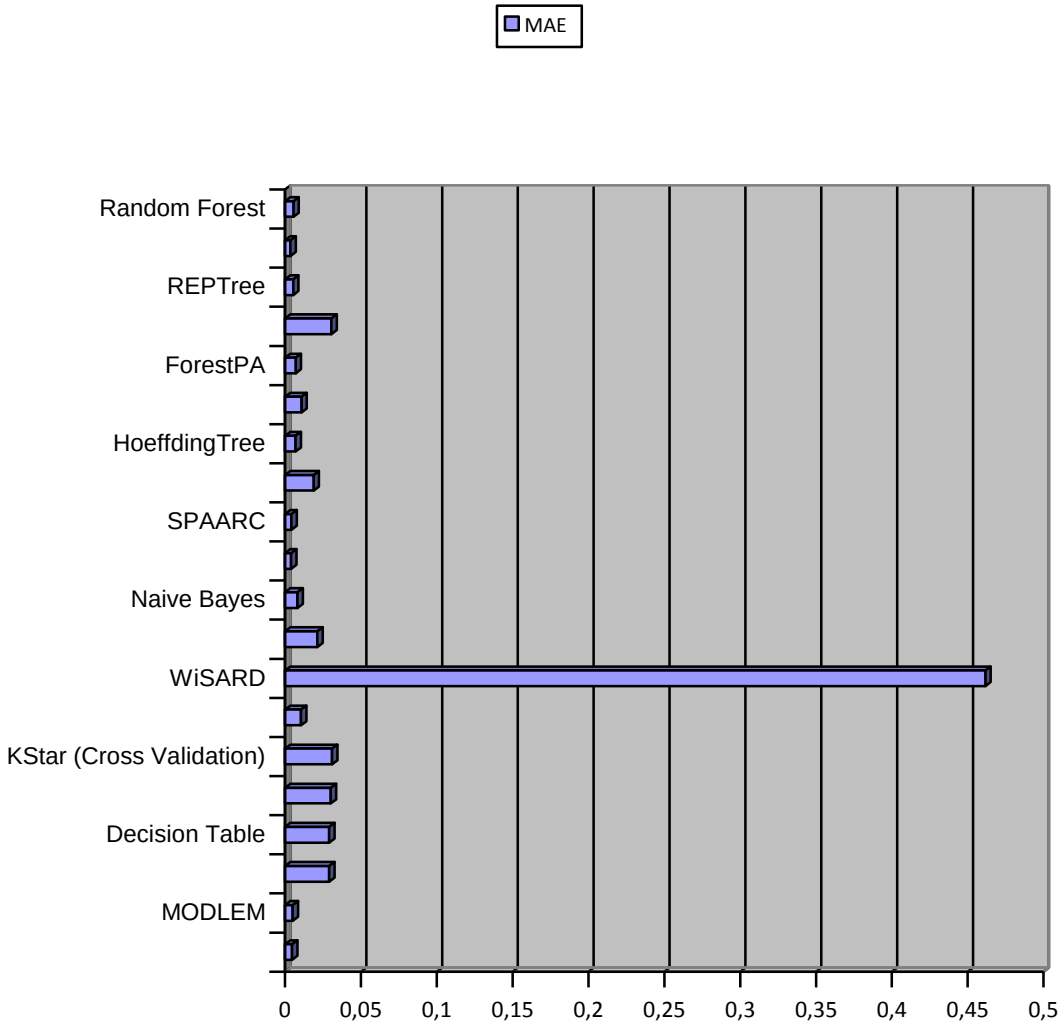
Ακολουθούν τα αποτελέσματα του testing με το εξωτερικό dataset.

Algorithm	Accuracy
Random Forest	89.5582%
Random Tree	66.2651%
REPTree	65.0602%
DecisionStump	14.4578%
ForestPA	75.1004%
Functional Trees	58.2329%
HoeffdingTree	57.8313%
Logistic Model Trees	65.4618%
SPAARC	73.494%
Bayes Network	69.4779%
Naive Bayes	65.0602%
Simple Logistic	65.4618%
WiSARD	68.6747%
K-nearest neighbours	57.4297%
Kstar	4.8193%
Locally Weighted Learning	24.498%
Decision Table	38.9558%
MODLEM	65.8635%
PART Decision List	71.8876%

Πίνακας 13: Algorithms external supplied testing dataset without workloads accuracies

4.4 Αξιολόγηση ακρίβειας αλγορίθμων μηχανικής μάθησης

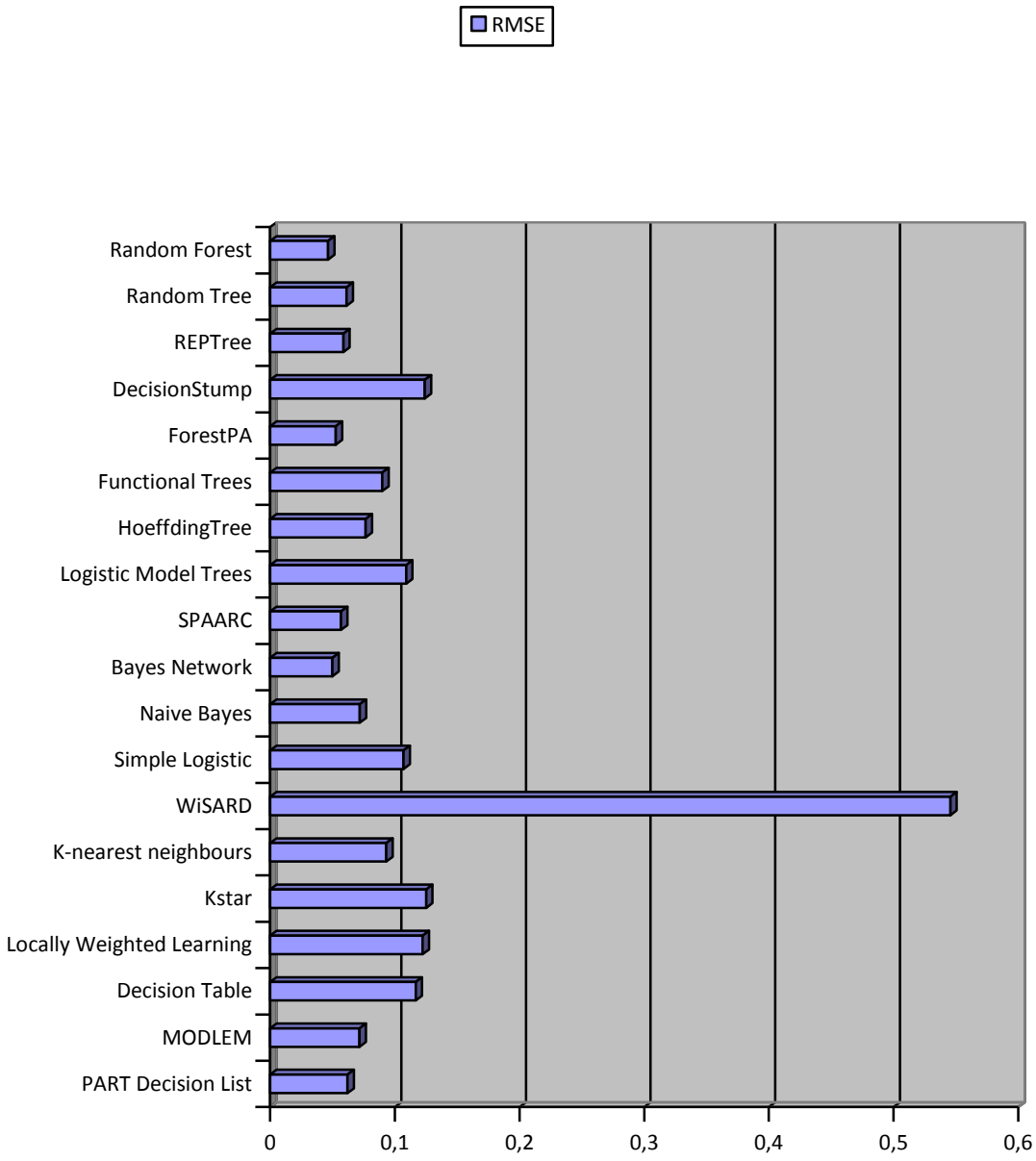
Παρουσιάζουμε τις γραφικές απεικονίσεις όλων των αποτελεσμάτων.



Γράφημα 2: MAE Cross Validation with workload

Βλέπουμε ότι για όλους τους αλγορίθμους το MAE που προκύπτει για τη μέθοδο επαλήθευσης προβλέψεων με cross validation, με εξαίρεση τον αλγόριθμο WiSARD, τα αποτελέσματα είναι σχετικά καλά. Ο WiSARD παρουσιάζει MAE ίσο με 0.4617 ενώ όλοι οι υπόλοιποι αλγόριθμοι δεν ξεπερνούν το 0.0307. Το καλύτερο

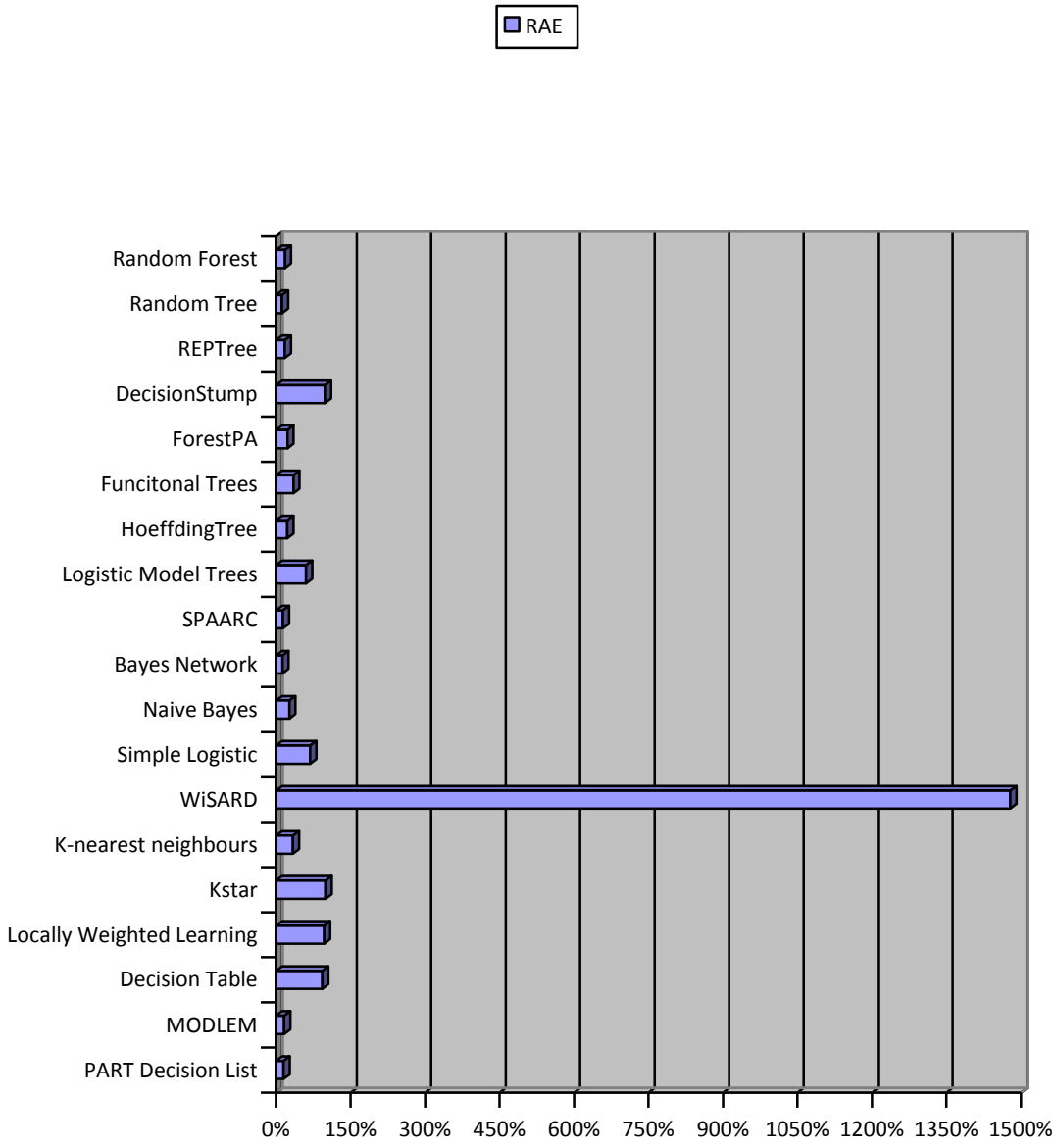
αποτέλεσμα παρουσιάζει ο Random Tree με MAE ίσο με 0.004 ενώ ο Random Forest που μας ενδιαφέρει δεν απέχει πολύ με MAE 0.0056.



Γράφημα 3: RMSE Cross Validation with workload

Βλέπουμε ότι όπως και προηγουμένως, το RMSE παρουσιάζει καλά αποτελέσματα με εξαίρεση και πάλι τον αλγόριθμο WiSARD. Το RMSE του WiSARD

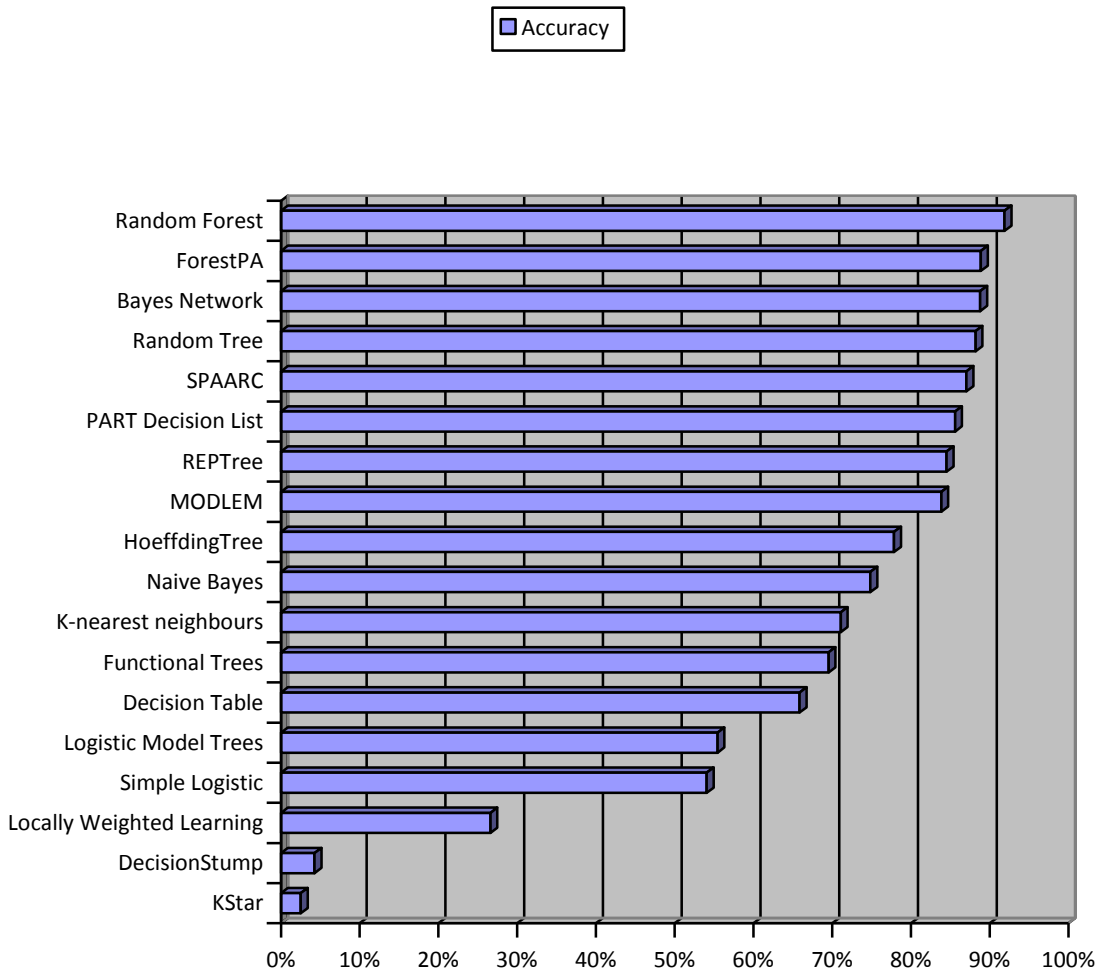
είναι 0.5456 ενώ για όλους του υπόλοιπους αλγόριθμους δεν ξεπερνάει το 0.1251. Το καλύτερο αποτέλεσμα παρουσιάζει ο K-Star με RMSE ίσο με 0.1251 ενώ ο Random Forest ίσο με 0.0464.



Γράφημα 4: RAE Cross Validation with workload

Συνεχίζοντας με το RAE, βλέπουμε παρόμοια συμπεριφορά όπως είναι και λογικό άλλωστε. Ο αλγόριθμος WiSARD ξεχωρίζει αυτήν την φορά με αρκετά μεγάλη διαφορά φτάνοντας στο 1478.70%. Οι υπόλοιποι αλγόριθμοι ωστόσο αυτήν

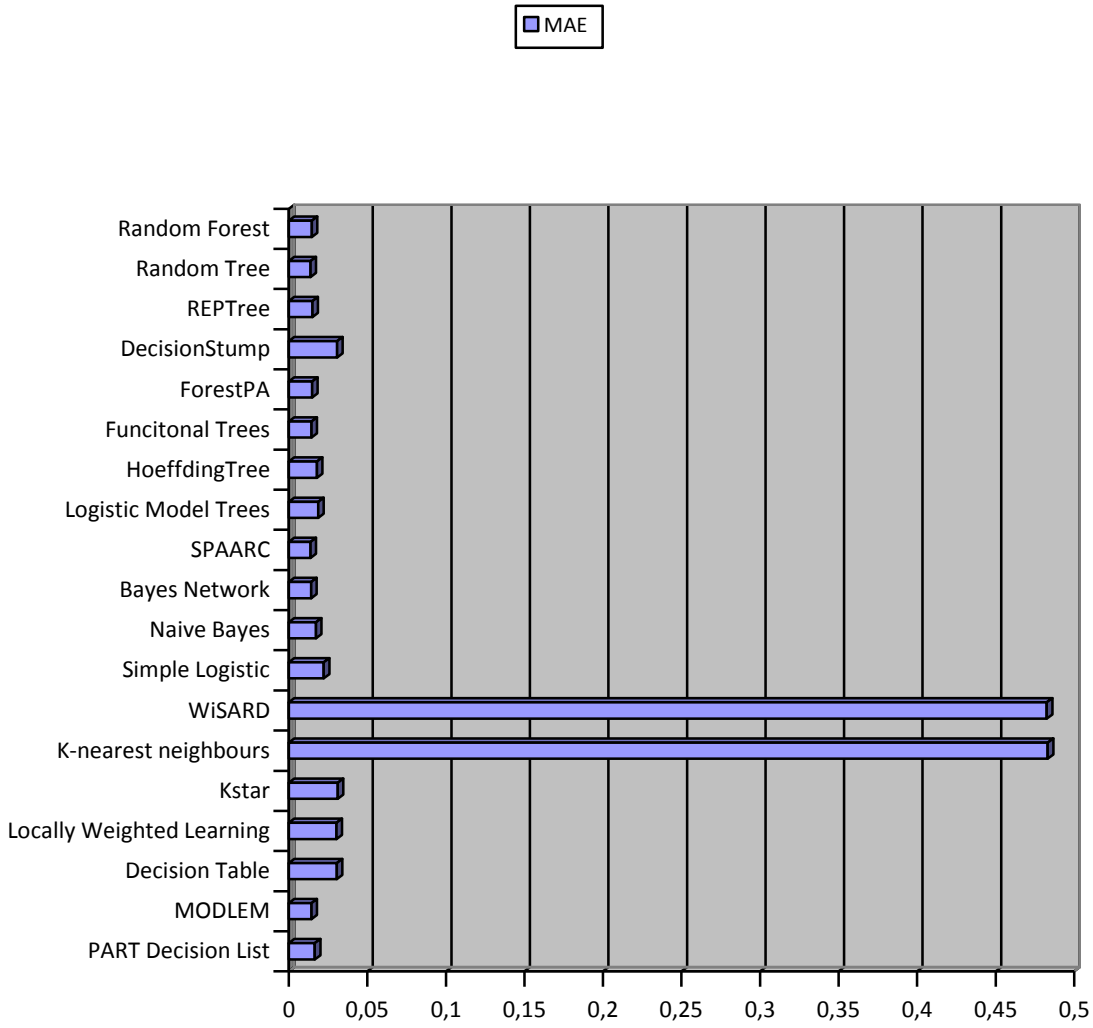
την φορά παρουσιάζουν πιο αισθητές διαφορές με τους DecisionStump, K-Star, Locally Weighted Learning και Decision Table να ξεπερνάνε το 90%. Οι ForestPA, Functional Trees, Hoeffding Tree, Logistic Model Trees, Naïve Bayes, Simple Logistic και K-nearest κυμαίνονται από 22.80% έως 99.49%. Όλοι οι υπόλοιποι αλγόριθμοι παρουσιάζουν RAE κάτω από 20%. Ο Random Forest δηλαδή βρίσκεται και πάλι στην καλύτερη κατηγορία αποτελεσμάτων με RAE 17.84% το οποίο δεν απέχει από το καλύτερο αποτέλεσμα 12.00% του Random Tree.



Γράφημα 5: Accuracies Cross Validation with workload

Παρατηρώντας τα accuracies που είναι και το μέτρο ορθών προβλέψεων βλέπουμε ότι ο αλγόριθμος Random Forest υπερτερεί όλων με ποσοστό ακρίβειας

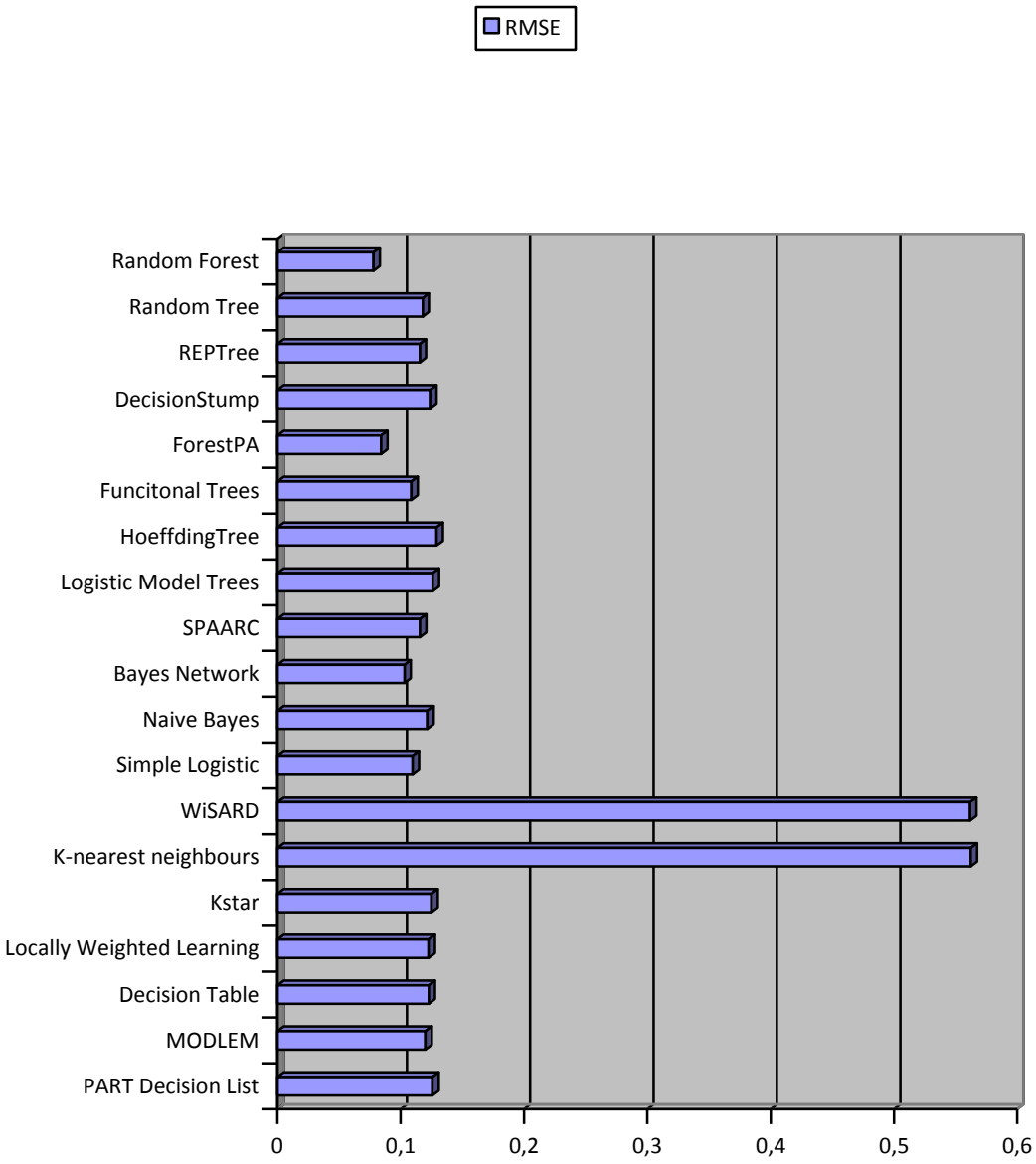
προβλέψεων 91.88%. Αρκετά καλά αποτελέσματα παρουσιάζουν και οι ForestPA, Bayes Network Random Tree με ποσοστά πάνω από 88%. Στη συνέχεια τα αποτελέσματα σταδιακά φθίνουν για τους υπόλοιπους αλγόριθμους. Μέχρι και αυτό το σημείο λοιπό βλέπουμε ότι με τη μέθοδο Cross Validation ο Random Forest έχει το μεγαλύτερο ποσοστό ορθών προβλέψεων ενώ όλα τα σφάλματά του βρίσκονται κοντά στην καλύτερη επίδοση σε σχέση με τους υπόλοιπους αλγόριθμους.



Γράφημα 6: MAE Supplied Test Set with workload

Ξεκινώντας με τα διαγράμματα για τον έλεγχο προβλέψεων με εξωτερικά δεδομένα βλέπουμε ότι και πάλι ο αλγόριθμος WiSARD μαζί αυτήν την φορά με τον K-nearest neighbours έχουν το υψηλότερο MAE. Το καλύτερο MAE έχει ο

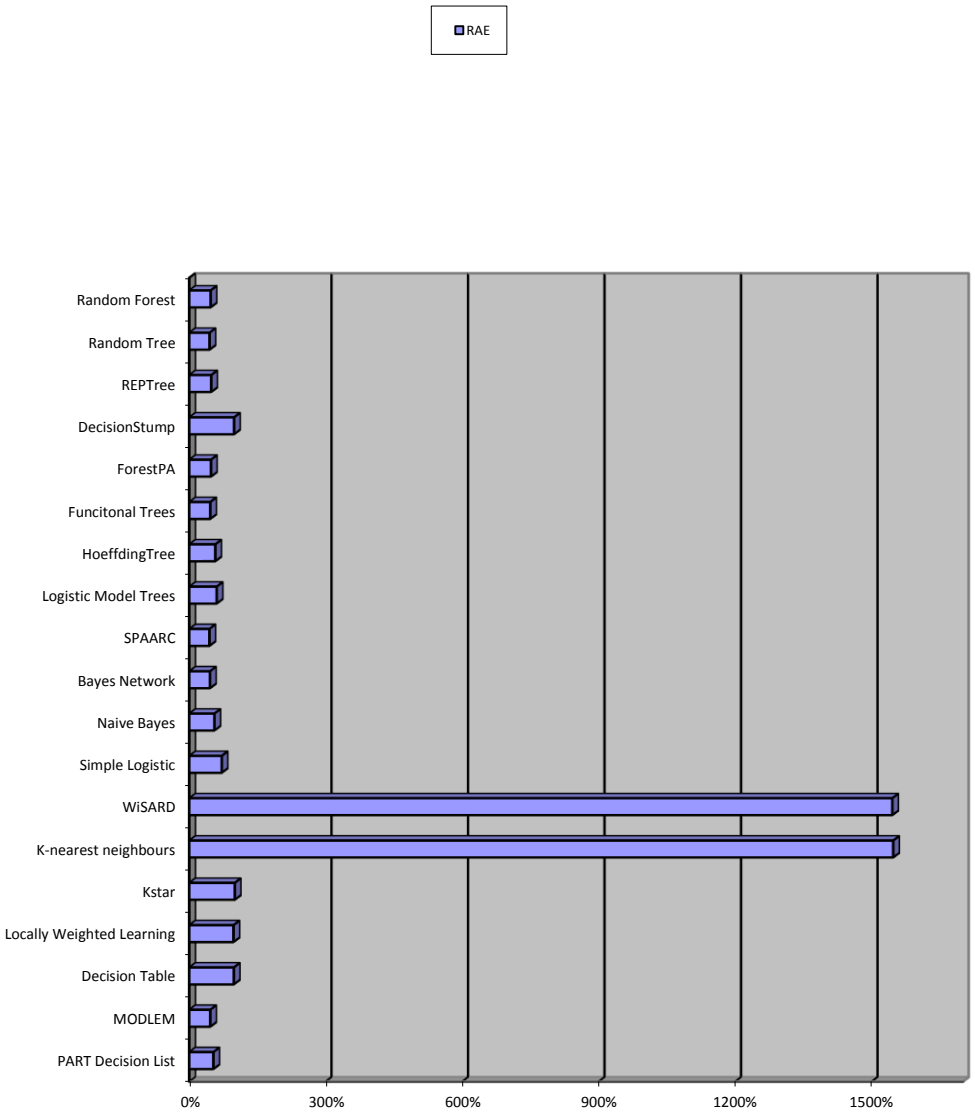
αλγόριθμος SPAARC με τιμή 0.0138, ενώ ο Random Forest δεν απέχει πολύ αφού έχει MAE ίσο με 0.0147.



Γράφημα 7: RMSE Supplied Test Set with workload

Βλέπουμε ότι όπως και προηγουμένως, οι δύο χειρότεροι αλγόριθμοι σε ό,τι αφορά το RMSE είναι και πάλι ο K-nearest neighbours και ο WiSARD με τιμές 0.5623 και 0.5617 αντίστοιχα. Εκτός από τον Random Forest και τον ForestPA όλοι

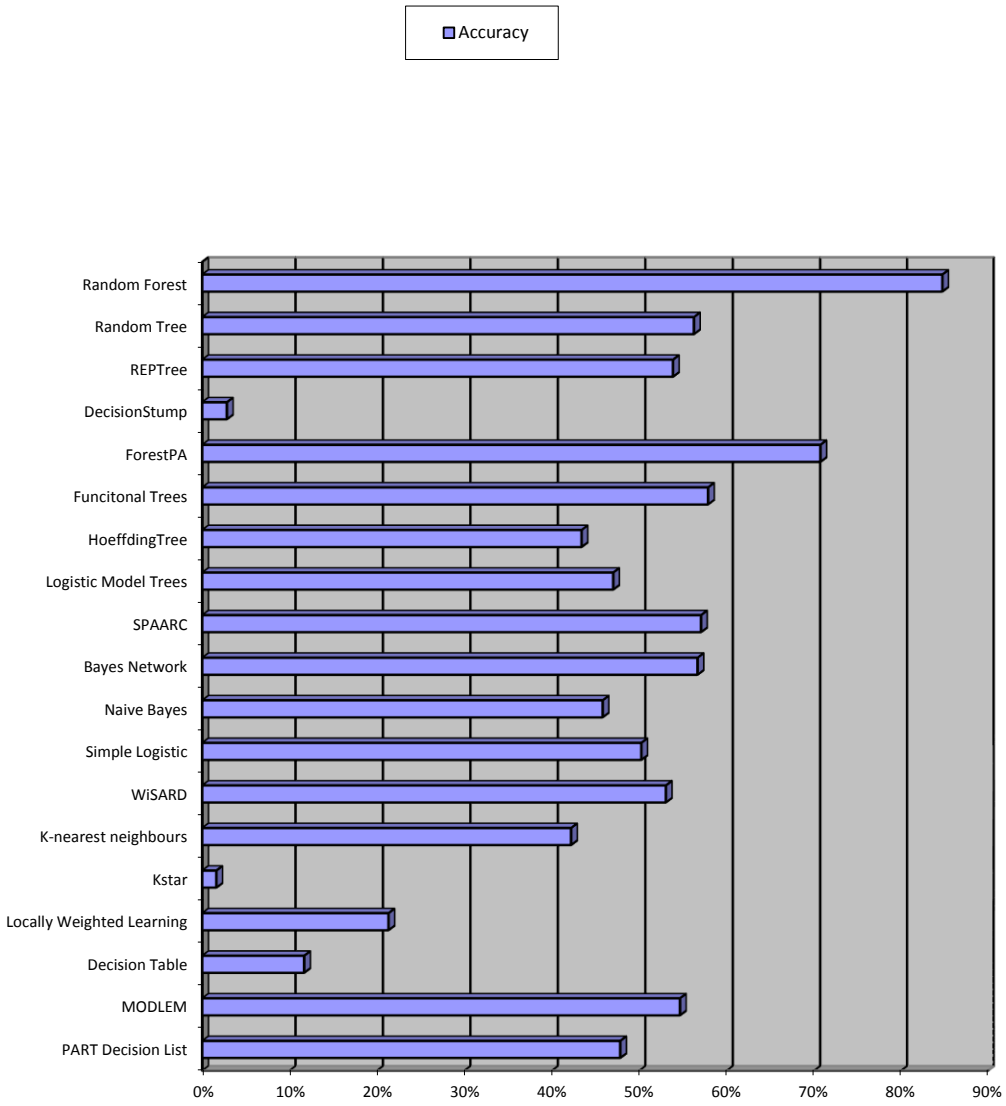
οι υπόλοιποι αλγόριθμοι παρουσιάζουν RMSE μεταξύ 0.11 και 0.13. Ο Random Forest πετυχαίνει το καλύτερο αποτέλεσμα με RMSE ίσο με 0.078



Γράφημα 8: RAE Supplied Test Set with workload

Ολοκληρώνοντας τις απεικονίσεις των σφαλμάτων, βλέπουμε πως για το RAE και πάλι οι δύο χειρότεροι αλγόριθμοι με διαφορά είναι ο WiSARD και ο K-nearest neighbours με ποσοστά 1544.98% και 1547.02%. Στη συνέχεια για τους υπόλοιπους αλγόριθμους τα ποσοστά ξεκινούν από το 100.2% του K-star και

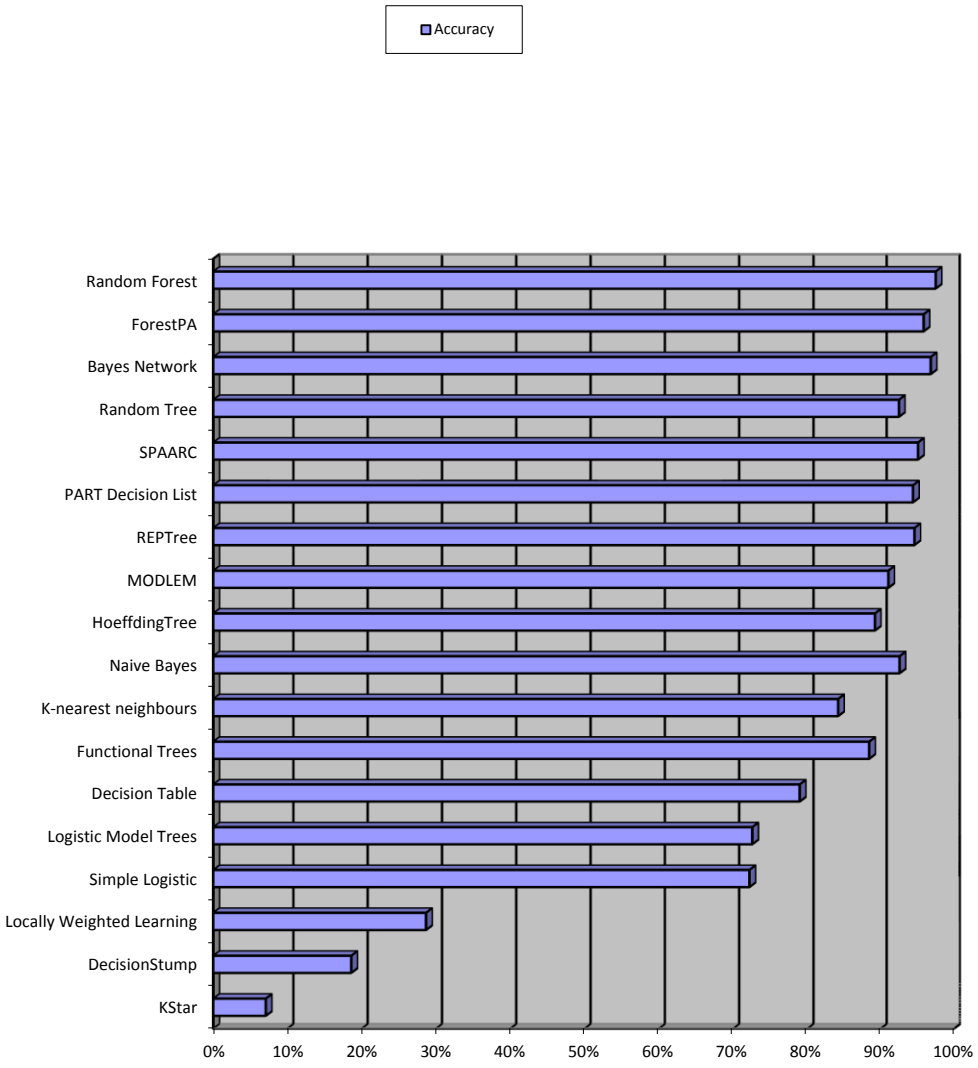
πέφτουν σταδιακά μέχρι το 44.18% του SPAARC. Ο Random Forest βρίσκεται κοντά στην καλύτερη επίδοση με ποσοστό RAE 46.93%.



Γράφημα 9: Accuracies Supplied Test Set with workload

Όπως και στην περίπτωση του Cross Validation στα ποσοστά ορθών προβλέψεων την καλύτερη επίδοση έχει ο Random Forest με ποσοστό προβλέψεων 84.74%. Όλοι οι υπόλοιποι αλγόριθμοι παρουσιάζουν μη συγκρίσιμα αποτελέσματα

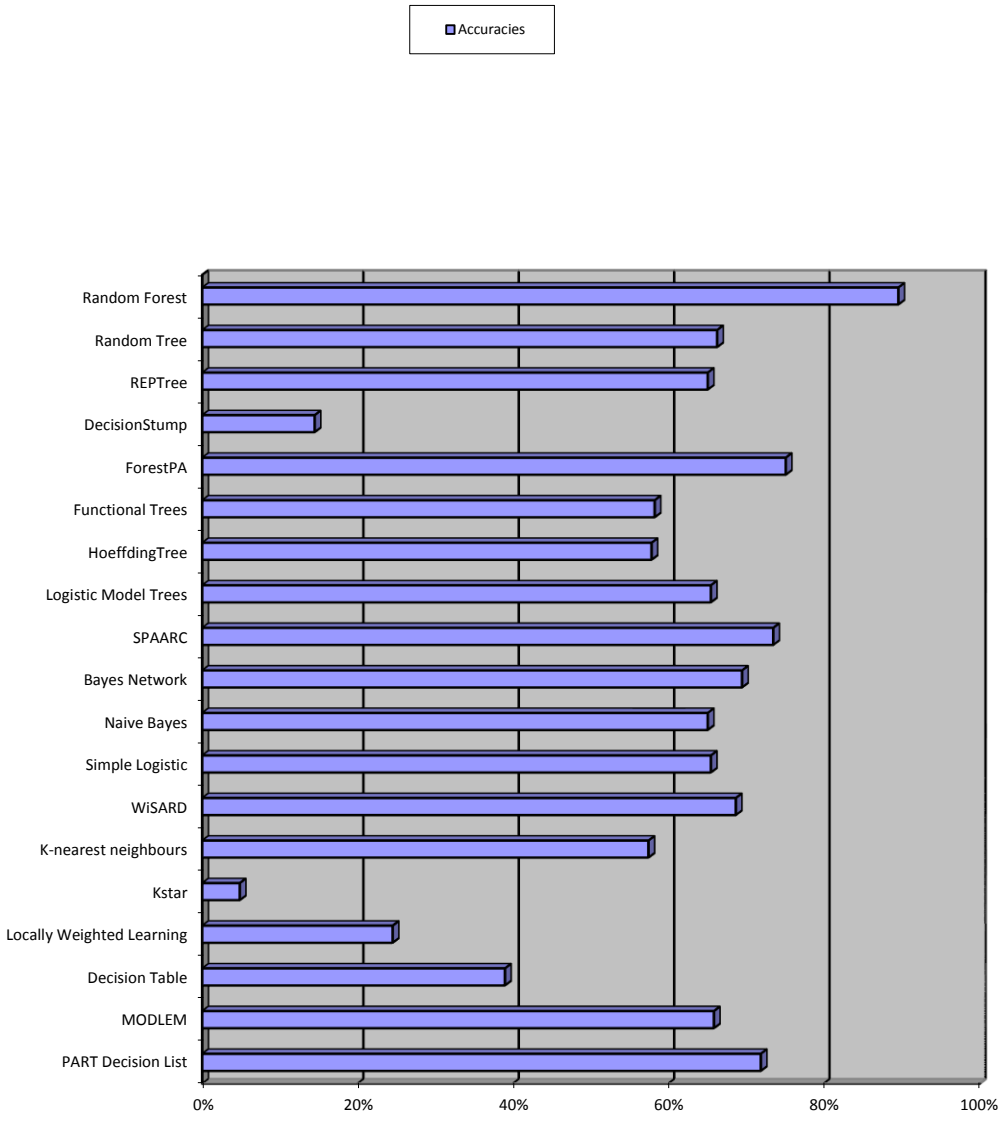
αφού ο δεύτερος καλύτερος αλγόριθμος είναι ο ForestPA με ποσοστό ακρίβειας 70.68%. Τα υπόλοιπα αποτελέσματα γενικά κυμαίνονται γύρω από το 50%.



Γράφημα 10: Accuracies Cross Validation without workload

Το μοντέλο προβλέψεων που προκύπτει από τα ομογενοποιημένα δεδομένα όπου δεν λαμβάνονται υπόψιν οι διαφορές στα workloads παρουσιάζει με Cross Validation ποσοστά προβλέψεων βελτιωμένα για όλους τους αλγόριθμους. Ο Random Forest υπερτερεί όλων με ποσοστό ορθών προβλέψεων 97.49%.

Παρατηρούμε ότι πολλοί από τους υπόλοιπους αλγόριθμους έχουν κοντινά ποσοστά και πέρα από τους 3 τελευταίους όλοι έχουν ποσοστά προβλέψεων πάνω από 70% και πη πλειοψηφία αυτών πάνω από 90%.



Γράφημα 11: Accuracies Supplied Test Set without workload

Όπως και στην περίπτωση του Cross Validation, έτσι και στον έλεγχο με εξωτερικά δεδομένα τα αποτελέσματα των προβλέψεων είναι βελτιωμένα για όλους τους αλγόριθμους. Παρόμοια με όλες τις προηγούμενες περιπτώσεις τα καλύτερα ποσοστά προβλέψεων έχει ο Random Forest.

4.5 Συμπεράσματα αξιολόγησης

Βλέπουμε ότι όπως επιζητούσαμε, ο Random Forest παρουσιάζει τα καλύτερα αποτελέσματα. Τα ποσοστά ακρίβειας με 10-fold cross validation φαίνονται ιδιαίτερα υψηλά (91.88%), πράγμα που σημαίνει ότι τελικά οι μετρικές που επιλέχθηκαν να απαρτίζουν το διάνυσμα των προφίλ ικανοποιούν το κριτήριο της διαφοροποίησης (4.2). Αν δεν την ικανοποιούσαν ο αλγόριθμος δεν θα μπορούσε να διακρίνει τα προφίλ μεταξύ τους και θα βλέπουμε χαμηλό ποσοστό προβλέψεων ή/και πολύ ψηλά επίπεδα σφαλμάτων, πράγμα που δε συμβαίνει. Αυτό που μένει λοιπόν να εξετάσουμε είναι αν ικανοποιείται και το κριτήριο της ανεξαρτησίας (4.2). Μελετώντας τα αποτελέσματα με το testing για εξωτερικά δεδομένα βλέπουμε προφανώς μία μείωση στην ακρίβεια, ωστόσο και πάλι παραμένει ικανοποιητικά υψηλή (84.74%). Τα δεδομένα όμως του testing dataset όπως έχουμε ήδη παρουσιάσει προέρχονται από εκτελέσεις σε διαφορετικό υλικό από αυτό που προέρχονται τα δεδομένα από τα οποία δημιουργήθηκε το μοντέλο των προβλέψεων. Συμπεραίνουμε λοιπόν ότι ικανοποιείται και το τελευταίο κριτήριο της ανεξαρτησίας (4.2). Η επόμενη σημαντική παρατήρηση προκύπτει από τη σύγκριση με τα αποτελέσματά των ομογενοποιημένων δεδομένων. Βλέπουμε ότι έχουμε βελτίωση σε όλες τις περιπτώσεις με ένα εντυπωσιακό ποσοστό ακρίβειας (97.19% με cross validation και 89.56% με external training set). Φανερώνεται έτσι ότι τα προφίλ που υπολογίζουμε όντως συνδέονται άμεσα με τον τύπο της εφαρμογής από την οποία προήλθαν. Συνοψίζοντας λοιπόν, υπάρχουν ισχυρές ενδείξεις ότι το profiling μιας εφαρμογής μπορεί να γίνει σε οποιαδήποτε υλική υποδομή και να εξαχθούν συμπεράσματα τα οποία ο διαχειριστής θα μπορέσει να αξιοποιήσει προβλέποντας τις απαιτήσεις της εφαρμογής. Όσο πιο λεπτομερής διαχωρισμός των εφαρμογών που απαρτίζουν το μοντέλο υπάρχει, τόσο αυξάνεται και η δυσκολία διάκρισης μεταξύ τους, όπως και είναι λογικό. Με την κατάλληλη επιλογή όμως των benchmarks η ακρίβεια προβλέψεων βρίσκεται σε πολύ υψηλά επίπεδα.

5 Συμπεράσματα και μελλοντικές κατευθύνσεις

Στη διπλωματική αυτή εργασία δημιουργήσαμε ένα εργαλείο profiling containerized εφαρμογών με docker, με στόχο τη διευκόλυνση της κατάλληλης επιλογής του περιβάλλοντος εκτέλεσης αυτών. Ορίσαμε ένα απλό REST API με βάση τα node-RED flows μέσω του οποίου αυτοματοποιήσαμε την καταγραφή μετρικών και παραγωγή των profiles. Εξετάσαμε το κατά πόσο μπορούν offline προφίλ να προβλέψουν τις ανάγκες σε πόρους μιας εφαρμογής ώστε μέσω της αντιστοίχισης των προφίλ αυτών με γνωστά προφίλ, ένας διαχειριστής να είναι σε θέση να βελτιστοποιήσει την επιλογή του κατάλληλου instance από τα διαθέσιμα του provider. Συλλέξαμε δεδομένα με χρήση της διαδικασίας που ορίσαμε δημιουργώντας ένα σύνολο δεδομένων με 1300 εγγραφές. Μέσω πειραματισμών σε ετερογενή hardware, με πληθώρα από benchmarks καταφέραμε να δημιουργήσουμε ένα μοντέλο πρόβλεψης με ποσοστά ακρίβειας έως και 84.74%. Ομογενοποιώντας τα προφίλ από τα χρησιμοποιούμενα benchmarks με βάση τον τύπο της εφαρμογής φτάσαμε στο 89.56%. Τα ετερογενή hardware περιλαμβάνουν τόσο πραγματικά όσο και εικονικά συστήματα. Επιλέξαμε συγκεκριμένες μετρικές για τη δημιουργία του διανύσματος των προφίλ και τον αλγόριθμο Random Forest, έπειτα από θεωρητική αλλά και πρακτική ανάλυση των ιδιοτήτων τους. Προβήκαμε σε μία στατιστική ανάλυση των διακυμάνσεων των τιμών των συλλεγμένων μετρικών μεταξύ των διάφορων εκτελέσεων ώστε να δικαιολογήσουμε τον ορισμό του σχήματος των profiles. Συγκρίναμε τελικά τα σφάλματα και τα αποτελέσματα των προβλέψεων για μοντέλα που προέκυψαν από πολλούς διαφορετικούς αλγόριθμους. Συνδυάζοντας όλα τα παραπάνω προσπαθήσαμε να κάνουμε τη διαδικασία των προβλέψεων να εξαρτάται όσο λιγότερο γίνεται από το υλικό εκτέλεσης των εφαρμογών. Με βάση όλα τα προηγούμενα φαίνεται ότι υπάρχει μεγάλη προοπτική στην βελτιστοποίηση επιλογής των κατάλληλων VMs στο cloud με τεχνικές που δεν επιβαρύνουν το κόστος αφού μπορούν να γενικευτούν σε offline περιβάλλοντα. Περαιτέρω εξέταση και απόδειξη αυτών των ισχυρισμών θα μπορούσε να γίνει με πειραματισμούς σε ακόμα περισσότερα διαφορετικά μηχανήματα και με τελείως διαφορετικά benchmarks. Επίσης θα μπορούσε να εξεταστεί το ενδεχόμενο να δημιουργηθούν ξεχωριστά μοντέλα για διαφορετικές φάσεις εκτέλεσης των εφαρμογών, δημιουργώντας τεχνητό φόρτο εργασίας και

λαμβάνοντας υπόψιν τα προφίλ που προκύπτουν μόνο για ένα συγκεκριμένο χρονικό διάστημα όπου διατηρούμε τον φόρτο αυτόν στα επίπεδα που θέλουμε.

Αναφορές

[1] Rani, P. CLOUD COMPUTING: A NEW REVOLUTION IN INFORMATION TECHNOLOGY SECTOR.

[2] Attaran, M. (2017). Cloud computing technology: leveraging the power of the internet to improve business performance. *Journal of International Technology and Information Management*, *26*(1), 112-137.

[3] Haji, L. M., Ahmad, O. M., Zeebaree, S. R., Dino, H. I., Zebari, R. R., & Shukur, H. M. (2020). Impact of cloud computing and internet of things on the future internet. *Technology Reports of Kansai University*, *62*(5), 2179-2190.

[4] Li, X., Pan, L., & Liu, S. (2022). A survey of resource provisioning problem in cloud brokers. *Journal of Network and Computer Applications*, 103384.

[5] Wu, D., & Gokhale, A. (2013, December). A self-tuning system based on application profiling and performance analysis for optimizing hadoop mapreduce cluster configuration. In *20th Annual International Conference on High Performance Computing* (pp. 89-98). IEEE.

[6] Martin, A., & Marangozova-Martin, V. (2018). Automatic benchmark profiling through advanced workflow-based trace analysis. *Software: Practice and Experience*, *48*(6), 1195-1217.

[7] Kumar, J., Singh, A. K., & Buyya, R. (2021). Self directed learning based workload forecasting model for cloud resource management. *Information Sciences*, *543*, 345-366.

[8] Baughman, M., Chard, R., Ward, L., Pitt, J., Chard, K., & Foster, I. (2018, December). Profiling and predicting application performance on the cloud. In *11th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*.

[9] Chard, R., Chard, K., Ng, B., Bubendorfer, K., Rodriguez, A., Madduri, R., & Foster, I. (2016, May). An automated tool profiling service for the cloud. In

2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid) (pp. 223-232). IEEE.

[10] Varghese, B., Subba, L. T., Thai, L., & Barker, A. (2016, May). DocLite: A docker-based lightweight cloud benchmarking tool. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (pp. 213-222). IEEE.

[11] Scheuner, J., & Leitner, P. (2018, July). Estimating cloud application performance based on micro-benchmark profiling. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)* (pp. 90-97). IEEE.

[12] Kaviani, N., Wohlstadter, E., & Lea, R. (2011, December). Profiling-as-a-service: Adaptive scalable resource profiling for the cloud in the cloud. In *International Conference on Service-Oriented Computing* (pp. 157-171). Springer, Berlin, Heidelberg.

[13] Catillo, M., Ocone, L., Rak, M., & Villano, U. (2020, September). Auto-scaling Applications in the Cloud by Simple Indexes with Complex Loads. In *2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)* (pp. 76-81). IEEE.

[14] Do, A. V., Chen, J., Wang, C., Lee, Y. C., Zomaya, A. Y., & Zhou, B. B. (2011, July). Profiling applications for virtual machine placement in clouds. In *2011 IEEE 4th international conference on cloud computing* (pp. 660-667). IEEE.

[15] Kandalintsev, A., Cigno, R. L., Kliazovich, D., & Bouvry, P. (2014, February). Profiling cloud applications with hardware performance counters. In *The International Conference on Information Networking 2014 (ICOIN2014)* (pp. 52-57). IEEE.

[16] Mariani, G., Anghel, A., Jongerius, R., & Dittmann, G. (2018). Predicting cloud performance for HPC applications before deployment. *Future Generation Computer Systems*, *87*, 618-628.