Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
ΔΠΜΣ Επιστημη Δεδομενων και Μηχανικη Μαθηση

Σύστημα Υποστήριξης Αποφάσεων για Παραμετροποίηση Ροών Εργασίας Apache Spark σε Συστοιχίες Kubernetes

Διπλωματικη Εργασια

του

ΣΠΥΡΟΥ ΑΡΗ

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Δεκέμβριος 2022

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
ΔΠΜΣ Επιστημη Δεδομενων και Μηχανικη Μαθηση

Σύστημα Υποστήριξης Αποφάσεων για Παραμετροποίηση Ροών Εργασίας Apache Spark σε Συστοιχίες Kubernetes

Διπλωματικη Εργασια

του

ΣΠΥΡΟΥ ΑΡΗ

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την Δεκέμβριος 2022.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Νεκτάριος Κοζύρης	Ιωάννης Κωνσταντίνου	Δημήτριος Τσουμάκος
Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Π.Θ.	Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Δεκέμβριος 2022

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
ΔΠΜΣ Επιστημη Δεδομενων και Μηχανικη Μαθηση

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

(Υπογραφή)

.....
Άρης Σπύρου

Δεκέμβριος 2022

Περίληψη

Τα τελευταία χρόνια βλέπουμε μία αυξητική τάση όσον αφορά τον όγκο τον δεδομένων που παράγονται. Τα δεδομένα αποτελούν βασικό προϊόν είτε υποπροϊόν όλων των ανθρωπίνων δραστηριοτήτων καθώς σε διάφορα στάδια εμπλέκεται συχνά ένας εγκέφαλος που είναι ικανός να παράγει και να εξάγει δεδομένα. Ως αποτέλεσμα έχουν δημιουργηθεί πολλά εργαλεία που αποσκοπούν στην ανάλυση και στην εφαρμογή μεθόδων μηχανικής μάθησης σε μεγάλους όγκους δεδομένων. Ώριμα συστήματα όπως το Hadoop και το Apache Spark αποτελούν ακρογωνιαίους λίθους για την διαχείριση μεγάλων δεδομένων. Πλέον με την στροφή των υπολογιστικών μοτίβων στην χρήση μηχανών ελαφριάς εικονικοποίησης εισάγονται στο προσκήνιο και άλλα συστήματα όπως το Kubeflow που ζει σε συστοιχίες Kubernetes. Δεν υπάρχουν ωστόσο ολοκληρωμένες λύσεις που να επιτρέπουν την πλήρη διαλειτουργικότητα των συστοιχιών Apache Spark με το οικοσύστημα του Kubeflow. Στην παρούσα διπλωματική εργασία επεκτείνουμε το εργαλείο Kubeflow Pipelines με σκοπό την υποστήριξη επιτόπιων συστοιχιών Apache Spark, αναπτύσσουμε δοκιμαστική διοχέτευση που επιδεικνύει της δυνατότητες του συστήματος που αναπτύξαμε, αποτιμούμε την συστοιχία Apache Spark σε γνωστό benchmark της βιομηχανίας και προτείνουμε μια μέθοδο για την υποστήριξη της απόφασης στην εκκίνηση συστοιχιών.

Λέξεις Κλειδιά

Δεδομένα, Εργαλεία ανάλυσης, Μηχανική μάθηση, Hadoop, Apache Spark, ελαφριά εικονικοποίηση, μηχανές ελαφριάς εικονικοποίησης, Kubeflow, Kubernetes, Kubeflow Pipelines, TPC-DS

Abstract

In recent years we have seen an increasing trend in the volume of data being produced. Data is a core product or by-product of all human activities due to the involvement of computer processors capable of extracting and producing data at various stages of different processes. As a result, many tools have been developed that are meant to analyze and apply machine learning methods to large volumes of data. Mature systems like Hadoop and Apache Spark are cornerstones of big data management. Nowadays the popular computing paradigm has shifted to the use of lightweight virtualization engines, other systems such as Kubeflow that live within the Kubernetes ecosystem are more widespread. However, there are no complete solutions that allow full interoperability of Apache Spark clusters with the Kubeflow ecosystem. In this thesis we extend the Kubeflow Pipelines tool to support on-premises Apache Spark clusters, we develop a test pipeline that demonstrates the capabilities of the system we developed, we evaluate the Apache Spark cluster on an industry standard benchmark and propose a method to support users in the cluster-setup decision process.

Keywords

Data, Analysis tools, Machine Learning, Hadoop, Apache Spark, lightweight virtualization, lightweight virtualization engines, Kubeflow, Kubernetes, Kubeflow Pipelines, TPC-DS

στους γονείς μου

Ευχαριστίες

Καταρχάς θα ήθελα να ευχαριστήσω τον κ. Ιωάννη Κωνσταντίνου για τις κατευθυντήριες που μου έδωσε κατά την διάρκεια της εκπόνησης της διπλωματικής εργασίας αλλά και για τις άνευ ωραρίου συμβουλές που μου προσέφερε που κατέστησαν την δουλειά μου ευκολότερη. Στη συνέχεια θα ήθελα να εκφράσω την ευγνωμοσύνη μου στον κ. Νεκτάριο Κοζύρη και το Εργαστήριο Υπολογιστικών Συστημάτων για την ευκαιρία που μου δόθηκε να ασχοληθώ με ένα ενδιαφέρον θέμα σε ένα υγιές περιβάλλον. Θα ήθελα επίσης να ευχαριστήσω την οικογένειά μου για την απαράμιλλη υποστήριξη τους καθ' όλη την διάρκεια της ακαδημαϊκής μου πορείας καθώς χωρίς αυτούς δεν θα μπορούσα να είμαι τώρα εδώ. Τέλος, θα ήθελα να ευχαριστήσω την Κασσιανή για την υποστήριξη της και την πίστη που επέδειξε σε εμένα, κατά την διάρκεια των σπουδών μου αλλά και της καθημερινότητας.

Αθήνα, Δεκέμβριος 2022

Άρης Σπύρου

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

Λόγω του μεγάλου όγκου δεδομένων που παράγονται μέσω διαφόρων ψηφιακών δραστηριοτήτων όπως για παράδειγμα από εφαρμογές στο διαδίκτυο, από αυτοκίνητα με σύγχρονους ψηφιακούς εγκεφάλους, και οικιακές συσκευές (έξυπνο ψυγείο, κλιματιστικά κλπ.), παρουσιάζεται συχνά η ανάγκη διαφόρων ενδιαφερόμενων όπως επιχειρήσεων και ακαδημαϊκών να γίνει κάποιου είδους επεξεργασία αυτών των πληροφοριών με απώτερο σκοπό την εκμαίευση χρήσιμων γνώσεων. Χρησιμοποιώντας κανείς εργαλεία ανάλυσης δεδομένων αλλά και τεχνικές/αλγορίθμους μηχανικής μάθησης μπορεί να φθάσει σε χρήσιμα συμπεράσματα που έχουν άλλοτε εμπορική ή ερευνητική αξία. Παράλληλα παρατηρείται τα τελευταία χρόνια ραγδαία ανάπτυξη στις υπηρεσίες υπολογιστικού νέφους τόσο για την χρήση υπολογιστικών πόρων όσο και για την αποθήκευση δεδομένων. Με την ανάπτυξη στην τεχνολογία του υλικού (hardware) βλέπουμε τόσο πτώση στις τιμές του όσο και αύξηση στις δυνατότητες αυτού. Το γεγονός αυτό έχει καλλιεργήσει πρόσφορο έδαφος για παρόχους όπως η Amazon (AWS), η Google (GCP) και η Microsoft (Azure) να διατηρούν η καθεμία δική της πλατφόρμα παροχής υπηρεσιών υπολογιστικού νέφους.

Μεταξύ άλλων υπηρεσιών αυτές οι πλατφόρμες προσφέρουν περιβάλλοντα δημιουργίας και διαχείρισης «μηχανών» ελαφριάς εικονικοποίησης (lightweight container virtualization). Μέσω αυτών των μηχανών επιτυγχάνεται μείωση στους χρόνους υλοποίησης ενός έργου καθώς πολλές από τις εφαρμογές που χρησιμοποιούνται στον χώρο της αναλυτικής επεξεργασίας υπάρχουν σε μορφή ικανή να τρέξει σε τέτοια υποδομή.

Ένα από αυτά τα συστήματα διαχείρισης εικονικών μηχανών είναι το Google Kubeflow [1] το οποίο λειτουργεί αποκλειστικά πάνω σε μία μηχανή ενορχήστρωσης μηχανών ελαφριάς εικονικοποίησης που λέγεται Kubernetes ή κοινώς K8s [2]. Το Kubeflow είναι ένα σύστημα που υπόσχεται να απλοποιήσει την υλοποίηση ροών μηχανικής μάθησης σε δομές Kubernetes. Μεταξύ άλλων εργαλείων περιέχει και το Kubeflow Pipelines [3] μέσω του οποίου οι προγραμματιστές μπορούν να δημιουργήσουν ροές επεξεργασίας δεδομένων με την μορφή κατευθυνόμενων ακυκλικών γράφων (directed acyclic graphs – DAG) με κάθε κόμβο να αναπαριστά ένα στάδιο επεξεργασίας.

Σε κατανεμημένα περιβάλλοντα επεξεργασίας ήτοι για μεγάλους όγκους δεδομένων ένα από τα επικρατέστερα εργαλεία είναι το Apache Spark [4]. Το Spark είναι μία μηχανή ικανή να κατανέμει δουλειές επεξεργασίας δεδομένων (data engineering) και έχει περαιτέρω εφαρμογές σε επιστήμη δεδομένων και μηχανική μάθηση. Υποστηρίζει πολλές γλώσσες προγραμματισμού (Python, Java, Scala, R, SQL) και τρέχει σε τοπικές συστοιχίες ή σε

έναν υπολογιστή κάνοντας χρήση πολλαπλών πυρήνων. Υπάρχει πλέον ένα project [5] που προσπαθεί να ενσωματώσει το Spark στο οικοσύστημα του Kubernetes για να γίνει δυνατή η εκμετάλλευση των δυνατοτήτων ενορχήστρωσης μίας συστοιχίας υπολογιστών από την μηχανή του Spark επιτυγχάνοντας γρηγορότερη επεξεργασία και ευελιξία.

1.1 Αντικείμενο της διπλωματικής

Μελετώντας τον οδηγό εκκίνησης του Kubeflow είναι ξεκάθαρο από την αρχή ότι αυτό το εργαλείο προορίζεται για χρήστες που είναι ήδη πελάτες της Google και συνεπώς ενταγμένοι στο οικοσύστημα του GCP. Υπάρχουν για παράδειγμα έτοιμα configurations για να υλοποιήσει κανείς διοχετεύσεις μέσω του Kubeflow Pipelines και να κάνει χρήση απομακρυσμένων πόρων που βρίσκονται στο GCP χωρίς ωστόσο να αποκλείονται και κατάλληλες ρυθμίσεις για άλλα περιβάλλοντα υπολογιστικού νέφους όπως αυτό του AWS ή του Azure.

Σκοπός λοιπόν της παρούσας διπλωματικής εργασίας είναι η επέκταση του εργαλείου Kubeflow Pipelines με σκοπό την ομαλή υποστήριξη επιτόπιων (on-premises) υπολογιστικών πόρων καθώς και την εκκίνηση φόρτων εργασίας Spark. Για τον σκοπό αυτό έχει αναπτυχθεί ένα component ικανό να εκκινήσει μια συστοιχία Spark και να μεταφέρει το αποτέλεσμα της επεξεργασίας σε μετέπειτα στάδιο της διοχέτευσης το οποίο μπορεί να αποτελείται είτε από κάποια διεργασία αναλυτικής επεξεργασίας είτε από κάποιον αλγόριθμο μηχανικής μάθησης. Επιπλέον στόχος είναι να αποτιμηθεί η απόδοση του συστήματος όσον αφορά φόρτους εργασίας Spark και να αναπτυχθεί ανεξάρτητα ένα σύστημα υποστήριξης αποφάσεων που θα μπορεί να καθοδηγεί τον χρήστη στην λήψη αποφάσεων όταν πρόκειται να επιλέξει πόρους για την ροή φόρτων εργασίας.

1.2 Δομή της εργασίας

Η δομή της παρούσας διπλωματικής εργασίας έχει ως εξής:

- Στο κεφάλαιο 2 γίνεται μία επισκόπηση στις δομικές τεχνολογίες που αποτελούν θεμέλιο για την διπλωματική.
- Στο κεφάλαιο 3 περιγράφεται η δομή των συστημάτων που απαρτίζουν την διπλωματική εργασία.
- Στο κεφάλαιο 4 αναλύεται η διαδικασία υλοποίησης των διαφόρων συστημάτων.
- Στο κεφάλαιο 5 περιγράφεται η δημιουργία της διοχέτευσης Kubeflow και του μοντέλου που έτρεξε εντός αυτής.
- Στο κεφάλαιο 6 γίνεται αναφορά στη μεθοδολογία αποτίμησης της συστοιχίας Apache Spark.
- Στο κεφάλαιο 7 περιγράφεται το σύστημα υποστήριξης αποφάσεων που αναπτύχθηκε στα πλαίσια της διπλωματικής εργασίας.
- Στο κεφάλαιο 8 συνοψίζεται η συνεισφορά, οι περιορισμοί της εργασίας καθώς και ιδέες για μελλοντικές επεκτάσεις.

Κεφάλαιο 2

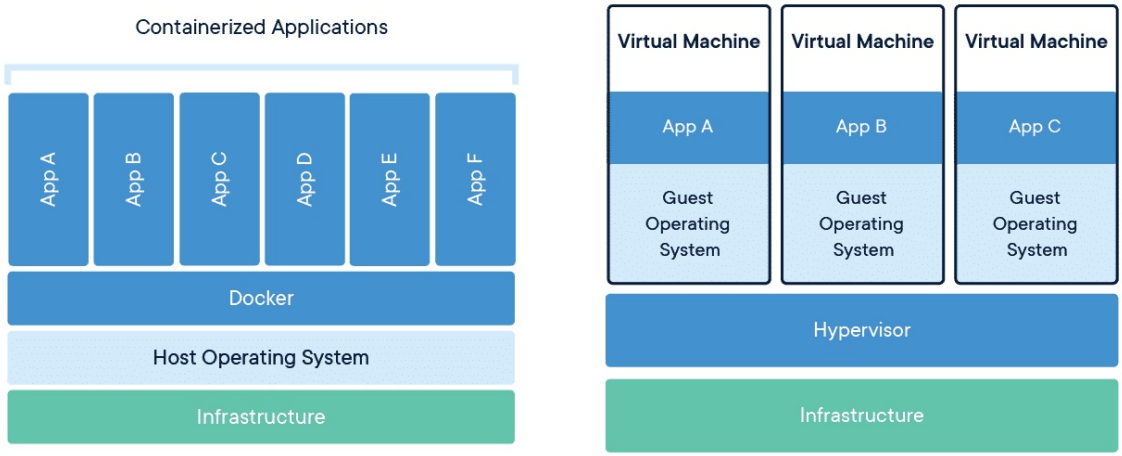
Επισκόπηση Τεχνολογιών

Σε αυτό το κεφάλαιο περιγράφονται επαρκώς αλλά όχι εκτενώς διάφορες έννοιες, τεχνολογίες και εργαλεία που θεωρούνται απαραίτητα για την διεξαγωγή της παρούσας διπλωματικής και την κατανόησή της από τον αναγνώστη. Για λόγους καλύτερης κατανόησης και λογικής συνοχής το κεφάλαιο θα αρχίσει με πιο θεμελιώδης όρους και θα προχωρήσει σε πιο ευρείες έννοιες.

2.1 Containers

Τα containers είναι μία μορφή εικονικοποίησης που χαρακτηρίζεται από το πόσο ελαφριά είναι καθώς και την ευελιξία της. Η βασική διαφορά τους από τις παραδοσιακές εικονικές μηχανές είναι ότι δεν χρησιμοποιούν hypervisor, με αποτέλεσμα να εμφανίζουν γρηγορότερη ανάθεση υπολογιστικών πόρων και μικρότερο χρόνο εκκίνησης νέων εφαρμογών που στήνονται μέσα σε containers.

Αντιθέτως με τις κλασικές εικονικές μηχανές (VMs) που ήταν αναγκαία η εκκίνηση ενός λειτουργικού από την αρχή αλλά και η εκ των προτέρων ανάθεση πόρων στο συγκεκριμένο εικονικό μηχάνημα, τα containers μπορούν να εκκινηθούν γρήγορα χωρίς να χρειάζεται να δεσμεύουν αχρείαστους πόρους από την αρχή της λειτουργίας τους.



Εικόνα 2.1: *Διαφορές μεταξύ containers και εικονικών μηχανών*

Τα containers αποτελούνται από όλα τα απαραίτητα κομμάτια για να τρέξει μια εφαρμογή ή ένα πρόγραμμα. Σε ένα container συμπεριλαμβάνονται ο κώδικας, οι βιβλιοθήκες που

χρειάζονται για να τρέξει ακόμα και το λειτουργικό σύστημα για το οποίο είναι σχεδιασμένος ο κώδικας. Αυτό σημαίνει ότι το container είναι ανεξάρτητο από το υπόλοιπο υπολογιστικό περιβάλλον που μπορεί να τρέχει στο παρασκήνιο. Αυτό σημαίνει ότι η εφαρμογή ήτοι το container μπορεί να τρέξει παντού από ένα απλό επιτραπέζιο υπολογιστή, σε μία συστοιχία ή σε ένα οποιοδήποτε υπολογιστικό νέφος. Από εκεί πηγάζει η ευελιξία που έχει κανείς όταν αναπτύσσει εφαρμογές σε container.

Τα containers είναι μια καλή εναλλακτική από το να αναπτύσσει κανείς προγράμματα για μία πλατφόρμα ή ένα λειτουργικό σύστημα, το οποίο είναι γνωστό ότι προκαλεί προβλήματα όταν θέλει κανείς να μεταφέρει αυτή τη δουλειά σε άλλα περιβάλλοντα που ενδεχομένως να έχουν διαφορετικά συστατικά (hardware/software). Το οποίο συνήθως σημαίνει ότι εμφανίζονται διάφορα bug και σφάλματα που κατ' επέκταση χρειάζονται χρόνο εκ μέρους του προγραμματιστή για διόρθωση.

2.1.1 Πλεονεκτήματα

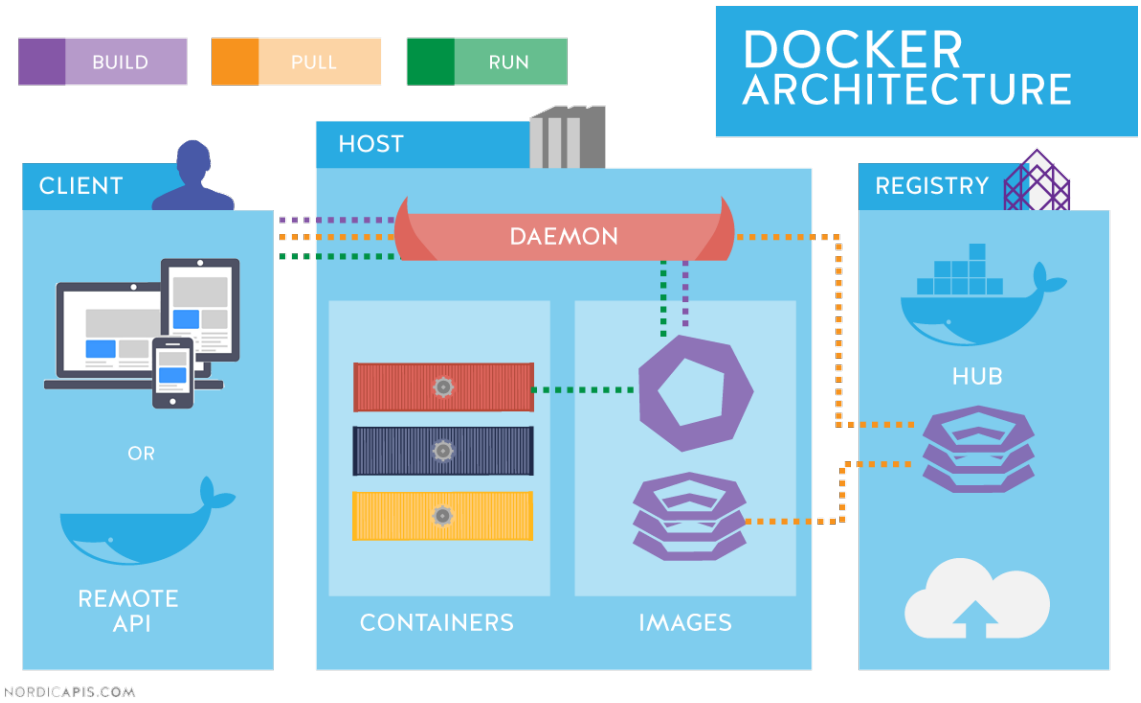
Τα containers [6] προσφέρουν πολλά πλεονεκτήματα στους προγραμματιστές. Μεταξύ άλλων είναι και τα ακόλουθα:

- Φορητότητα: Κάθε container δημιουργεί ένα εκτελέσιμο πακέτο λογισμικού που είναι ανεξάρτητο από το λειτουργικό σύστημα του μηχανήματος στο οποίο τρέχει και συνεπώς είναι φορητό και μπορεί να τρέχει ακριβώς το ίδιο σε οποιαδήποτε πλατφόρμα ή υπολογιστικό νέφος.

- Ευελιξία: Η ανοιχτού κώδικα μηχανή Docker για την λειτουργία container ξεκίνησε το σημερινά επικρατές πρότυπο που περιέχει απλά εργαλεία ανάπτυξης λογισμικού και μία καθολική προσέγγιση συσκευασίας προγραμμάτων που δουλεύει και σε Linux καθώς και σε Windows. Το επικρατές οικοσύστημα των container έχει πλέον μεταφερθεί σε μηχανές που διαχειρίζεται το Open Container Initiative (OCI). Ωστόσο αυτό δεν επηρεάζει τους προγραμματιστές καθώς υπάρχει πλήρης συμβατότητα.

- Ταχύτητα: Τα containers συχνά χαρακτηρίζονται ως «ελαφριά» το οποίο στην πράξη σημαίνει ότι μοιράζονται τον πυρήνα του λειτουργικού συστήματος του μηχανήματος στο οποίο τρέχουν και συνεπώς δεν επιβαρύνονται σε σχέση με συμβατικά VM. Έτσι επιτυγχάνεται περισσότερη αποτελεσματικότητα στους διακοσμητές που τρέχουν containers και μειώνει το κόστος που προέρχεται από αδειοδότηση και συντήρηση ενώ ταυτόχρονα επιταχύνει τους χρόνους εκκίνησης καθώς δεν χρειάζεται εκκίνηση κάποιου λειτουργικού συστήματος.

- Απομόνωση σφάλματος: Κάθε εφαρμογή που τρέχει σε container είναι απομονωμένη και λειτουργεί ανεξάρτητα από άλλες. Η αποτυχία ενός container δεν επηρεάζει την λειτουργία των άλλων. Οι ομάδες ανάπτυξης λογισμικού μπορούν να εντοπίσουν και να διορθώσουν τεχνικά ζητήματα που βρίσκονται σε ένα container χωρίς να χρειαστεί να κλείσουν άλλα containers το οποίο αυξάνει το ποσοστό διαθεσιμότητας μίας υπηρεσίας που παρέχεται. Επιπρόσθετα το η μηχανή που τρέχει το container μπορεί να χρησιμοποιήσει τεχνικές ασφάλειας απομόνωσης του λειτουργικού συστήματος όπως το SELinux access control.



Εικόνα 2.2: *Επισκόπηση της αρχιτεκτονικής του Docker*

- Αποδοτικότητα: Οι εφαρμογές που τρέχουν σε περιβάλλοντα container μοιράζονται τον πυρήνα του λειτουργικού και οι διεπαφές των εφαρμογών εντός ενός container μπορούν να αξιοποιούνται και από άλλα container. Συνεπώς τα container είναι εκ φύσεως πιο μικρά σε χωρητικότητα από ένα VM και χρειάζονται λιγότερο χρόνο εκκίνησης το οποίο καθιστά εφικτό να τρέχουν πολλά περισσότερα containers σε μία υπολογιστική δομή (π.χ. cloud, server κ.λπ.) από ότι VM.

- Ευκολία διαχείρισης: Μία πλατφόρμα ενορχήστρωσης container αυτοματοποιεί την διαδικασία εγκατάστασης, κλιμάκωσης και διαχείρισης φόρτων εργασίας και υπηρεσιών που τρέχουν σε containers. Τέτοιες πλατφόρμες (όπως το Kubernetes που θα δούμε στη συνέχεια) κάνουν εύκολη τις διαδικασίες διαχείρισης όπως είναι η κλιμάκωση των εφαρμογών, την κυκλοφορία νέων εκδόσεων των εφαρμογών και παρέχουν μεταξύ άλλων λειτουργίες παρακολούθησης (monitoring), καταγραφής συμβάντων (logging) και αποσφαλμάτωσης (debugging). Το Kubernetes το οποίο πιθανώς να είναι το πιο διαδεδομένο σύστημα ενορχήστρωσης container είναι μία τεχνολογία ανοικτού κώδικα (αρχικά δόθηκε στην κοινότητα ανοικτού κώδικα από την Google και βασίστηκε σε ένα ιδιόκτητο project ονόματι Borg) που αυτοματοποιεί container βασισμένα στο λειτουργικό Linux. Το Kubernetes είναι συμβατό με διάφορες μηχανές container όπως η δημοφιλής Docker αλλά δουλεύει και με οποιαδήποτε μηχανή συμφωνεί με τα πρότυπα OCI [7] (Open Container Initiative) για τους τύπους των εικόνων.

- Ασφάλεια: Η απομόνωση των εφαρμογών ως containers εμποδίζει εκ φύσεως την εισβολή κακόβουλου κώδικα από το να επηρεάσει άλλα containers ή το μηχάνημα σαν σύνολο. Επιπλέον, τα δικαιώματα ασφάλειας μπορούν να οριστούν κατά τέτοιο τρόπο έτσι ώστε να αποκλείεται αυτόματα η εισχώρηση ανεπιθύμητων στοιχείων στα

container ή να περιορισθεί η επικοινωνία με περιττούς πόρους.

2.1.2 Πληροφορίες για την υλοποίηση

Στην συγκεκριμένη διπλωματική εργασία όπου ήταν απαραίτητη η δημιουργία νέων εικόνων έγινε χρήση του γνωστού λογισμικού διαχείρισης containers ονόματι Docker [8]. Το Docker ουσιαστικά επιτρέπει στον χρήστη να δημιουργήσει μία νέα εικόνα με την δήλωση ενός αρχείου Dockerfile. Το αρχείο αυτό περιέχει όλες τις απαραίτητες πληροφορίες για την δημιουργία ενός container όπως σε ποια προϋπάρχουσα εικόνα να βασιστεί, ποιες άλλες ενέργειες πρέπει να εκπονηθούν πριν την έναρξη του container (π.χ. δήλωση μεταβλητών περιβάλλοντος, εγκατάσταση λογισμικού κ.λπ.) και ποια αρχεία χρειάζεται.

2.2 Kubernetes

Το Kubernetes γνωστό και ως k8s είναι μία πλατφόρμα ενορχήστρωσης container ανοικτού κώδικα η οποία αυτοματοποιεί πολλές διεργασίες που αφορούν στην δήλωση, διαχείριση και κλιμάκωση εφαρμογών που τρέχουν σε containers.

Ο πιο βασικός όρος στο οικοσύστημα του Kubernetes είναι η συστοιχία (cluster). Μία συστοιχία μπορεί να αποτελείται από μηχανήματα σε on-premise, δημόσια, ιδιωτικά και υβριδικά υπολογιστικά νέφη.

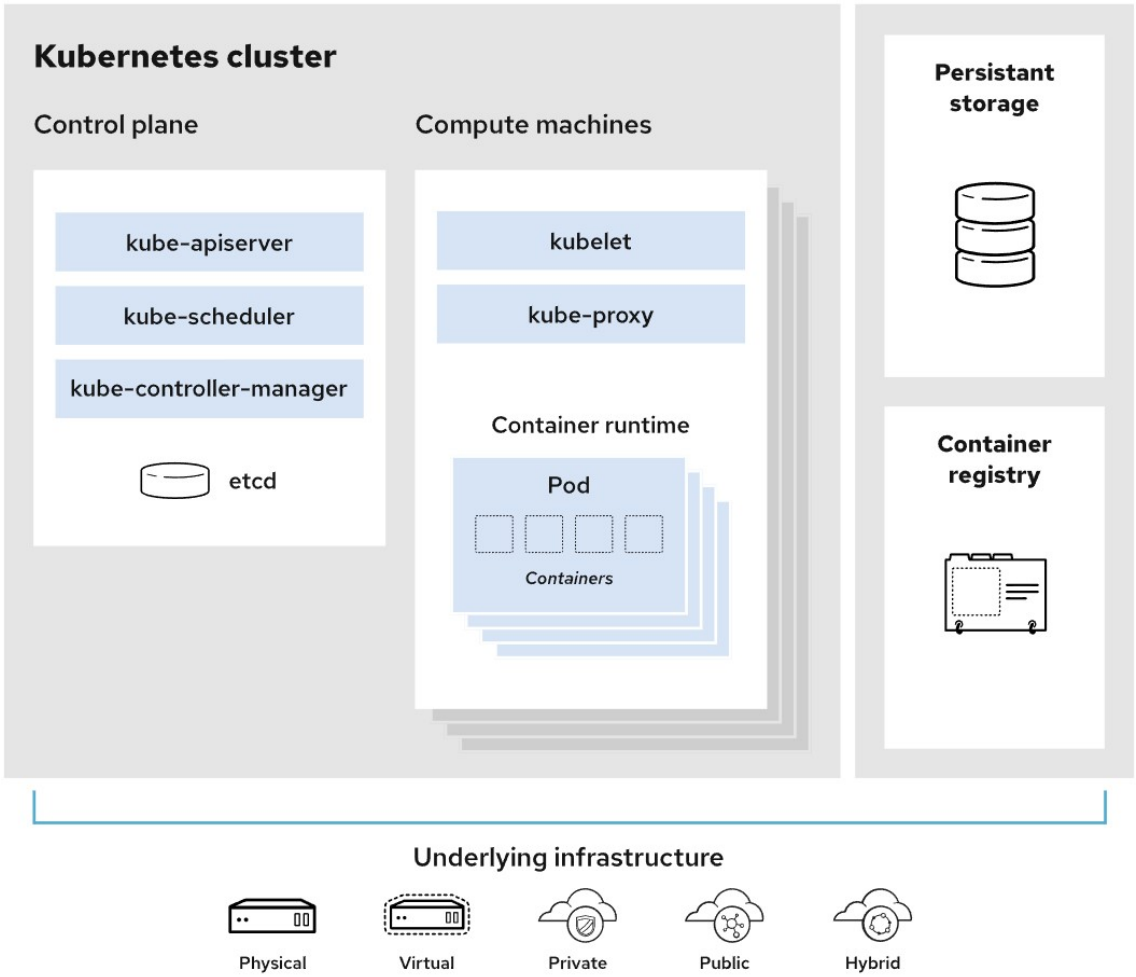
Το κύριο πλεονέκτημα που προσδίδει το Kubernetes, είναι ότι δίνει την δυνατότητα να δρομολογούνται και να εκτελούνται containers σε συστοιχίες. Γενικότερα μέσω του Kubernetes μπορεί κανείς να υλοποιήσει ένα σύστημα-υποδομή που τρέχει εξ ολοκλήρου σε containers ακόμα και σε περιβάλλοντα παραγωγής που οι απαιτήσεις είναι υψηλές. Το Kubernetes αναλαμβάνει όλη την αυτοματοποίηση διαχειριστικών ενεργειών που χρειάζεται ένα περιβάλλον με containers. Το πιο βασικό είναι ότι είναι υπεύθυνο για την επιλογή του μηχανήματος εντός μίας συστοιχίας στο οποίο θα τρέξει μία εφαρμογή. Αυτό γίνεται έξυπνα μέσω της επιτήρησης των διαθέσιμων πόρων κάθε μηχανήματος με σκοπό την αποφυγή αναπάντεχων διακοπών λειτουργίας. Επιπλέον επιτρέπει τον εύκολο έλεγχο και την αυτοματοποίηση της εκτέλεσης νέων εφαρμογών και της ανανέωσης ήδη υπαρχόντων. Δίνει επίσης την δυνατότητα να συνδεθεί και να προστεθεί αποθηκευτικός χώρος για την εκτέλεση stateful εφαρμογών. Μπορεί κανείς να κλιμακώσει τις εφαρμογές του και τους πόρους που καταλαμβάνουν χωρίς να διακοπεί η λειτουργία τους. Τέλος παρέχει υπηρεσίες όπως έλεγχο κατάστασης και αυτόματη αντικατάσταση, επανεκκίνηση, αντιγραφή και κλιμάκωση.

Το Kubernetes ωστόσο δεν επιτυγχάνει όλες αυτές τις λειτουργίες μόνο του out-of-the-box. Σε κάποια θέματα βασίζεται σε άλλα λογισμικά για να παρέχει τις πλήρεις του υπηρεσίες: Για το αρχείο εικόνων χρησιμοποιείται για παράδειγμα το Docker Registry [9], για τη δικτύωση για παράδειγμα το Calico [10] και για την ασφάλεια χρησιμοποιούνται προϊόντα όπως το SELinux [11], RBAC [12], OAUTH [13] κ.λπ.

2.2.1 Αρχιτεκτονική

Kubernetes Cluster λέγεται μία τρέχουσα συστοιχία Kubernetes. Αυτή η συστοιχία απαρτίζεται από δυο κομμάτια: το επίπεδο ελέγχου και τα υπολογιστικά μηχανήματα ή

αλλιώς κόμβους. Κάθε κόμβος τρέχει ένα λειτουργικό σύστημα Linux και μπορεί να είναι ένα φυσικό ή ένα εικονικό μηχάνημα. Κάθε κόμβος τρέχει κάψουλες και κάθε κάψουλα περιέχει ένα ή περισσότερα container. Το επίπεδο ελέγχου αναλύεται σε τέσσερα κύρια κομμάτια: το kube apiserver, το kube scheduler, το kube controller manager και το etcd. Ακολουθεί συνοπτική περιγραφή της λειτουργίας του κάθε κομματιού:



Εικόνα 2.3: *Επισκόπηση της αρχιτεκτονικής του Kubernetes*

- kube apiserver: Επικυρώνει και διαμορφώνει δεδομένα για τα αντικείμενα της διεπαφής προγραμματισμού που περιλαμβάνουν κάψουλες, υπηρεσίες, ελεγκτές αναπαραγωγής και άλλα. Κάνοντας χρήση μίας υπηρεσίας REST παρέχει διεπαφή για την κατάσταση της συστοιχίας, την οποία χρησιμοποιούν τόσο οι άνθρωποι-χειριστές (μέσω του εργαλείου γραμμής εντολών kubectl) όσο και άλλα στοιχεία εντός και εκτός της συστοιχίας.

- kube scheduler: Είναι μία διεργασία του επιπέδου ελέγχου που αναθέτει κάψουλες σε κόμβους. Ο scheduler καθορίζει ποιοι κόμβοι είναι έγκυρες τοποθετήσεις για κάθε κάψουλα στην σειρά προγραμματισμού σύμφωνα πάντα με περιορισμούς δηλωμένους από τον προγραμματιστή και τους διαθέσιμους πόρους. Ο scheduler στη συνέχεια ταξινομεί τους διαθέσιμους κόμβους ανάλογα με την καταλληλότητά τους και ορίζει

τελικά σε ποιον κόμβο θα τρέξει μία κάψουλα.

- kube controller manager: Είναι ένας daemon που ενσωματώνει τους βασικούς βρόγχους ελέγχου που έρχονται μαζί με το Kubernetes. Αυτό σημαίνει ότι παρακολουθεί την κατάσταση της συστοιχίας μέσω του apiserver και κάνει αλλαγές με σκοπό να μετακινήσει την τρέχουσα κατάσταση προς την επιθυμητή κατάσταση.

- etcd [14]: Το Kubernetes χρειάζεται να καταγράφει κάπου την κατάσταση όλων των κινητών κομματιών που το απαρτίζουν. Το etcd είναι ένα σύστημα αποθήκευσης τύπου key-value που λειτουργεί ως η μοναδική πηγή αλήθειας για την κατάσταση των κόμβων, των καψουλών, των εφαρμογών και όλων των άλλων κομματιών ενός συστήματος Kubernetes.

Τέλος έχουμε τα κομμάτια που απαρτίζουν τους υπολογιστικούς κόμβους τα οποία είναι τα εξής:

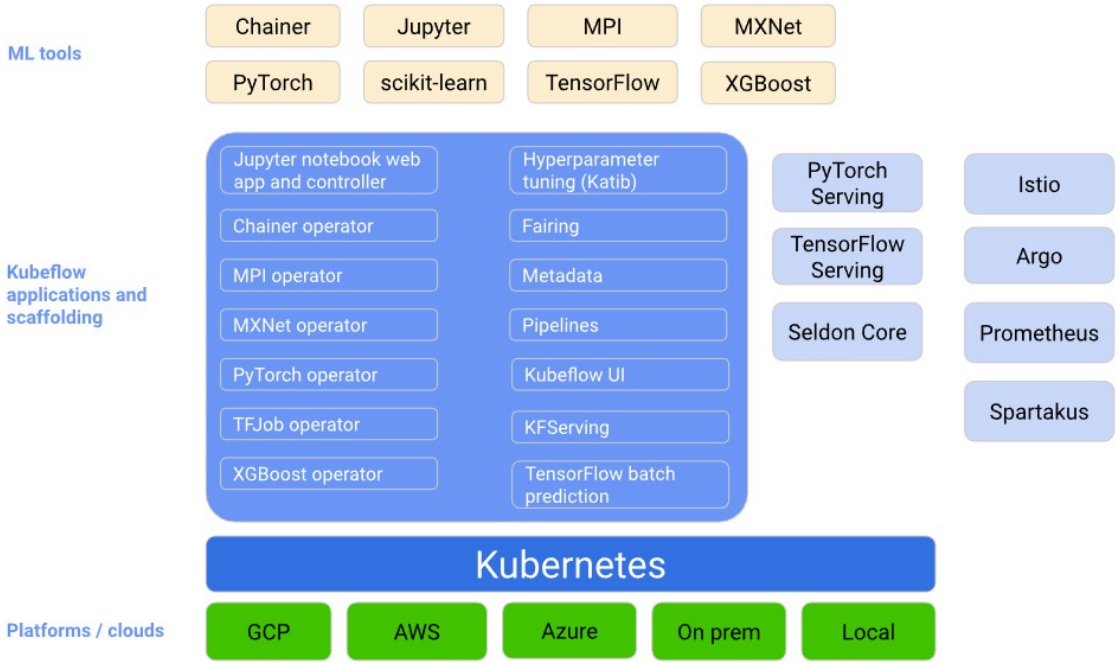
- kubelet: Είναι ο κύριος πράκτορας που τρέχει σε κάθε κόμβο. Μπορεί να καταχωρήσει τον κόμβο στον apiserver χρησιμοποιώντας συνήθως το όνομα του υπολογιστή. Το kubelet λειτουργεί με την χρήση PodSpec. Το PodSpec είναι ένα αντικείμενο YAML ή JSON που περιγράφει την λειτουργία μίας κάψουλας. Το kubelet λαμβάνει αυτά τα PodSpec μέσω του apiserver και διασφαλίζει ότι οι κάψουλες που περιγράφονται εντός των αντικειμένων λειτουργούν και ότι είναι υγιείς.

- kube proxy: Λειτουργεί ως διακομιστής μεσολάβησης για τις διάφορες υπηρεσίες που τρέχουν εντός μιας συστοιχίας. Μπορεί να προωθήσει ροές πληροφορίας πρωτοκόλλων TCP, UDP και SCTP.

- container runtime: Γνωστό και ως μηχανή εκτέλεσης container [15]. Είναι λογισμικό που μπορεί να τρέξει container σε ένα λειτουργικό σύστημα υποδοχής. Σε μία αρχιτεκτονική container, τα runtimes είναι υπεύθυνα για την φόρτωση εικόνων container από ένα αποθετήριο, την παρακολούθηση των πόρων του τοπικού συστήματος, την απομόνωση των πόρων του συστήματος για χρήση ενός container και τη διαχείριση του κύκλου ζωής των container.

2.3 Kubeflow

Το Kubeflow είναι μία σουίτα εργαλείων που επιτρέπει σε επιστήμονες δεδομένων να δημιουργούν ροές εργασίας μηχανικής μάθησης ευκολότερα σε περιβάλλοντα Kubernetes. Μέσω του Kubeflow δύναται να δημιουργηθεί μία ολοκληρωμένη λύση μηχανικής μάθησης με την χρήση γνωστών εργαλείων όπως τα Jupyter Notebooks [16], το PyTorch [17] και το TensorFlow [18]. Το Kubeflow αυτοματοποιεί την διαδικασία εγκατάστασης, ρύθμισης και συντήρησης των αναγκαίων προαπαιτούμενων συνθετικών μερών προκειμένου ένας χρήστης να μπορεί να αφοσιωθεί στην δημιουργία του μοντέλου μηχανικής μάθησης και όχι στην υποθάλπουσα υποδομή του Kubernetes.



Εικόνα 2.4: *Επισκόπηση της αρχιτεκτονικής του Kubeflow*

2.3.1 Kubeflow Pipelines

Μία διοχέτευση είναι μία περιγραφή μίας ροής εργασίας ML συμπεριλαμβανομένων όλων των στοιχείων στη ροή εργασίας και του τρόπου με τον οποίο συνδυάζονται με τη μορφή γραφήματος. Η διοχέτευση περιλαμβάνει τον ορισμό των εισόδων (παραμέτρων) που απαιτούνται για την εκτέλεση της διοχέτευσης και τις εισόδους και εξόδους κάθε στοιχείου.

Ένα στοιχείο διοχέτευσης είναι ένα αυτόνομο σύνολο κώδικα, συσκευασμένο ως εικόνα Docker, που εκτελεί ένα βήμα στη διοχέτευση. Για παράδειγμα, ένα στοιχείο μπορεί να είναι υπεύθυνο για την προεπεξεργασία δεδομένων (preprocessing), τον μετασχηματισμό δεδομένων (transformation), την εκπαίδευση μοντέλων και ούτω καθεξής.

Οι διοχετεύσεις Kubeflow είναι μια πλατφόρμα για την δημιουργία και την εφαρμογή φορητών, κλιμακόσιμων ροών εργασιών μηχανικής μάθησης που βασίζονται σε containers. Η πλατφόρμα αποτελείται από:

- Μία διεπαφή χρήστη για την διαχείριση και την παρακολούθηση ιστορικού πειραμάτων, εργασιών και εκτελέσεων.
- Ένα σύστημα για τον προγραμματισμό ροών εργασίας ML πολλαπλών βημάτων.
- Ένα πακέτο ανάπτυξης λογισμικού (SDK) για τον χειρισμό διοχετεύσεων και στοιχείων.
- Notebooks για αλληλεπίδραση με το σύστημα με την χρήση του SDK.

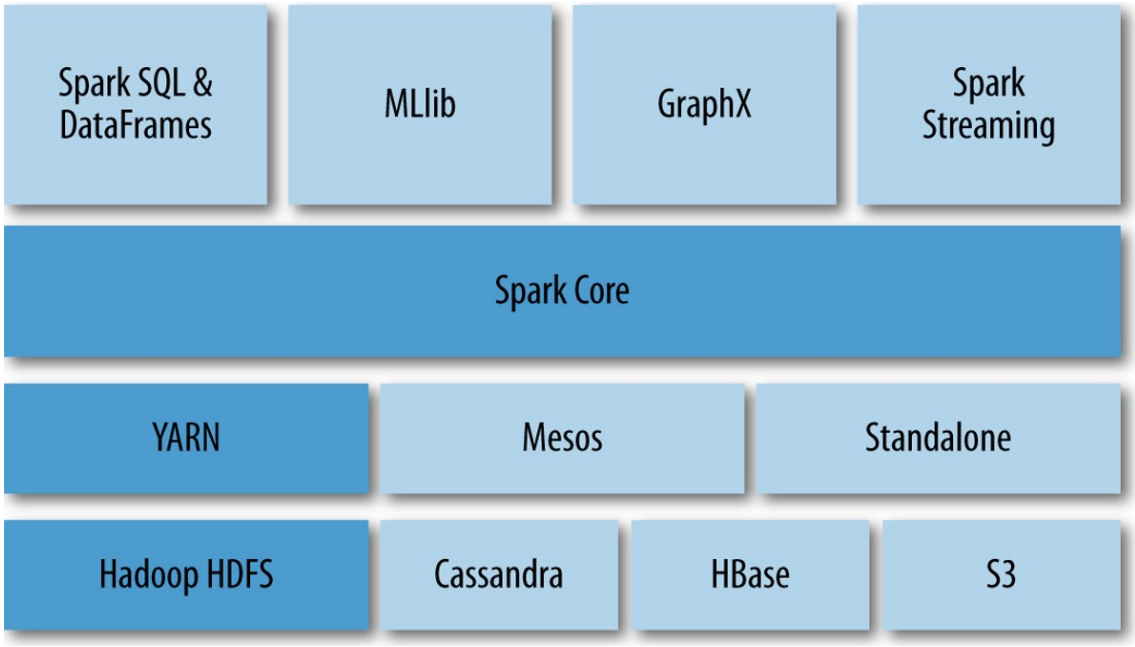
Παράδειγμα μίας διοχέτευσης βρίσκεται στο παράρτημα Α΄.1

2.4 Apache Spark

Η αρχιτεκτονική του Apache Spark έχει την βάση της στο ανθεκτικό κατανεμημένο σύνολο δεδομένων (RDD), ένα σύνολο πολλαπλών στοιχείων δεδομένων μόνο για ανάγνωση που διανέμονται σε ένα σύμπλεγμα μηχανών, το οποίο διατηρείται με τρόπο ανεκτικό σε σφάλματα. Πέραν του RDD API παρέχεται το Dataframe API το οποίο είναι δομημένο πάνω στο RDD API, καθώς και το Dataset API.

Το Spark και τα RDD αναπτύχθηκαν το 2012 ως απάντηση στους περιορισμούς στο υπολογιστικό μοτίβο MapReduce [19], το οποίο επιβάλλει μια συγκεκριμένη προγραμματιστική τακτική τύπου διαίρει και βασίλευε στα κατανεμημένα προγράμματα: Τα προγράμματα MapReduce διαβάζουν δεδομένα εισόδου από το δίσκο, χαρτογραφούν μια συνάρτηση στα δεδομένα (χωρίς να την εφαρμόζουν αμέσως), μειώνουν τα αποτελέσματα της χαρτογράφησης συνήθως με μία συνάρτηση συσσωμάτωσης (aggregation) και αποθηκεύουν τα αποτελέσματα μείωσης στο δίσκο.

Εντός του Apache Spark η διαχείριση της ροής εργασίας γίνεται ως κατευθυνόμενος ακυκλικός γράφος (DAG). Οι κόμβοι αντιπροσωπεύουν τα RDD ενώ οι ακμές αντιπροσωπεύουν τις λειτουργίες στα RDD.



Εικόνα 2.5: *Το οικοσύστημα του Spark. Από πάνω προς τα κάτω: Τα διαφορετικά APIs που προσφέρει το Spark, ο πυρήνας του Spark, ο διαχειριστής συστοιχίας και τέλος το κατανεμημένο σύστημα αποθήκευσης.*

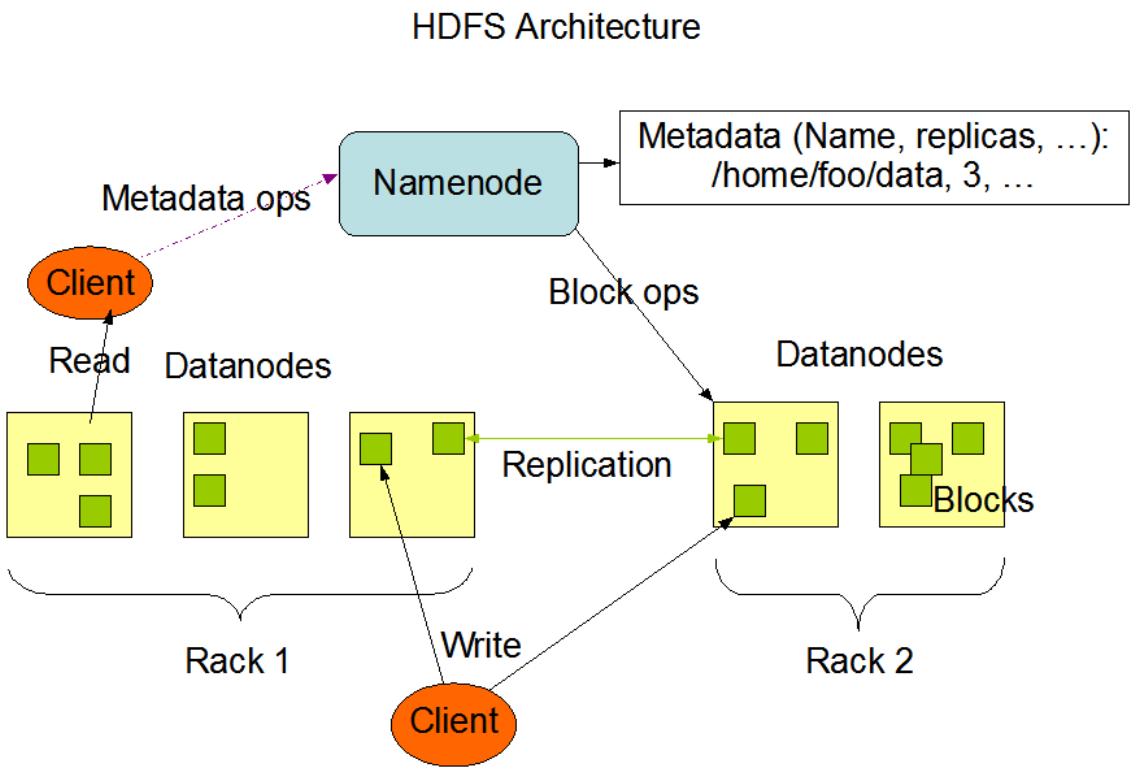
Το Apache Spark χρειάζεται έναν διαχειριστή συστοιχίας και ένα κατανεμημένο σύστημα αποθήκευσης. Για τη διαχείριση συστοιχίας, το Spark υποστηρίζει αυτόνομο (εγγενές σύμπλεγμα Spark, όπου μπορείτε να εκκινήσετε ένα σύμπλεγμα είτε με μη αυτόματο τρόπο είτε να χρησιμοποιήσετε τα σενάρια εκκίνησης που παρέχονται από το πακέτο εγκατάστασης. Είναι επίσης δυνατή η εκτέλεση αυτών των δαιμόνων σε ένα μόνο μηχάνημα για δοκιμή), Hadoop YARN [20], Apache Mesos [21] ή Kubernetes.

Για κατανεμημένη αποθήκευση, το Spark μπορεί να διασυνδέεται με μια μεγάλη ποικιλία λύσεων, συμπεριλαμβανομένων των Hadoop Distributed File System (HDFS) [22], Cassandra [23], OpenStack Swift [24], Amazon S3 [25]. Το Spark υποστηρίζει επίσης μια ψευδο-κατανεμημένη τοπική λειτουργία, που χρησιμοποιείται συνήθως μόνο για σκοπούς ανάπτυξης ή δοκιμής, όπου δεν απαιτείται κατανεμημένη αποθήκευση και μπορεί να χρησιμοποιηθεί το τοπικό σύστημα αρχείων. Σε ένα τέτοιο σενάριο, το Spark εκτελείται σε ένα μόνο μηχάνημα με έναν εργάτη ανά πυρήνα CPU.

2.5 Hadoop Distributed File System

Το κατανεμημένο σύστημα αρχείων Hadoop (HDFS) είναι ένα κατανεμημένο, επεκτάσιμο και φορητό σύστημα αρχείων γραμμένο σε Java για το Hadoop framework. Παρέχει εντολές γραμμής εντολών και διεπαφή προγραμματισμού εφαρμογών Java (API) που είναι παρόμοιες με άλλα συστήματα αρχείων. Το HDFS διαθέτει τρεις υπηρεσίες ως εξής:

- Name Node
- Secondary Name Node
- Data Node



Εικόνα 2.6: *Η αρχιτεκτονική του HDFS. Ο πελάτης επικοινωνεί με τον κόμβο ονομάτων για όποια διεργασία χρειάζεται ο οποίος με την σειρά του ανακατευθύνει το αίτημά του στον αρμόδιο κόμβο δεδομένων. Έτσι αποφεύγεται το bottleneck. Οι κόμβοι δεδομένων επικοινωνούν μεταξύ τους για τις ανάγκες αναπαραγωγής των δεδομένων σε πολλαπλά αντίγραφα.*

Ο κόμβος ονομάτων καθώς και ο δευτερεύων κόμβος ονομάτων είναι κόμβος αφέντη ενώ ο κόμβος δεδομένων είναι κόμβος σκλάβου.

Πιο αναλυτικά:

- Κόμβος ονομάτων: Το HDFS αποτελείται από έναν μόνο κόμβο ονομάτων που ονομάζεται κύριος κόμβος. Ο κύριος αυτός κόμβος μπορεί να παρακολουθεί αρχεία, να διαχειρίζεται το σύστημα αρχείων και κρατάει τα μεταδεδομένα όλων των αποθηκευμένων δεδομένων. Συγκεκριμένα, ο κόμβος ονομάτων περιέχει τις λεπτομέρειες του αριθμού των block, τις θέσεις του κόμβου δεδομένων στον οποίο είναι αποθηκευμένα τα δεδομένα, που αποθηκεύονται τα αντίγραφα και άλλες λεπτομέρειες. Ο κόμβος ονόματος έχει άμεση επαφή με την διεπαφή ή την γραμμή εντολών και είναι υπεύθυνος για τον συντονισμό των κόμβων δεδομένων.

- Κόμβος δεδομένων: Ένας κόμβος δεδομένων αποθηκεύει δεδομένα ως block. Αυτός είναι επίσης γνωστός ως κόμβος σκλάβος και είναι υπεύθυνος για την αποθήκευση των δεδομένων στο HDFS, τα οποία προορίζονται προς ανάγνωση και εγγραφή από τον χρήστη. Οι κόμβοι δεδομένων λειτουργούν με την μορφή δαίμων. Κάθε κόμβος δεδομένων στέλνει ένα μήνυμα heartbeat στον κόμβο ονομάτων κάθε τρία δευτερόλεπτα και μεταδίδει ότι είναι ζωντανός. Με αυτόν τον τρόπο, όταν ο κόμβος ονομάτων δεν λάβει μήνυμα απάντησης από έναν κόμβο δεδομένων για 2 λεπτά, θα θεωρήσει αυτόν τον κόμβο ως νεκρό και θα ξεκινήσει τη διαδικασία των επαναλήψεων μπλοκ σε κάποιον άλλο κόμβο δεδομένων.

- Δευτερεύων κόμβος ονόματος: Είναι υπεύθυνος μόνο για τη φροντίδα των σημείων ελέγχου (checkpoints) των μεταδεδομένων του συστήματος αρχείων που βρίσκονται στον κόμβο ονόματος. Είναι επίσης γνωστός και ως κόμβος του σημείου ελέγχου (checkpoint node). Είναι ουσιαστικά ο βοηθητικός κόμβος για τον κόμβο ονομάτων.

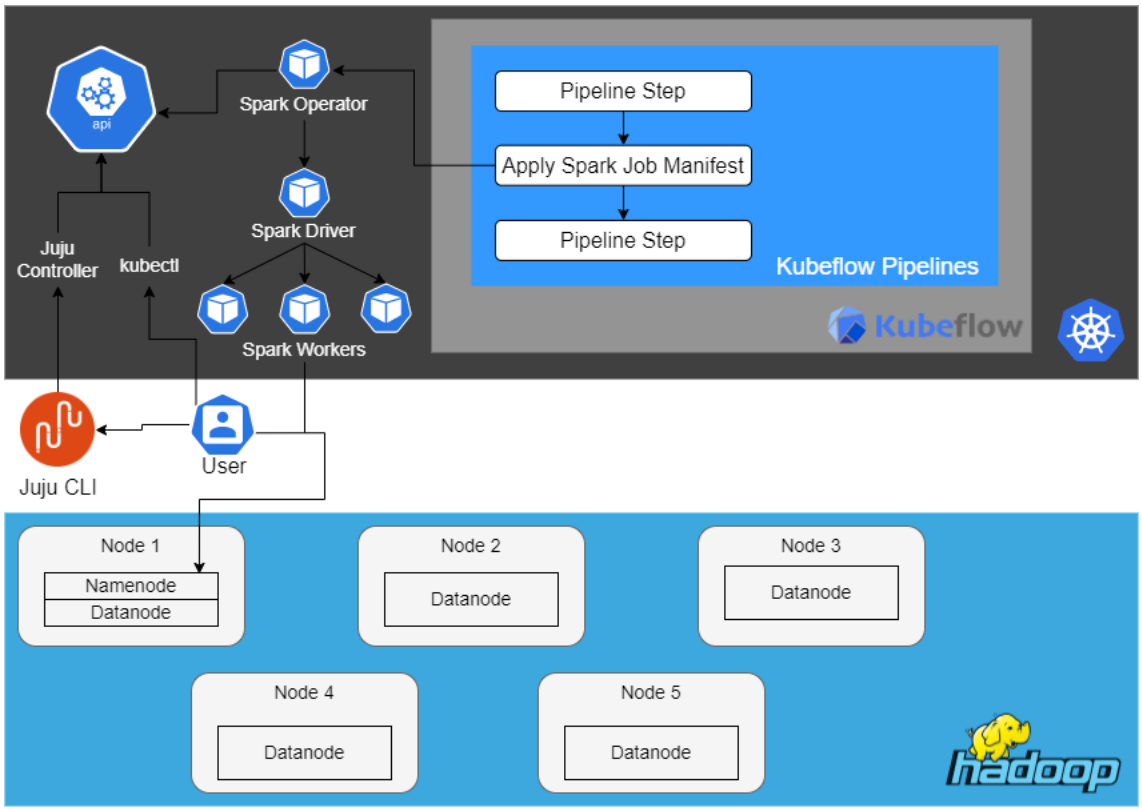
Το Hadoop έχει συνήθως έναν μοναδικό κόμβο ονομάτων συν μία συστοιχία κόμβων δεδομένων, αν και είναι διαθέσιμη η επιλογές χρήσης περισσότερων κόμβων ονομάτων λόγω της κρισιμότητας του. Κάθε κόμβος δεδομένων εξυπηρετεί μπλοκ δεδομένων μέσω του δικτύου χρησιμοποιώντας ένα πρωτόκολλο block συγκεκριμένο στο HDFS. Το σύστημα αρχείων χρησιμοποιεί υποδοχές TCP/IP για επικοινωνία.

Το HDFS αποθηκεύει μεγάλα αρχεία (συνήθως μεγέθους από gigabyte έως terabyte) σε πολλαπλά μηχανήματα. Επιτυγχάνει αξιοπιστία με την αναπαραγωγή των δεδομένων σε πολλούς κεντρικούς υπολογιστές και, ως εκ τούτου, θεωρητικά δεν απαιτεί πλεονάζουσα συστοιχία ανεξάρτητων δίσκων (RAID) αποθήκευσης στους κεντρικούς υπολογιστές. Με την προεπιλεγμένη τιμή αναπαραγωγής, τρία, τα δεδομένα αποθηκεύονται σε τρεις κόμβους: δύο στο ίδιο rack και έναν σε διαφορετικό rack. Οι κόμβοι δεδομένων μπορούν να συνομιλούν μεταξύ τους για να εξισορροπήσουν εκ νέου τα δεδομένα, να μετακινήσουν αντίγραφα και να διατηρήσουν την αναπαραγωγή των δεδομένων σε υψηλά επίπεδα. Το HDFS δεν είναι πλήρως συμβατό με το POSIX με σκοπό την προσφορά αυξημένων δυνατοτήτων. Λόγω της έλλειψης συμβατότητας έχει επιτευχθεί αυξημένη απόδοση για τη μεταφορά δεδομένων (throughput) και η υποστήριξη για λειτουργίες που δεν περιέχονται στο POSIX, όπως το append.

Κεφάλαιο 3

Αρχιτεκτονική

Στο παρόν κεφάλαιο περιγράφεται η μέθοδος που ακολουθήθηκε για να γίνει η δόμηση της συστοιχίας και των συστημάτων που την απαρτίζουν, η αρχιτεκτονική του μοντέλου μηχανικής μάθησης που δημιουργήθηκε προκειμένου να παρουσιαστεί η λειτουργία του Spark ως εγγενές μέλος της συστοιχίας Kubernetes καθώς και η αρχιτεκτονική του συστήματος υποστήριξης αποφάσεων που υλοποιήθηκε.



Εικόνα 3.1: *Επισκόπηση της τελικής αρχιτεκτονικής. Στην εικόνα περιγράφεται η λειτουργία της συστοιχίας με έναν χρήστη. Στον πραγματικό κόσμο ο χρήστης αυτός θα αναλύονταν σε περισσότερους. Αναφορικά ένα διαχειριστής θα είχε πρόσβαση στο Juju CLI και στο HDFS CLI όπως φαίνεται από τα βέλη (αριστερά και δεξιά του χρήστη) ενώ διάφοροι χρήστες (π.χ διαχειριστής ή προγραμματιστής) θα είχαν πρόβαση στο kubectl ο καθένας με συγκεκριμένα δικαιώματα. Επιπλέον φαίνεται η διαδικασία εκκίνησης μιας συστοιχίας Spark από την αρχή που βρίσκεται μέσα σε ένα στάδιο ενός Kubeflow Pipeline στη συνέχεια μεταβαίνει το αίτημα στον Spark Operator ο οποίος με την σειρά του εκκινεί έναν driver.*

3.1 Αρχιτεκτονική Συστοιχίας

Σε αυτό το σημείο θα γίνει μία επισκόπηση της αρχιτεκτονικής της συστοιχίας. Το πλήθος των μηχανημάτων το οποίο αξιοποιήθηκε για την εκπόνηση της εργασίας προέρχεται από μία συστοιχία OpenStack [26] του CSLab και αποτελείται από πέντε εικονικές μηχανές. Τα συστατικά στοιχεία της αρχιτεκτονικής είναι τα εξής:

- MicroK8s: Πρόκειται για μία λύση της Canonical που επιτρέπει την εύκολη υλοποίηση μίας συστοιχίας Kubernetes σε όλα τα λειτουργικά (Windows, macOS, Linux) ενώ ταυτόχρονα έχει βελτιστοποιήσεις που ελαχιστοποιούν το αντίκτυπο στην RAM. Μαζί με την βασική εγκατάσταση του Kubernetes έρχονται κάποια δημοφιλή και πολύ βασικά πρόσθετα που δεν λείπουν από καμία συστοιχία όπως το Calico για την ενδοεπικοινωνία των καψουλών και το Prometheus για την συλλογή μετρικών. Το MicroK8s έχει και υποστήριξη για περιβάλλον υψηλής διαθεσιμότητας με την χρήση της βάσης δεδομένων Dqlite [27] η οποία είναι μία γρήγορη, ενσωματωμένη μόνιμη βάση δεδομένων SQL η οποία χρησιμοποιεί τον αλγόριθμο Raft [28] για την εκλογή ηγέτη.

- Kubeflow: Για τους σκοπούς της εργασίας επιλέχθηκε η έκδοση Charmed Kubeflow [29] που διατηρείται από την Canonical. Πρόκειται για μία έκδοση του Kubeflow που βασίζεται στην τεχνολογία των Charmed Operators και ως εκ τούτου χρειάζεται ένα επιπλέον εργαλείο που λέγεται Juju [30] που είναι αναγκαίο για την εγκατάσταση και την διαχείριση charms.

- Apache Spark: Ένας από τους στόχους της εργασίας ήταν η χρήση του Spark εντός του οικοσυστήματος μίας συστοιχίας Kubernetes. Για τον σκοπό αυτό επιλέχθηκε η χρήση του Kubernetes Operator for Apache Spark ή αλλιώς spark-on-k8s-operator [31].

- Hadoop Distributed File System: Αυτό το κατανεμημένο σύστημα αποθήκευσης χρησιμοποιείται από τις ροές εργασίας Spark ως αποθηκευτικός χώρος ενδιάμεσων προσωρινών αρχείων (αλλά όχι shuffle files), ως πηγή συνόλου δεδομένων αλλά και αποθετήριο αποτελεσμάτων από προγράμματα. Ο τρόπος που εφαρμόστηκε είναι με τυπική εγκατάσταση και στους 5 κόμβους όπου ο ένας επιτελεί έργο κόμβου ονομάτων και δευτερεύοντος κόμβου ονομάτων, και οι 5 τρέχουν κόμβους δεδομένων.

Προφανώς στην προκειμένη συστοιχία δεν υπάρχει διαχωρισμός μεταξύ αποθηκευτικού χώρου και υπολογιστικών πόρων.

3.2 Ροή Μηχανικής Μάθησης σε Spark

Για τους σκοπούς αποτίμησης του συστήματος επιλέχθηκε η εκτέλεση μίας σχετικά απλής ροής εργασίας μηχανικής μάθησης. Η ροή αφορά ένα σύνολο δεδομένων που προέρχεται από το γνωστό UC Irvine Machine Learning αποθετήριο. Συγκεκριμένα στόχος είναι η έναρξη και η διαχείριση της κατάστασης μίας ροής εργασίας Spark στα πλαίσια ροής μίας διοχέτευσης Kubeflow. Το σύνολο δεδομένων [32] αφορά σε στοιχεία που προέρχονται από

την απογραφή πληθυσμού των ΗΠΑ το 1994. Στόχος του μοντέλου μηχανικής μάθησης είναι η ταξινόμηση (classification) των ατόμων σε δύο βαθμίδες βάσει εισοδήματος. Η διαδικασία εκπαίδευσης και αποτίμησης αποτελείται από τα εξής στάδια:

- Μεταφόρτωση στο HDFS: Πριν γίνεται οποιαδήποτε ενέργεια τα δεδομένα πρέπει να μεταφορτωθούν στο HDFS προκειμένου να είναι προσβάσιμα από όλους τους δυνατούς κόμβους-εργάτες του Spark. Αυτό είναι το μόνο στάδιο που κατά κάποιον τρόπο γίνεται «χειροκίνητα» από έναν client που επικοινωνεί με τον κόμβο ονομάτων και έχει στο τοπικό σύστημα αρχείων τα αρχεία που αποτελούν το σύνολο δεδομένων.

- Εξερεύνηση του συνόλου δεδομένων: Σε αυτό το στάδιο έγινε ένας έλεγχος προκειμένου να γίνει εξοικείωση με τα δεδομένα και να αποφασιστεί τι είδους μετασχηματισμοί θα εφαρμοστούν. Για τους σκοπούς αυτού του σταδίου χρησιμοποιήθηκε ένα διαδραστικό Jupyter Notebook που έτρεχε σε ένα Docker container σε υπολογιστή εκτός της συστοιχίας.

- Προεπεξεργασία: Πριν την εκπαίδευση του μοντέλου είναι αναγκαία η αντιμετώπιση προβληματικών εγγραφών αλλά και η κωδικοποίηση τύπου one-hot σε κατηγορικές μεταβλητές. Τα επεξεργασμένα δεδομένα αποθηκεύονται προσωρινά στο HDFS.

- Εκπαίδευση: Σε αυτό το σημείο γίνεται η εκπαίδευση ενός μοντέλου μηχανικής μάθησης στο σύνολο εκπαίδευσης.

- Πρόβλεψη: Τέλος γίνεται η αποτίμηση του μοντέλου βάσει της απόδοσης του στην ταξινόμηση καταγραφών από το σύνολο ελέγχου.

3.3 Σύστημα Υποστήριξης Αποφάσεων

Μετά την αποτίμηση του συστήματος και την απόκτηση δεδομένων που αφορούν στην απόδοση του συστήματος βάσει διαφορετικών συνδυασμών ρυθμίσεων κρίθηκε χρήσιμη η ανάπτυξη ενός εργαλείου που ενοποιεί την πληροφορία που αντλήθηκε. Το εργαλείο αυτό πρόκειται για ένα σύστημα υποστήριξης αποφάσεων. Το σύστημα αυτό αποτελείται από:

- Βάση δεδομένων: Η πληροφορία που αντλήθηκε μετά από αποτίμηση του συστήματος. Περισσότερα για αυτό θα αναλυθούν στο επόμενο κεφάλαιο.

- Μοντέλο: Πρόκειται για το σύνολο των παραμέτρων εισόδων που μπορούν να οριστούν και το μοντέλο μηχανικής μάθησης που αναπτύχθηκε.

- Διεπαφή: Όλα τα συστήματα υποστήριξης αποφάσεων πρέπει να έχουν μία φιλική προς τον χρήστη διεπαφή με σκοπό την καλύτερη εξυπηρέτηση του χρήστη και την σωστή λειτουργία του συστήματος.

Κεφάλαιο 4

Περιγραφή Εγκατάστασης Συστοιχίας

Στο παρόν κεφάλαιο θα περιγραφεί η διαδικασία που ακολουθήθηκε για την υλοποίηση της συστοιχίας Kubernetes και όλων των οικοσυστημάτων που την απαρτίζουν δηλαδή το Kubernetes, το Kubeflow, το HDFS και το Spark.

4.1 Kubernetes

Προαπαιτούμενο βήμα για την εγκατάσταση του Kubeflow είναι η αρχικοποίηση μίας συστοιχίας Kubernetes. Η συστοιχία υλοποιήθηκε σε 5 εικονικά μηχανήματα που παρείχε το εργαστήριο CSLab με λειτουργικό σύστημα Ubuntu 20.04 LTS. Τα χαρακτηριστικά των μηχανημάτων είναι 4 vCPU, 8GB RAM, 60 GB χωρητικότητα δίσκου. Πρώτα πρέπει να εγκατασταθεί το microk8s σε όλα τα μηχανήματα μέσω του snap. Για λόγους αναπαραγωγιμότητας αξίζει να σημειωθεί ότι η έκδοση του microk8s που χρησιμοποιήθηκε είναι η 1.21 η οποία έρχεται με την έκδοση 1.21.13 του Kubernetes. Αφού εγκατασταθεί το microk8s, εντάσσονται όλοι οι κόμβοι σε μία κοινή συστοιχία και μετά από λίγα λεπτά η συστοιχία είναι έτοιμη. Η βασική εγκατάσταση του microk8s έρχεται μαζί με ένα αποθετήριο δημοφιλών deployments τα οποία συχνά είναι χρήσιμα σε περιβάλλοντα ανάπτυξης εφαρμογών Kubernetes. Τα deployments που εφαρμόστηκαν είναι τα εξής:

- Metrics server: Καταγράφει σε τακτά χρονικά διαστήματα την απορρόφηση πόρων από τις διάφορες οντότητες που ζουν στη συστοιχία. Μέσω αυτής της υπηρεσίας καθίσταται δυνατή η αυτόματα κλιμάκωση των deployments από το Kubernetes.

- Calico: Το Calico είναι ένα εργαλείο για δικτύωση και ασφάλεια δικτύου. Πρόκειται για ένα CNI [33] plugin ανοικτού κώδικα που διαχειρίζεται το επίπεδο δικτύου μεταξύ διαφορετικών καψουλών (pod).

- K8s dashboard: Ένας πίνακας ελέγχου σε μορφή γραφικού περιβάλλοντος προσβάσιμος από φυλλομετρητή που επικοινωνεί με το kube-apiserver και προσφέρει μία εικόνα στο τι συμβαίνει στην συστοιχία αλλά και μερικά εργαλεία διαχείρισης των διαφορετικών οντοτήτων που ζουν μέσα σε αυτήν.

- Hostpath provisioner: Χρήσιμο για δημιουργία PersistentVolumes που τοποθετούνται στο τοπικό σύστημα αρχείων. Αυτό το πρόσθετο είναι χρήσιμο κατά την διαδικασία ανάπτυξης εφαρμογών αλλά δεν προτείνεται η χρήση του σε παραγωγικά περιβάλλοντα

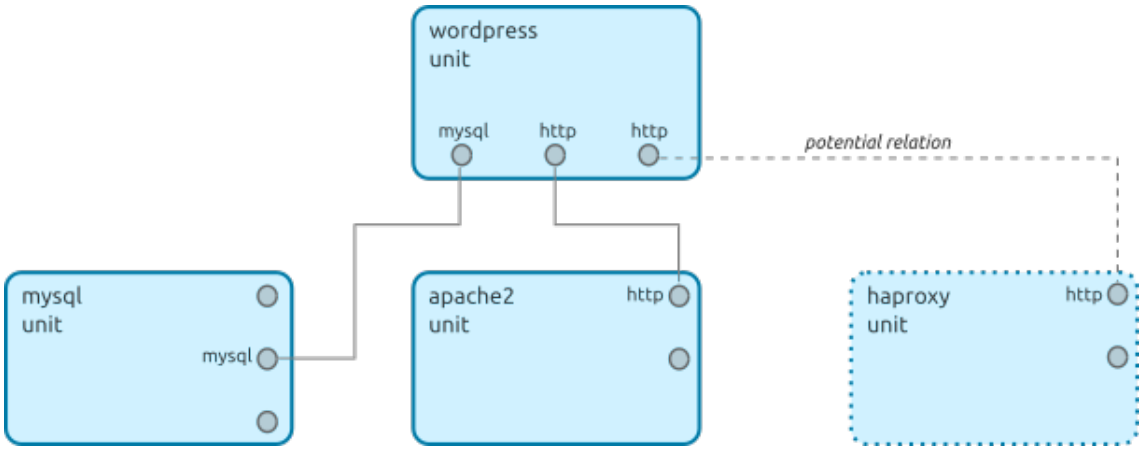
καθώς τα PersistentVolumes αυτά δεν μπορούν αν μετακινηθούν από το μηχάνημα-οικοδεσπότη και είναι προσβάσιμα μόνο εφαρμογές που τρέχουν στο ίδιο.

- coreDNS: Εντός μίας συστοιχίας Kubernetes κάθε κάψουλα αλλά και κάθε υπηρεσία έχουν μία καταγραφή DNS. Για αυτό τον σκοπό το Kubernetes χρησιμοποιεί μία κάψουλα DNS καθώς και μία υπηρεσία DNS (Service). Επιπλέον ρυθμίζει την μηχανή εκτέλεσης container (kubelet-kubelite) του συστήματος-οικοδεσπότη να πει στα εκάστοτε container ότι όταν θέλουν επίλυση ονόματος DNS να απευθυνθούν στην τοπική υπηρεσία DNS. Το coreDNS είναι ένας εξυπηρετητής DNS ο οποίος επιτελεί λειτουργίες αρχειοθέτησης ονομάτων-διευθύνσεων, ανακατεύθυνσης κ.λπ.
- Ingress: Το Ingress επιτρέπει την αντιστοίχιση διαφόρων αιτημάτων σε διαφορετικά backend βάσει κανόνων που ορίζονται μέσω του Kubernetes API.

4.2 Kubeflow

4.2.1 Juju

Πριν εγκατασταθεί το πακέτο Charmed Kubeflow πρέπει να εγκατασταθεί μέσω snap ένα εργαλείο που προαναφέρθηκε ονόματι Juju. Το Juju είναι ουσιαστικά ένα εργαλείο διαχείρισης μεγάλων και σύνθετων deployments. Συμβάλει στην διαχείριση του συνολικού κύκλου ζωής των deployments και "συνδέει" διάφορα deployments μεταξύ τους με εύκολο τρόπο. Για παράδειγμα έστω ότι θέλουμε να βάλουμε ένα deployment που βασίζει την λειτουργία του σε ένα key-value αποθηκευτικό μέσο. Το Juju μοντελοποιεί αυτή την εξάρτηση ως μία διεπαφή που χρειάζεται να συνδεθεί με μία άλλη συμβατή διεπαφή. Μπορούμε να επιλέξουμε συνεπώς μία οποιαδήποτε key-value λύση όπως Cassandra, Hbase ή Bigtable και να τη συνδέσουμε με το deployment που θέλουμε. Ως εκ τούτου μας δίνεται η δυνατότητα να έχουμε πλήρη ευελιξία στη δόμηση ενός project. Για την εγκατάσταση του Kubeflow

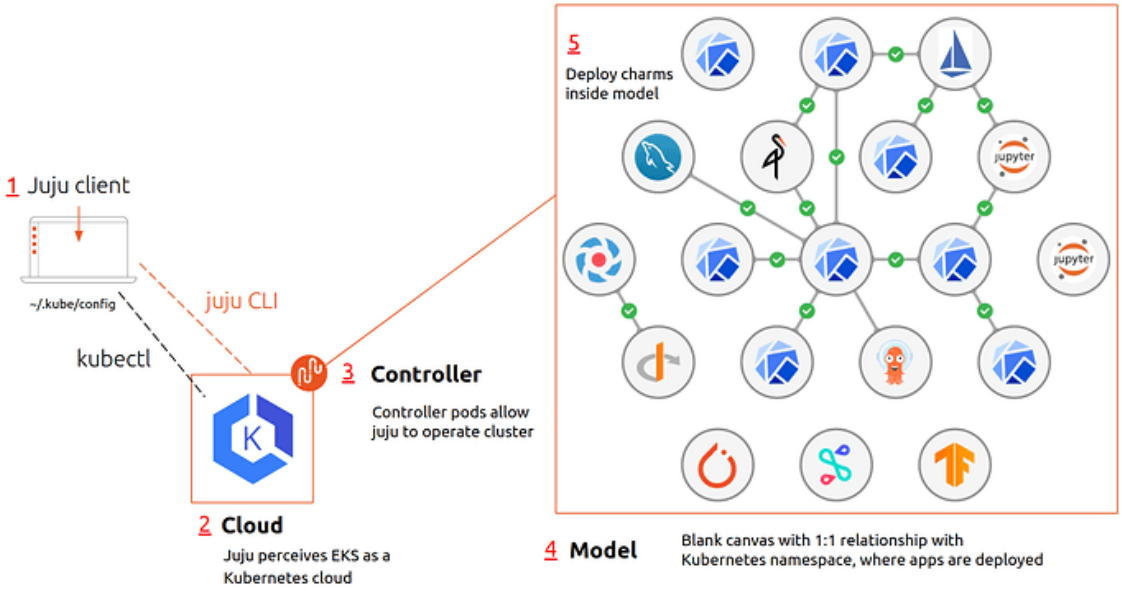


Εικόνα 4.1: *Αναπαράσταση της διαχείρισης των εξαρτήσεων από το Juju*

χρειάζονται να γίνουν τα εξής βήματα:

- Εκκίνηση ενός ελεγκτή [34] (controller) Juju στη συστοιχία Kubernetes. Ο ελεγκτής Juju λειτουργεί ως ένας ή περισσότεροι πράκτορες ελέγχου Juju. Οι διεπαφές-πελάτες

Juju, π.χ. Juju CLI, αλληλεπιδρούν με έναν πράκτορα ελέγχου για να στείλουν εντολές και να λάβουν πληροφορίες από αυτόν. Ο πράκτορας ελέγχου αλληλεπιδρά επίσης με το backend της βάσης δεδομένων του Juju για τη διατήρηση της κατάστασης και με τον πάροχο υπολογιστικού νέφους (στην περίπτωσή μας είναι το επιτόπιο microk8s) για να διαχειριστεί τους πόρους υπολογισμού, δικτύωσης και αποθήκευσης για τις εφαρμογές που τρέχουν.



Εικόνα 4.2: *Ο τρόπος λειτουργίας του Juju από αφαιρετική σκοπιά*

- Προσθήκη ενός μοντέλου στον ελεγκτή. Το μοντέλο έχει ουσιαστικά την ίδια λειτουργία με ένα Kubernetes namespace και αντιστοιχίζεται πλήρως με ένα ομώνυμο namespace που δημιουργείται στη συστοιχία. Λειτουργεί αφαιρετικά ως βάση για όλα τα deployments που εγκαθιστώνται.

- Τώρα πλέον μπορούμε να εγκαταστήσουμε οποιαδήποτε έκδοση Charmed Kubeflow επιθυμούμε. Για τους σκοπούς της διπλωματικής κρίθηκε επαρκής η έκδοση kubeflow-lite (αριθμός έκδοσης 1.4).

Αφού εγκαταστάθηκε το Kubeflow και όλα τα απαραίτητα υποσυστήματα χρειάζεται να γίνουν μόνο μερικές ρυθμίσεις:

- Πρώτον πρέπει να δημιουργηθεί ένας ρόλος για το istio-ingressgateway το οποίο φιλοξενεί το γραφικό περιβάλλον διαχείρισης του Kubeflow (dashboard). Ο ρόλος αυτός δίνει την δυνατότητα στο istio-ingressgateway να διαχειρίζεται τις διάφορες οντότητες του Kubernetes.

- Στη συνέχεια ορίζονται οι δημόσιες IP τις οποίες βλέπουν το dex-auth [35] και το oidc-gatekeeper [36].

- Τέλος ορίζεται το όνομα χρήστη και ο κωδικός χρήστη που χρησιμεύουν κατά την σύνδεση στο Kubeflow dashboard.

4.3 Hadoop Distributed File System

Αφού ετοιμάστηκε η συστοιχία καθώς και το Kubeflow πρέπει πλέον να υλοποιηθεί το κατανεμημένο αποθηκευτικό σύστημα HDFS. Για την εγκατάσταση του HDFS υπήρχαν δύο επιλογές: να εγκατασταθεί το HDFS εκτός της συστοιχίας του Kubernetes ή εντός σε περιβάλλον container. Ωστόσο δεν κρίθηκε χρήσιμο στο παρόν σύστημα η δεύτερη εκδοχή και έτσι προχωρήσαμε με την πρώτη. Η διαδικασία πρέπει να γίνει και στους 5 κόμβους και είναι η εξής:

- Εγκατάσταση του Java Development Kit 8 (openjdk-8-jdk) στους κόμβους.
- Λήψη του hadoop Για την παρούσα διπλωματική χρησιμοποιήθηκε η έκδοση 3.3.4.
- Ρύθμιση των μεταβλητών περιβάλλοντος του συστήματος.
- Καθώς το Spark όπως θα δούμε και αργότερα τρέχει κόμβους-εργάτες σε περιβάλλον container χρειάζεται επίσης να ρυθμίσουμε το DNS Service της συστοιχίας Kubernetes. Συγκεκριμένα πρέπει να προσθέσουμε εγγραφές που αντιστοιχίζουν τα ονόματα (hostname) με τις διευθύνσεις IP των 5 κόμβων καθώς το Hadoop λειτουργεί με τα hostnames τα οποία σε περίπτωση που δεν γίνει η διαδικασία είναι αδύνατο να επιλυθούν.
- Προσθήκη όλων των hostnames στο κατάλληλο αρχείο (slaves) που διαβάζει το Hadoop όταν γίνει η εκκίνηση των κόμβων δεδομένων.
- Ρύθμιση των ματβλητών περιβάλλοντος του Hadoop στο αρμόδιο αρχείο (hadoop-env.sh)

Η εγκατάσταση είναι σχεδόν έτοιμη. Το μόνο που μένει να γίνει είναι η ρύθμιση δύο πολύ βασικών αρχείων για το HDFS: το core-site.xml και το hdfs-site.xml. Οι ρυθμίσεις που έγιναν αφορούσαν τα εξής:

- Ρύθμιση του προκαθορισμένου μονοπατιού (fs.default.name) στο core-site.xml.
- Ρύθμιση του παράγοντα αναπαραγωγής (dfs.replication) που καθορίζει τον αριθμό των αντιγράφων ενός αρχείου.
- Ρύθμιση των μονοπατιών (dfs.namenode.name.dir, dfs.datanode.data.dir) που αποθηκεύονται τα δεδομένα και τα μεταδεδομένα στη περίπτωση του κόμβου ονομάτων.
- Ρύθμιση του μεγέθους του μεγέθους των block (dfs.blocksize).
- Ενεργοποίηση της διεπαφής WebHDFS [37] (dfs.webhdfs.enabled).
- Ενεργοποίση της δυνατότητας append σε αρχεία (dfs.support.append).
- Απενεργοποίηση της ασφάλειας. Αυτό το βήμα ουσιαστικά επιτρέπει σε οποιαδήποτε διεπαφή-πελάτη να κάνει αλλαγές στα αρχεία που βρίσκονται εντός του HDFS. Η προκαθορισμένη ρύθμιση είναι να υπάρχει ασφάλεια. Αυτό έγινε καθαρά για λόγους

διευκόλυνσης καθώς διαφορετικά θα έπρεπε να ρυθμιστούν ανάλογα οι διαφορετικοί χρήστες και ομάδες του HDFS. Κρίθηκε ωστόσο ότι αυτό είναι εκτός των στόχων της διπλωματικής. Σε περιπτώσεις παραγωγικών περιβαλλόντων αυτή η ρύθμιση που έγινε αποθαρρύνεται.

Για τις ρυθμίσεις και τη διαδικασία της εγκατάστασης ακολουθήθηκαν οι οδηγίες που είναι διαθέσιμες στην επίσημη ιστοσελίδα του Apache Hadoop [38].

Το HDFS είναι έτοιμο. Πριν ξεκινήσει η λειτουργία του είναι απαραίτητη η διαδικασία του format στο κατανεμημένο σύστημα αρχείων. Πλέον είναι δυνατή και η σύνδεση στο γραφικό περιβάλλον, η χρήση του εργαλείου γραμμής εντολών και η χρήση του REST API WebHDFS.

4.4 Spark Operator for Kubernetes

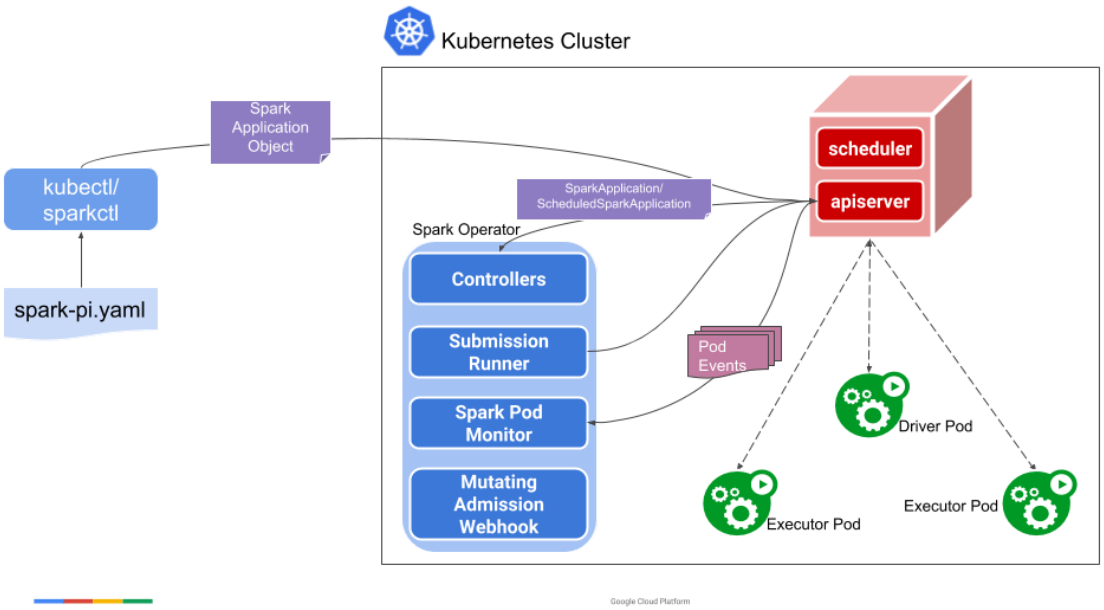
Το μόνο που μένει πλέον για την επίτευξη του στόχου ήτοι την χρήση του Spark μέσω διοχετεύσεων Kubeflow, είναι να εγκατασταθεί ο Spark στην συστοιχία. Εδώ πάλι είχαμε δύο επιλογές: η ¨παραδοσιακή¨ επιλογή θα ήταν να εγκατασταθεί τον Spark εκτός της συστοιχίας (δηλ. εκτός του περιβάλλοντος του Kubernetes) και να θέτονται φόρτοι εργασίας ρυθμίζοντας τον Spark να χρησιμοποιεί ως master την διεύθυνση του kube-api το οποίο στη συνέχεια θα δημιουργούσε container εργάτες στη συστοιχία. Η εναλλακτική που ακολουθήθηκε εν τέλει και που πλέον θεωρείται πιο εύρωστος τρόπος να κάνει κανείς χρήση του Spark στα πλαίσια ενός υπολογιστικού νέφους και ειδικά όταν πρόκειται για δημόσια νέφη που συχνά δεν είναι δυνατή η πρόσβαση στα ίδια τα μηχανήματα, είναι η χρήση ενός διαχειριστή (δηλ. operator) που απευθύνεται σε συστοιχίες Kubernetes.

Ο διαχειριστής είναι ένα εργαλείο που έχει αναπτύξει μία ομάδα εντός της Google που αποσκοπεί στην ένταξη του περιβάλλοντος του Spark στο οικοσύστημα του Kubernetes. Όπως συστήνεται για την εγκατάσταση του διαχειριστή στην συστοιχία, έγινε χρήση του εργαλείου helm3 [39] και των οδηγιών στο σχετικό αποθετήριο [31] που διατηρείται από την Google Cloud Platform.

4.4.1 Αρχιτεκτονική του Διαχειριστή Spark

Ο διαχειριστής [40] αποτελείται από:

- έναν ελεγκτή SparkApplication που παρακολουθεί για συμβάντα δημιουργίας, ενημέρωσης και διαγραφής αντικειμένων SparkApplication και δρα βάσει συμβάντων,
- έναν δρομέα υποβολής που τρέχει spark-submit για υποβολές που δέχεται από τον ελεγκτή,
- έναν παρατηρητή καψουλών Spark που παρακολοθεί τις κάψουλες Spark και στέλνει ενημερώσεις για την κατάστασή τους στον ελεγκτή SparkApplication,
- ένα Mutating Admission Webhook [41] που χειρίζεται προσαρμογές για τις κάψουλες οδηγούς και τις κάψουλες εργάτες Spark με βάση τις ρυθμίσεις (annotations) στις κάψουλες που προστέθηκαν από τον ελεγκτή,



Εικόνα 4.3: *Η αρχιτεκτονική του Spark on Kubernetes Operator.*

Συγκεκριμένα, ο χρήστης χρησιμοποιεί το sparkctl (ή kubectl) για να δημιουργήσει ένα αντικείμενο SparkApplication. Ο ελεγκτής SparkApplication λαμβάνει το αντικείμενο μέσω ενός παρατηρητή από τον διακομιστή API, δημιουργεί μια υποβολή που φέρει τα ορίσματα spark-submit και στέλνει την υποβολή στον δρομέα υποβολής. Ο δρομέας υποβολής υποβάλλει την εφαρμογή για εκτέλεση και δημιουργεί την κάψουλα οδηγό (driver) εφαρμογής. Κατά την εκκίνηση, η κάψουλα οδηγός δημιουργεί τις κάψουλες εργάτες. Ενώ η εφαρμογή εκτελείται, ο παρατηρητής Spark ελέγχει την κατάσταση των καψουλών της εφαρμογής και στέλνει ενημερώσεις κατάστασης των καψουλών πίσω στον ελεγκτή, ο οποίος στη συνέχεια ενημερώνει την κατάσταση της εφαρμογής ανάλογα.

Κεφάλαιο 5

Ανάπτυξη Διοχέτευσης Kubeflow

Στο παρόν κεφάλαιο περιγράφεται η πορεία της υλοποίησης της διοχέτευσης Kubeflow που αναπτύχθηκε για την παρουσίαση της λειτουργίας του Spark εντός του οικοσυστήματος του Kubeflow Pipelines.

5.1 Δήλωση SparkApplication

Για την επίτευξη της διαχείρισης ενός φόρτου εργασίας Spark μέσω του Kubeflow Pipelines πρέπει πρώτα να δούμε πώς επιτυγχάνεται η δήλωση εργασιών στο Spark Operator for Kubernetes. Η δήλωση γίνεται με την μορφή αντικειμένων SparkApplication στο kube-api μέσω του kubectl. Αυτά τα αντικείμενα είναι σε μορφή αρχείων YAML όπως το παρακάτω:

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: pyspark-pi
  namespace: default
spec:
  type: Python
  pythonVersion: "3"
  mode: cluster
  image: "gcr.io/spark-operator/spark-py:v3.1.1"
  imagePullPolicy: Always
  mainApplicationFile: local:///opt/spark/examples/src/main/python/pi.py
  sparkVersion: "3.1.1"
  restartPolicy:
    type: OnFailure
    onFailureRetries: 3
    onFailureRetryInterval: 10
    onSubmissionFailureRetries: 5
    onSubmissionFailureRetryInterval: 20
  driver:
    cores: 1
    coreLimit: "1200m"
```

```
    memory: "512m"
    labels:
      version: 3.1.1
    serviceAccount: spark
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.1.1
```

Εδώ σκιαγραφείται ένα απλό παράδειγμα δήλωσης SparkApplication που προέρχεται από το αποθετήριο του Spark Operator for Kubernetes. Εν συντομία θα αναλυθούν κάποιες από τις βασικές ρυθμίσεις:

- spec.type: Εδώ ορίζεται η γλώσσα προγραμματισμού στην οποία έχει συνταχθεί το βασικό αρχείο προς εκτέλεση.
- spec.image: Η εικόνα Δοσκερ η οποία περιέχει μία συμβατή έκδοση του Spark Operator for Kubernetes και η οποία θα τρέχει στα containers που θα τρέχουν τον κώδικα.
- spec.mainApplicationFile: Το αρχείο που περιέχει τον πηγαίο κώδικα προς εκτέλεση. Όπως φαίνεται από το πρόθεμα "local:/" το αρχείο αυτό μπορεί να βρίσκεται εντός της εικόνας που φορτώνεται ή σε κάποιο αποθηκευτικό χώρο κάποιου νέφους εναλλακτικά μπορεί να συνδεθεί κάποιο Volume [42].
- spec.driver: Εδώ υπάρχουν όλες οι ρυθμίσεις που αφορούν στον οδηγό του Spark.
- spec.executor: Ομοίως με το παραπάνω πεδίο. Η μόνο σημαντική διαφορά είναι η δυνατότητα δήλωσης instances δηλαδή πλήθος εργατών για τον φόρτο εργασίας. Τα πεδία cores και memory αφορούν σε πυρήνες και μνήμη ανά εργάτη (δηλ. ανά instance).

Αυτού του είδους αρχεία μπορούν να δηλωθούν στη συστοιχία μέσω της σχετικής εντολής (apply, create).

5.2 Δήλωση Διοχέτευσης

Η διοχέτευσή που αναπτύχθηκε αποτελείται από απλά components που προϋπάρχουν στην βασική έκδοση του KFP. Δομικά η διοχέτευση περιέχει δύο βασικούς κόμβους εκτέλεσης εργασιών Spark, κόμβους ελέγχου κατάστασης και βοηθητικούς κόμβους.

Οι βασικοί κόμβοι είναι αυτοί που εκκινούν τις συστοιχίες Spark. Πρόκειται για διεργασίες που διαβάζουν το αρχείο δήλωσης ενός SparkApplication και στη συνέχεια χρησιμοποιούν το API του KFP για να δημιουργήσουν νέα αντικείμενα στην συστοιχία. Στο τέλος γυρνάνε το όνομα του SparkApplication που δημιουργήθηκε καθώς αυτό είναι δυναμικό.

Περισσότερο ενδιαφέρον έχουν οι κόμβοι ελέγχου κατάστασης. Αυτοί οι κόμβοι στην ουσία επιτρέπουν την διαχείριση της πορείας της ροής της διοχέτευσης. Πρόκειται για δύο

κόμβους που αναπτύχθηκαν με σκοπό την παρακολούθηση της εξέλιξης της λειτουργίας του SparkApplication. Αυτό είναι αναγκαίο στην περίπτωση που ένα μεταγενέστερο στάδιο της διοχέτευσης πρέπει να περιμένει την ολοκλήρωση κάποιας ροής εργασίας Spark προτού εκκινηθεί καθώς ενδεχομένως χρησιμοποιεί τα αποτελέσματα αυτής ως είσοδό του. Επιπλέον λόγω του ότι οι βασικοί κόμβοι δεν παρέχουν από μόνοι τους διαδραστικότητα με το περιβάλλον του Spark, δεν γνωστοποιείται η κατάσταση της εκτέλεσης εγγενώς στο περιβάλλον του KFP. Για τους σκοπούς της διπλωματικής εργασίας αναπτύχθηκαν δύο λειτουργίες container που ελέγχουν την κατάσταση της εκτέλεσης κάνοντας κλήσεις στο kube-api μέσω του εργαλείου kubectl. Αυτές οι λειτουργίες βασίζονται στην εικόνα kubectl με έκδοση 1.21.12-debian-10-r32 της bitnami και λειτουργούν με την μέθοδο polling δηλαδή κάνοντας ένα αίτημα στον διακοσμητή ανά τακτά χρονικά διαστήματα.

Τέλος οι βοηθητικοί κόμβοι είναι απλοί κόμβοι που ελέγχουν το αποτέλεσμα του ελέγχου κατάστασης. Ουσιαστικά ο έλεγχος κατάστασης κάνει αιτήματα στο kube-api όσο η κατάσταση του SparkApplication είναι "RUNNING". Στην πράξη το SparkApplication μπορεί να έχει μία από τις καταστάσεις "RUNNING", "FAILED" ή "COMPLETED". Αν η κατάσταση γίνει "FAILED" βγαίνει ένα απλό μήνυμα σφάλματος ενώ αν η κατάσταση γίνει "COMPLETED" ο βοηθητικός κόμβος επιτρέπει στην διοχέτευση να συνεχίσει στους παρακάτω κόμβους.

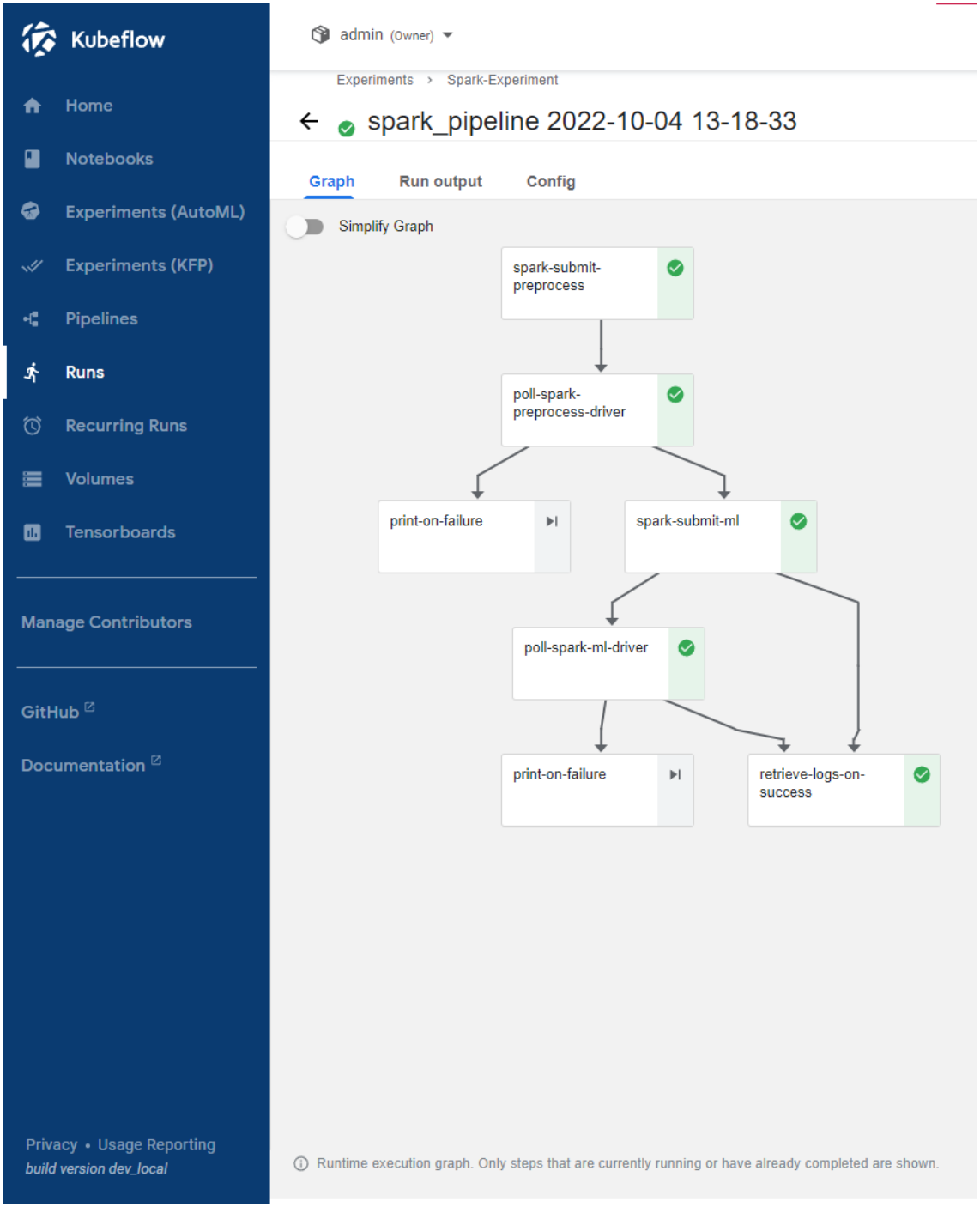
5.3 Μοντέλο Μηχανικής Μάθησης

Αφού είδαμε πώς είναι η δομή της διοχέτευσης από μία αφαιρετική σκοπιά αξίζει να δούμε την διαδικασία που τρέχει ο Apache Spark στο παρασκήνιο. Πρόκειται για ένα απλό σχετικά μοντέλο, σκοπός του οποίου είναι ο έμπρακτος έλεγχος του μοτίβου προγράμματος που περιγράφηκε στην προηγούμενη ενότητα. Αξίζει να σημειωθεί ότι το περιβάλλον του KFP δεν περιορίζει τον χρήστη ως προς το πλήθος κόμβων επεξεργασίας που θα χρησιμοποιήσει. Για λόγους επίδειξης κρίθηκε επαρκές να αναπτυχθεί μία διοχέτευση ήτοι ένα σύστημα μηχανικής μάθησης που χωρίζεται σε δύο κομμάτια με μία σχέση εξάρτησης μεταξύ τους.

5.3.1 Δεδομένα

Όπως προαναφέρθηκε τα δεδομένα προέρχονται από το γνωστό αποθετήριο UCI Machine Learning Repository και σκοπός του συνόλου δεδομένων είναι η ταξινόμηση των δειγμάτων σε δύο κατηγορίες ανάλογα το εισόδημα με ένα κατώφλι. Τα δείγματα αποτελούνται από τα εξής χαρακτηριστικά:

- Ηλικία: συνεχής
- Εργατική τάξη: διακριτή
- Ειδικό βάρος της εγγραφής [43]: συνεχής
- Εκπαίδευση: διακριτή
- Χρόνια εκπαίδευσης: συνεχής
- Οικογενειακή κατάσταση: διακριτή: διακριτή



Εικόνα 5.1: *Επιτυχής εκτέλεση της διοχέτευσης που αναπτύχθηκε*

- Επάγγελμα: διακριτή
- Φυλή: διακριτή
- Φύλο: συνεχής
- Κεφαλαιουχικό κέρδος: συνεχής
- Απώλεια κεφαλαίου: συνεχής
- Ώρες ανά εβδομάδα: συνεχής
- Χώρα ιθαγένειας: διακριτή
- Εισόδημα: διακριτή

5.3.2 Μεθοδολογία

Η διαδικασία της υλοποίησης χωρίζεται σε δύο κομμάτια: το κομμάτι της προπαρασκευής (preprocessing) και της εκπαίδευσης (training). Η προπαρασκευή αποτελείται από τα εξής βήματα:

- Διάβασμα δεδομένων από το HDFS και δημιουργία 2 Spark dataframe ένα για το σύνολο εκπαίδευσης και ένα για το σύνολο ελέγχου.
- Διαχωρισμός των διακριτών μεταβλητών.
- Δημιουργία μίας βοηθητικής μεταβλητής StringIndexer για κάθε διακριτή μεταβλητή.
- Κωδικοποίηση των StringIndexer σε μορφή one-hot.
- Αποθήκευση των Spark dataframe στον φάκελο tmp του HDFS.

Στη συνέχεια η εκπαίδευση αποτελείται από τα εξής βηματα:

- Διάβασμα των προεπεξεργασμένων δεδομένων από το HDFS και δημιουργία 2 Spark dataframe ένα για το σύνολο εκπαίδευσης και ένα για το σύνολο ελέγχου.
- Ενοποίηση όλων των κατηγορικών μεταβλητών με κωδικοποίηση one-hot σε ένα διάνυσμα με τη χρήση του VectorAssembler.
- Ενοποίηση των αναπαραστάσεων one-hot με τις συνεχείς μεταβλητές.
- Κωδικοποίηση της μεταβλητής στόχου σε δυαδική αναπαράσταση.
- Εκπαίδευση μοντέλου λογιστικής παλινδρόμησης.
- Έλεγχος ακρίβειας στο σύνολο ελέγχου.

5.3.3 Λεπτομέρειες Σχεδίασης

Το πρόγραμμα είναι γραμμένο σε Python και τρέχει στην έκδοση 3.9. Για την εκτέλεση του αλγορίθμου μηχανικής μάθησης και την προεπεξεργασία επιλέχθηκε το πλήρως κατανεμημένο framework SparkML [44]. Λόγω του ότι πρόκειται για εφαρμογή που τρέχει εντός container χρειάζεται η δημιουργία μίας κατάλληλης εικόνας. Η βασική εικόνα στην οποία βασίστηκε η τελική είναι αυτή του Spark Operator for Kubernetes (spark-operator/spark-py με έκδοση v3.1.1) και για τους σκοπούς της διπλωματικής δημιουργήθηκε μία εικόνα στην οποία προστέθηκε η βιβλιοθήκη numpy καθώς είναι απαραίτητη για την λειτουργία του Spark MLlib. Η τελική εικόνα που χρησιμοποιήθηκε μπορεί να βρεθεί στο DockerHub με το όνομα arisspyrou/spark-py. Ο κώδικας της Python που διαβάζεται από το KFP είναι τοποθετημένος στο HDFS.

Κεφάλαιο 6

Αποτίμηση Συστοιχίας Apache Spark

Σε αυτό το κεφάλαιο θα περιγραφεί η διαδικασία που ακολουθήθηκε για την αποτίμηση της λειτουργίας του Apache Spark εντός της συστοιχίας Kubernetes.

6.1 Benchmark

Επόμενο μέλημα στην διπλωματική είναι η αξιολόγηση των δυνατοτήτων του Spark εντός της συστοιχίας που έχουμε στη διάθεσή μας. Για τον σκοπό αυτό επιλέχθηκε η εκτέλεση ενός γνωστού στη βιομηχανία benchmark, το TPC-DS [45].

6.1.1 TPC-DS

Το TPC-DS είναι το de-facto σημείο αναφοράς στη βιομηχανία για τη μέτρηση της απόδοσης των λύσεων υποστήριξης αποφάσεων, συμπεριλαμβανομένων, των συστημάτων μεγάλων δεδομένων. Ο υποκείμενος στόχος του TPC-DS είναι προμηθευτές προϊόντων λιανικής, παρόλα αυτά το σχήμα της βάσης δεδομένων, ο πληθυσμός δεδομένων, τα ερωτήματα, το μοντέλο συντήρησης δεδομένων και οι κανόνες εφαρμογής έχουν σχεδιαστεί για να είναι αντιπροσωπευτικά των σύγχρονων συστημάτων υποστήριξης αποφάσεων. Το benchmark απεικονίζει συστήματα που έχουν τα εξής χαρακτηριστικά:

- Απευθύνονται σε μεγάλους όγκους δεδομένων.
- Δίνουν απαντήσεις σε ερωτήματα στον πραγματικό κόσμο των επιχειρήσεων.
- Εκτελούν ερωτήματα διαφόρων λειτουργικών απαιτήσεων και πολυπλοκοτήτων (π.χ. ad-hoc, αναφοράς, επαναληπτικά OLAP, εξόρυξη δεδομένων).
- Χαρακτηρίζονται από υψηλό φόρτο CPU και I/O.
- Συγχρονίζονται περιοδικά με πηγαίες βάσεις δεδομένων OLTP μέσω λειτουργιών συντήρησης.
- Τρέχουν σε συστήματα μεγάλων δεδομένων όπως RDBMS και Hadoop/Spark.

6.1.2 Πληροφορίες Υλοποίησης

Στην πραγματικότητα το TPC-DS απευθύνεται σε συστήματα μεγαλύτερης κλίμακας ικανά να διαχειριστούν όγκους της τάξεως των δεκάδων και των εκατοντάδων terabyte που έχουν

στη διάθεσή τους πολλούς πόρους. Η κλίμακα των πειραμάτων της παρούσας διπλωματικής εργασίας είναι πολύ μικρότερη και απευθύνεται σε προσομοίωση συνόλων δεδομένων δεκάδων gigabyte. Επιπλέον το αρχικό benchmark αφορά σε 99 διαφορετικά ερωτήματα που εξετάζουν όλες τις πτυχές ενός συστήματος αναλυτικά και εκτελούνται πολλαπλές φορές για να εξαχθεί μία όσο γίνεται πιο αντικειμενική αξιολόγηση των δυνατοτήτων ενός συστήματος. Λόγω των περιορισμένων πόρων που έχουμε στην διάθεσή μας επιλέξαμε να εκτελέσουμε 3 ερωτήματα που αφορούν σε διαφορετικού είδους φόρτων εργασίας. Συγκεκριμένα επιλέχθηκαν τα ερωτήματα q64, q70 και q82 που είναι απαιτητικά σε network shuffle, CPU και I/O αντίστοιχα.

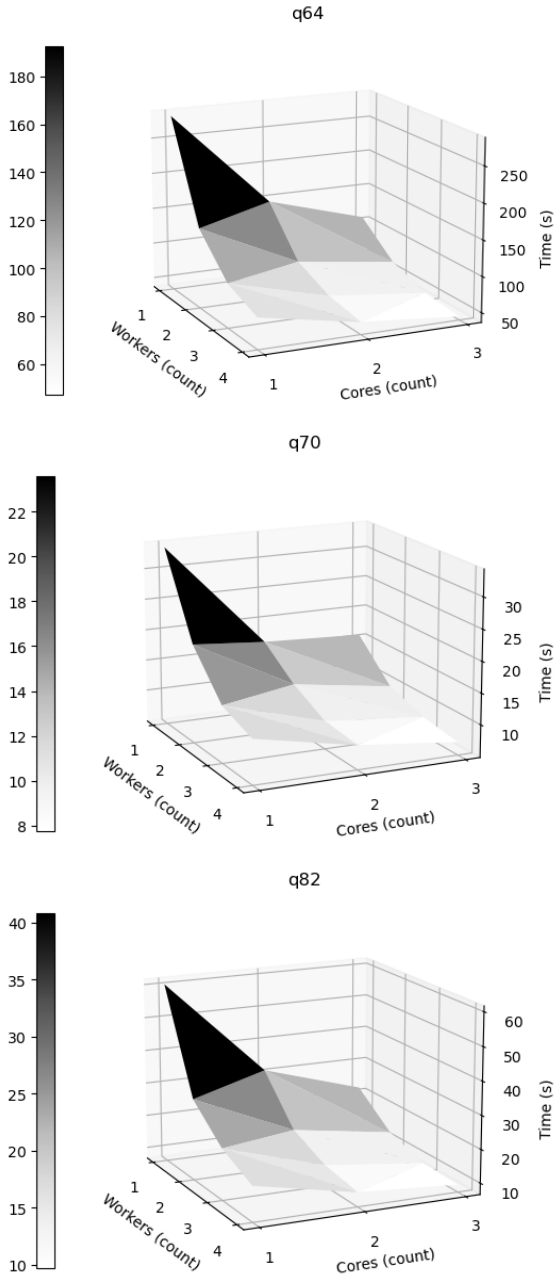
Η εκτέλεση των ερωτημάτων έγινε με σχετικό αρχείο YAML manifest που προέρχεται από την υπηρεσία Amazon Web Services όπως και η εικόνα του container που περιέχει το benchmark εγκατεστημένο. Η διαδικασία γίνεται σε δύο μέρη: πρώτα γίνεται δήλωση ενός φόρτου εργασίας που παράγει τα συνθετικά δεδομένα και αφού ολοκληρωθεί αυτή η διαδικασία τότε είναι δυνατή η δήλωση φόρτων που τρέχουν ερωτήματα με διάφορες παραμετροποίησεις. Το συνθετικό σύνολο δεδομένων αποτελείται από 5 partitions με παράγοντα κλίμακας 10 και τύπο αρχείου parquet. Τα δεδομένα μετά την παραγωγή τους αποθηκεύονται στο HDFS. Για τα πειράματα έχει κρατηθεί ο οδηγός του Spark σταθερός ως προς την παραμετροποίησή του και συγκεκριμένα του εκχωρούνται 2 vCPU με όριο έως 3,3, και 3 GB RAM. Όσον αφορά τους κόμβους εργάτες έχουν εκτελεστεί 36 πειράματα από 3 επαναλήψεις το καθένα για μεγαλύτερη ακρίβεια με συνδυασμούς των αριθμών vCPU, RAM και instances με τους εξής περιορισμούς:

- Η RAM μπορεί να είναι 1, 2 ή 3 GB καθώς τα μηνανήματα έχουν σύνολο 4 και μεγαλύτερες τιμές μπορούν να προκαλέσουν σφάλματα.
- Οι vCPU μπορούν να είναι 1, 2 ή 3 καθώς κάθε μηχάνημα έχει 4 και δεσμεύεται μία για την εύρυθμη λειτουργία του συστήματος.
- Τα instances μπορούν να είναι από 1 έως 4. Θεωρητικά θα μπορούσαμε να τρέξουμε παραπάνω instances με λιγότερους πόρους στο καθένα ωστόσο δεν επιχειρήθηκε και είναι ένας από τους περιορισμούς της διπλωματικής εργασίας.

Μετά το πέρας του κάθε πειράματος μία αναλυτική αναφορά αποθηκεύεται στο HDFS. Ο αναλυτικός πίνακας με τις διαφορετικές παραμετροποιήσεις αλλά και τα αποτελέσματα του κάθε πειράματος βρίσκεται στα παραρτήματα.

6.2 Ανάλυση Αποτελεσμάτων

Αφού ολοκληρώθηκαν όλα τα πειράματα, αποκτήθηκε σημαντική πληροφορία για το πώς επηρεάζεται ο χρόνος ροής ενός ερωτήματος από την παραμετροποίηση της συστοιχίας Apache Spark. Όπως προαναφέρθηκε μελετήθηκαν τρία ερωτήματα: ένα που είναι απαιτητικό σε network shuffle, ένα σε CPU και ένα σε I/O τα οποία θα αναφέρονται με τους κωδικούς τους q64, q70 και q82 αντίστοιχα. Από τα τρία ερωτήματα το πιο απαιτητικό συνολικά αποδείχθηκε το q64 έχοντας εύρος από 283 έως 41 δευτερόλεπτα με παραμετροποίηση 1 εργάτη, 1 πυρήνα και 1 GB RAM και 4 εργάτες, 3 πυρήνες (έκαστος) και 1 GB RAM.



Εικόνα 6.1: *Τρισδιάστατη γραφική αναπαράσταση των αποτελεσμάτων.*

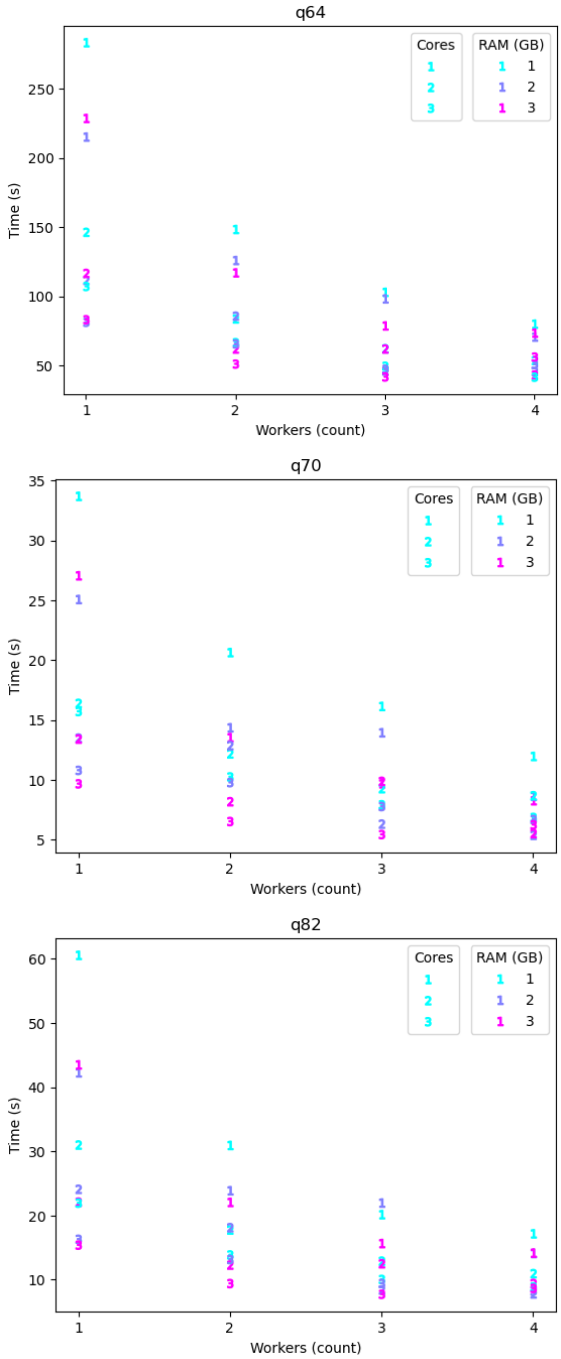
Στη συνέχεια οι παραμετροποιήσεις θα αναφέρονται με την μορφή X.Y.Z όπου X θα είναι οι εργάτες, Y θα είναι οι πυρήνες ανά εργάτη και Z τα gigabyte μνήμης ανά εργάτη και οι χρόνοι είναι πάντα σε δευτερόλεπτα. Αντιθέτως τα άλλα δύο ερωτήματα ήτοι τα q70, q82 έχουν εύρος 34 έως 6 και 61 έως 8. Όπως φαίνεται από την εικόνα 6.1 εν γένει όσο αυξάνονται οι εργάτες και οι πυρήνες τόσο μειώνεται ο χρόνος εκτέλεσης. Στην περίπτωση της μνήμης η εικόνα δεν είναι τόσο ξεκάθαρη. Βλέποντας τον πίνακα A.1 φαίνεται για παράδειγμα στο q64 και στις παραμετροποιήσεις 4.3.1, 4.3.2, 4.3.1 ότι ο χρόνος δεν μειώνεται όσο αυξάνουμε την μνήμη αλλά αυξάνεται. Αυτό δεν μπορεί να αποδοθεί κάπου παρά στο γεγονός ότι υπάρχουν αυξομειώσεις των χρόνων που βασίζονται σε εξωτερικούς παράγοντες (π.χ διεργασίες παρασκηνίου). Θεωρητικά λόγω του ότι το σύνολο δεδομένων είναι χωρισμένο σε 5 partitions και αποτελείται συνολικά από 10GB δεδομένων κάθε partition είναι γύρω στα 2GB. Αυτό επιβεβαιώνεται σε κάποιες περιπτώσεις όπως στο q64 και q82 με παραμετροποίηση 1,1,X αφού βλέπουμε ότι οι χρόνοι πέφτουν από 283 (1 GB RAM) στο 215 (2 GB RAM) και 228 (3 GB RAM) και από 60,5 (1 GB RAM) σε 42 (2 GB RAM) και 43 (3 GB RAM) αντίστοιχα. Άλλη μία ενδιαφέρουσα παρατήρηση είναι ότι δεν έχει μόνο σημασία πόσοι πυρήνες τρέχουν ένα ερώτημα αλλά σε πόσα μηχανήματα είναι κατανεμημένοι αυτοί οι πυρήνες. Για παράδειγμα βλέπουμε ότι οι χρόνοι είναι 70,5 (q64), 8,6 (q70), 14 (q82) για την παραμετροποίηση 4.1.2 και 85,7 (q64), 12,8 (q70) και 18,1 (q82) για την παραμετροποίηση 2.2.2. Η βελτίωση αυτή είναι περίπου 17,6%, 32,8% και 22,1% αντίστοιχα και δείχνει ότι είναι εν γένει καλύτερο να επιτυγχάνεται καλύτερη κατανομή των εργασιών παρά παραλληλοποίηση.

Η εικόνα 6.2 αποτελεί άλλη μία γραφική αναπαράσταση των αποτελεσμάτων του benchmark. Βλέπουμε ότι υπάρχει μία γενική τάση εκθετικής μείωσης του χρόνου εκτέλεσης όσο αυξάνονται οι εργάτες. Φαίνεται επίσης ότι όσο αυξάνονται οι εργάτες τόσο μειώνεται η διαφορά που έχουν οι χρόνοι των πειραμάτων με 1 πυρήνα σε σχέση με τα υπόλοιπα πειράματα της στήλης (ήτοι με ίδιο αριθμό εργατών). Τέλος μπορούμε να διακρίνουμε ότι υπάρχει το φαινόμενο των φθινουσών αποδόσεων όσο αφορά τον αριθμό των πυρήνων ανά εργάτη. Για παράδειγμα για 1 εργάτη βλέπουμε ότι η διαφορά μεταξύ 1 πυρήνα και 3 πυρήνων είναι περίπου 176 δευτερόλεπτα στην περίπτωση του 1 GB RAM. Αντιθέτως για το πείραμα 4.1.1 και 4.3.1 είναι 38,5 δευτερόλεπτα.

	Time q64	Time q70	Time q82
Workers	-0.636657	-0.622510	-0.632972
Cores	-0.549904	-0.540662	-0.551698
RAM	-0.132204	-0.251986	-0.190437

Πίνακας 6.1: *Πίνακας συσχετίσεων των τιμών των χρόνων με τους αριθμούς των εργατών, των πυρήνων και των GB RAM.*

Τέλος παρουσιάζουμε τον πίνακα 6.1 όπου έχουμε συγκεντρώσει τον συντελεστή συσχέτισης Pearson για τα ζεύγη μεταξύ εργατών, πυρήνων, RAM και χρόνου για κάθε ερώτημα. Όπως φαίνεται από τον πίνακα αλλά και όπως έχουμε ήδη προαναφέρει, φαίνεται ότι ο αριθμός των εργατών να είναι ο πιο σημαντικός παράγωντας στην μείωση του χρόνου του ερωτήματος. Σχεδόν εξίσου σημαντικό φαίνεται να είναι ο αριθμός των πυρήνων. Αξίζει να σημειωθεί ότι βλέπουμε χαμηλή διακύμανση στις τιμές στην πρώτη και δεύτερη σειρά του



Εικόνα 6.2: *Γραφική αναπαράσταση των αποτελεσμάτων. Τα ψηφία δείχνουν τον αριθμό των πυρήνων κάθε πειράματος ενώ το χρώμα του κάθε ψηφίου δείχνει τα GB της RAM.*

πίνακα. Στη συνέχεια φαίνεται ότι ο αριθμός των GB RAM δεν παίζει εξίσου μεγάλο ρόλο στην μείωση του χρόνου των ερωτημάτων. Ωστόσο το ενδιαφέρον στην τρίτη σειρά είναι ότι παρουσιάζεται μεγαλύτερη διακύμανση μεταξύ των ερωτημάτων.

Κεφάλαιο 7

Σύστημα Υποστήριξης Αποφάσεων

Τελικός στόχος της παρούσας διπλωματικής εργασίας είναι η εξαγωγή γνώσης που αφορά στην βελτιστοποίηση της παραμετροποίησης των συστοιχιών Spark λαμβάνοντας υπόψιν τον χρόνο αλλά και το κόστος ενός πειράματος.

Όπως έχουμε προαναφέρει οι χρόνοι των πειραμάτων προέρχονται από την εκτέλεση τριών διαφορετικών ερωτημάτων που το καθένα έχει διαφορετικά ποιοτικά χαρακτηριστικά. Συνολικά αυτά τα τρία αντιπροσωπεύουν τα διαφορετικά είδη φόρτων εργασίας που μπορούν να δηλωθούν στον Spark προς ανάλυση. Όπως φαίνεται στον πίνακα A.1 όσο αυξάνονται τα instances και οι πυρήνες τόσο ελαχιστοποιείται ο χρόνος εκτέλεσης. Ωστόσο ως γνωστόν όσο περισσότερα workers τρέχουν εντός του Spark τόσο μεγαλύτερο είναι το overhead για τις μικρές διαδικασίες που εκτελεί ο καθένας. Το γεγονός αυτό έχει ως αποτέλεσμα ότι ο χρόνος δεν μειώνεται ή αυξάνεται γραμμικά ενώ ταυτόχρονα εμφανίζει μεγάλο βαθμό μεταβλητότητας μεταξύ επαναλήψεων καθώς οι διεργασίες που τρέχουν στο παρασκήνιο αλλά και άλλες διαδικασίες συντήρησης που τρέχει το Kubernetes επηρεάζουν τον τελικό χρόνο ροής.

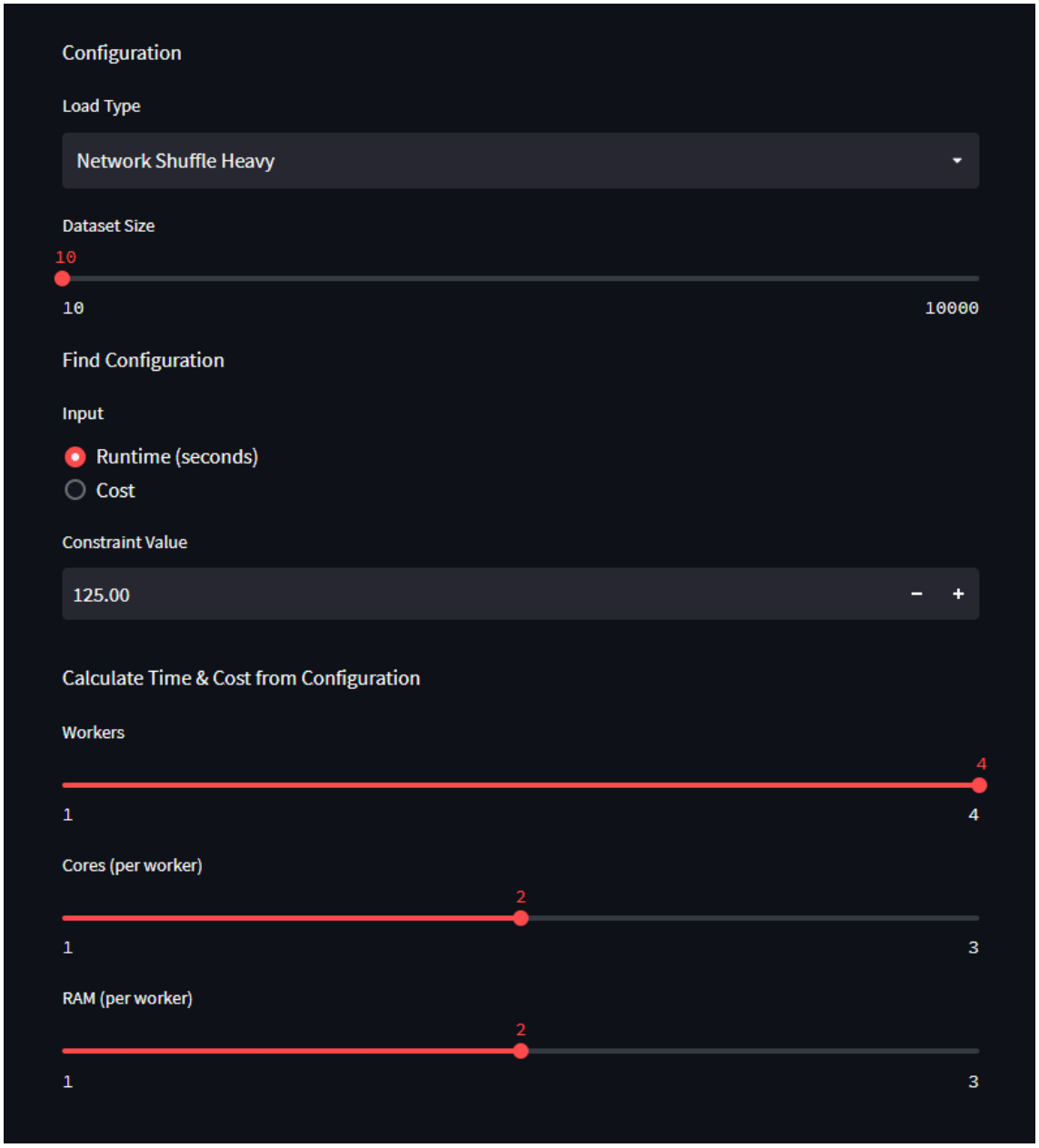
Παρόλη τη μεταβλητότητα και την μεγάλη διακύμανση που παρατηρήθηκε κατά τις επαναλήψεις των ερωτημάτων κρίθηκε σκόπιμο να υλοποιηθεί ένα σύστημα που παρέχει υποστήριξη στους χρήστες όταν επιλέγουν τις παραμετροποιήσεις για τους φόρτους εργασίας που εκτελούν. Για τον σκοπό αυτό αναπτύχθηκε ένα εργαλείο με την μορφή ενός web app που μέσα από το γραφικό του περιβάλλον μπορεί κανείς να πάρει μία πρώτη εικόνα για το πώς πρόκειται να τρέξει ένας εν δυνάμει φόρτος εργασίας.

Το εργαλείο που αναπτύχθηκε έχει δύο βασικές λειτουργίες. Πρώτον την εύρεση μίας πιθανής παραμετροποίησης του Spark με τις εξής επιλογές:

- Επιλογή είδους φόρτου εργασίας ανάμεσα σε 4 εναλλακτικές: Balanced, Network Shuffle Heavy, CPU Heavy και I/O Heavy.
- Επιλογή μεγέθους συνόλου δεδομένων με 4 εναλλακτικές: 10, 100, 1000, 10000.
- Προσδιορισμός περιορισμού είτε σε χρόνο (δευτερόλεπτα) είτε σε κόστος.

Σε αυτή τη λειτουργία το σύστημα που αναπτύχθηκε εκπαιδεύει δύο δέντρα αποφάσεων σε κατάλληλο υποσύνολο των δεδομένων που προέρχονται από τα πειράματα και κάνει μία εκτίμηση για την καλύτερη δυνατή παραμετροποίηση της συστοιχίας Spark λαμβάνοντας υπόψιν του τον περιορισμό που έχει οριστεί από τον χρήστη.

Η δεύτερη λειτουργία του συστήματος είναι η εκτίμηση του χρόνου και του κόστους δεδομένης μίας παραμετροποίησης προκειμένου να τρέξει ένας φόρτος εργασίας με τις επιλογές που προαναφέρθηκαν. Σε αυτή τη λειτουργία ο περιορισμός που έχει ορισθεί δεν συνυπολογίζεται καθώς σκοπός της είναι η προσομοίωση του χρόνου και του κόστους δεδομένης μίας παραμετροποίησης. Όπως και στην προηγούμενη λειτουργία έτσι και εδώ εκπαιδεύεται ένα δέντρο αποφάσεων πάνω σε ένα υποσύνολο των δεδομένων.

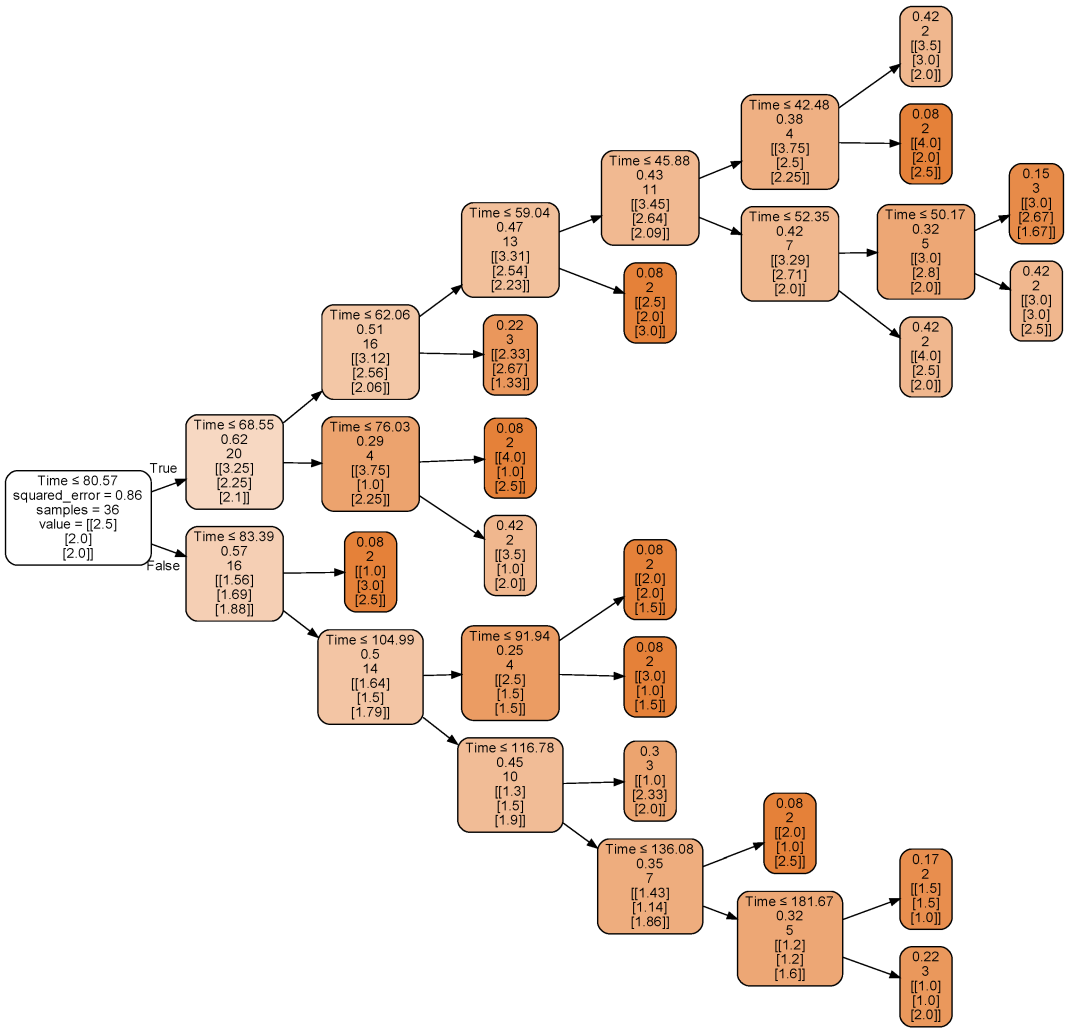


Εικόνα 7.1: *Η διεπαφή χρήστη του συστήματος υποστήριξης αποφάσεων που αναπτύχθηκε.*

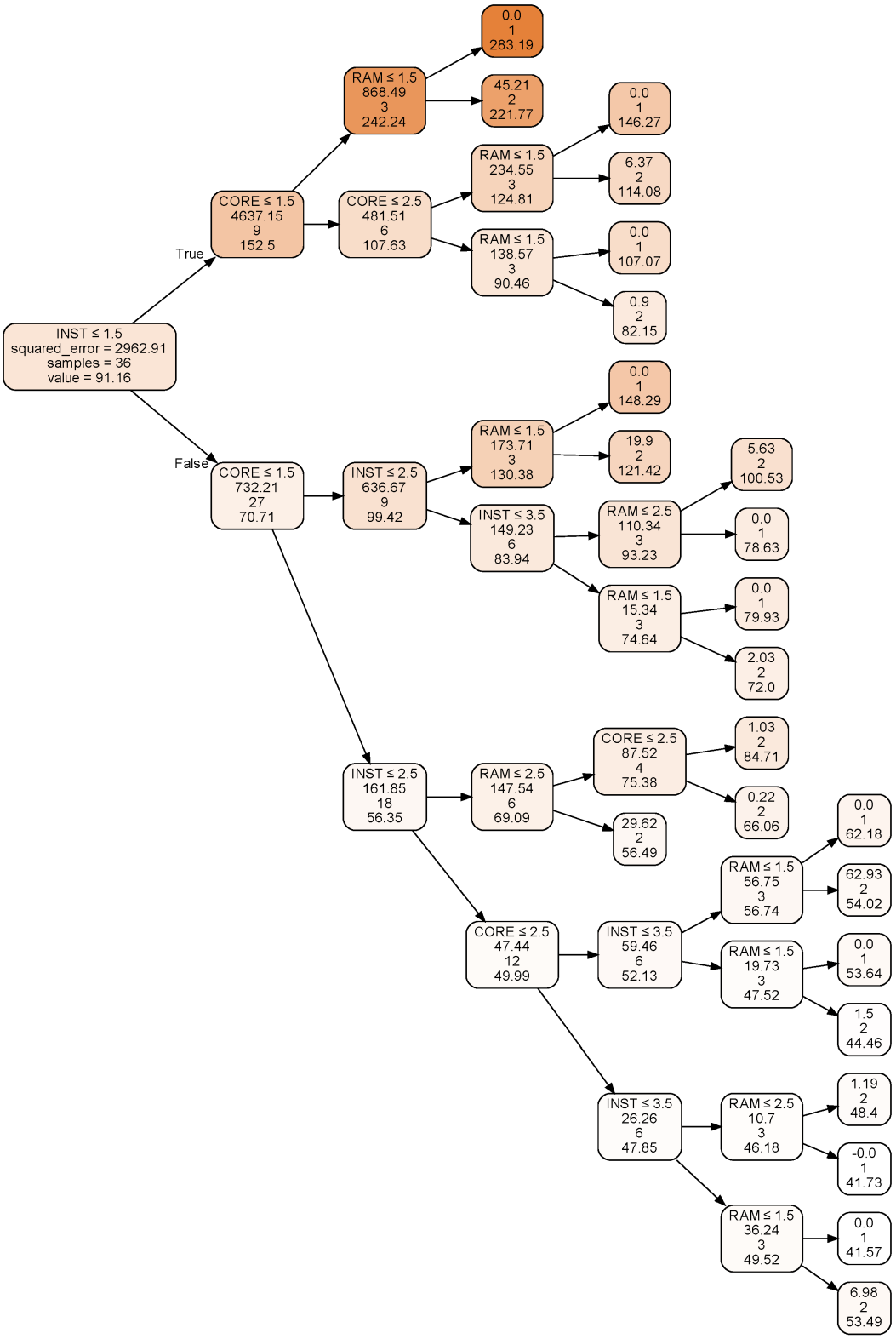
7.1 Δέντρα Αποφάσεων

Όπως προαναφέρθηκε το σύστημα υποστήριξης αποφάσεων βασίστηκε στην εκπαίδευση δέντρων αποφάσεων. Η διαδικασία εκπαίδευσης έχει να κάνει άμεσα με τις επιλογές του

χρήστη στην διεπαφή που αναπτύχθηκε. Για παράδειγμα αν αυτός/η επιλέξει είδος φόρτου network shuffle heavy τότε τα δέντρα που θα εκπαιδευτούν θα αφορούν μόνο στα δεδομένα που έχουν προκύψει για το ερώτημα q64. Αυτό γίνεται για να αυξηθεί όσο γίνεται περισσότερο η ακρίβεια του μοντέλου καθώς γνωρίζουμε εκ των προτέρων το είδος του φόρτου που θέλουμε να προβλέψουμε. Μία περαιτέρω βελτίωση που έγινε είναι ο περιορισμός των φύλλων του δέντρου για να επιτευχθεί καλύτερη γενίκευση αλλά και να εξαλειφθεί η επιρροή των έκτοπων τιμών. Αυτό έγινε αυξάνοντας τον ελάχιστο αριθμό δεδομένων ανά φύλλο σε 2. Πρακτικά αυτό σημαίνει ότι αν θέλουμε να προχωρήσουμε το δέντρο από ένα φύλλο τότε πρέπει μετά τον χωρισμό τα νέα φύλλα να έχουν τουλάχιστον 2 δεδομένα. Αυτή η πρακτική είναι αρκετά συνήθης στην παλινδρόμησή. Ακολουθούν τα σχήματα 7.2 και 7.3 στα οποία παρουσιάζονται δύο δέντρα αποφάσεων. Και τα δύο δέντρα αφορούν σε network shuffle heavy φόρτους εργασίας σε σύνολα δεδομένων μεγέθους 10GB. Το πρώτο δέντρο χρησιμοποιείται για την πρόβλεψη παραμετροποίησης με είσοδο τον χρόνο ροής ενώ το δεύτερο κάνει την αντίστροφη λειτουργία δηλαδή προβλέπει τον χρόνο με βάση μία παραμετροποίηση.



Εικόνα 7.2: *Δέντρο αποφάσεων που εκπαιδεύτηκε για network shuffle heavy φόρτο εργασίας σε σύνολο δεδομένων 10GB. Κάνει πρόβλεψη της παραμετροποίησης παίρνοντας ως είσοδο τον χρόνο ροής.*



Εικόνα 7.3: *Δέντρο αποφάσεων που εκπαιδεύτηκε για network shuffle heavy φόρτο εργασίας σε σύνολο δεδομένων 10GB. Κάνει πρόβλεψη του χρόνου ροής παίρνοντας ως είσοδο μία παραμετροποίηση.*

Κεφάλαιο 8

Συμπεράσματα

8.1 Συνεισφορά

Στα πλαίσια της παρούσας διπλωματικής αναπτύχθηκαν τα εξής:

- Αναπτύχθηκε διοχέτευση KFP ικανή να διαχειρίζεται πλήρως μία ροή εργασίας Apache Spark και δύναται να χρησιμοποιήσει αποτελέσματα αυτής σε μεταγενέστερα στάδια της διοχέτευσης.
- Ως εκ τούτου επιδείχθηκε η δυνατότητα πλήρης διαλειτουργικότητας μεταξύ των οικοσυστημάτων του Kubeflow και συστοιχιών Apache Spark.
- Αποτιμήθηκε η λειτουργία του Apache Spark με απαιτητικούς φόρτους εργασίας που προέρχονται από το TPC-DS benchmark.
- Αναπτύχθηκε σύστημα υποστήριξης αποφάσεων βασισμένο στα πειράματα που διεξήχθησαν ικανό να κατευθύνει έναν νέο χρήστη που θέλει να τρέξει φόρτους εργασίας Spark.

8.2 Περιορισμοί & Μελλοντικό Έργο

Το TPC-DS benchmark σαν φόρτος εργασίας δεν αντικατοπτρίζει πλήρως τους φόρτους εργασίας μηχανικής μάθησης. Ως εκ τούτου τα αποτελέσματα των πειραμάτων είναι ένα σημείο αναφοράς αλλά ενδεχομένως να χρειάζονται περαιτέρω δοκιμές για να δημιουργηθεί μία πιο ολοκληρωμένη βάση γνώσης από την οποία να μπορεί να αντλήσει το σύστημα υποστήριξης αποφάσεων. Επιπλέον, λόγω περιορισμών δεν έχουμε κάνει πειράματα σε συνθετικά σύνολα δεδομένων μεγαλύτερα των 10 GB και οι τιμές που δίνει το σύστημά μας είναι μόνο αφελείς προβολές σε μεγαλύτερα σύνολα. Οι δοκιμές σε μεγαλύτερα σύνολα θα χρειάζονταν περισσότερους πόρους τόσο αποθηκευτικούς όσο και υπολογιστικούς. Τα πραγματικά αποτελέσματα σε μεγαλύτερα σύνολα μπορεί να διαφέρουν σημαντικά. Στη συνέχεια για τους υπολογισμούς των χρόνων χρησιμοποιούνται μόνο στατικά ιστορικά δεδομένα από το benchmark. Ένας άλλος τρόπος θα ήταν η εύρεση παρόμοιων φόρτων εργασίας μέσω χαρακτηριστικών που εξάγονται είτε κατευθείαν από τον κώδικα (π.χ. πλήθος map, reduce) είτε από προηγούμενες εκτελέσεις του ίδιου φόρτου ή άλλων παρόμοιων. Στην πράξη είναι δύσκολη η a priori εκτίμηση του πόσο βαρύ είναι μία ροή Spark πριν τρέξει σε ένα πραγματικό σενάριο.

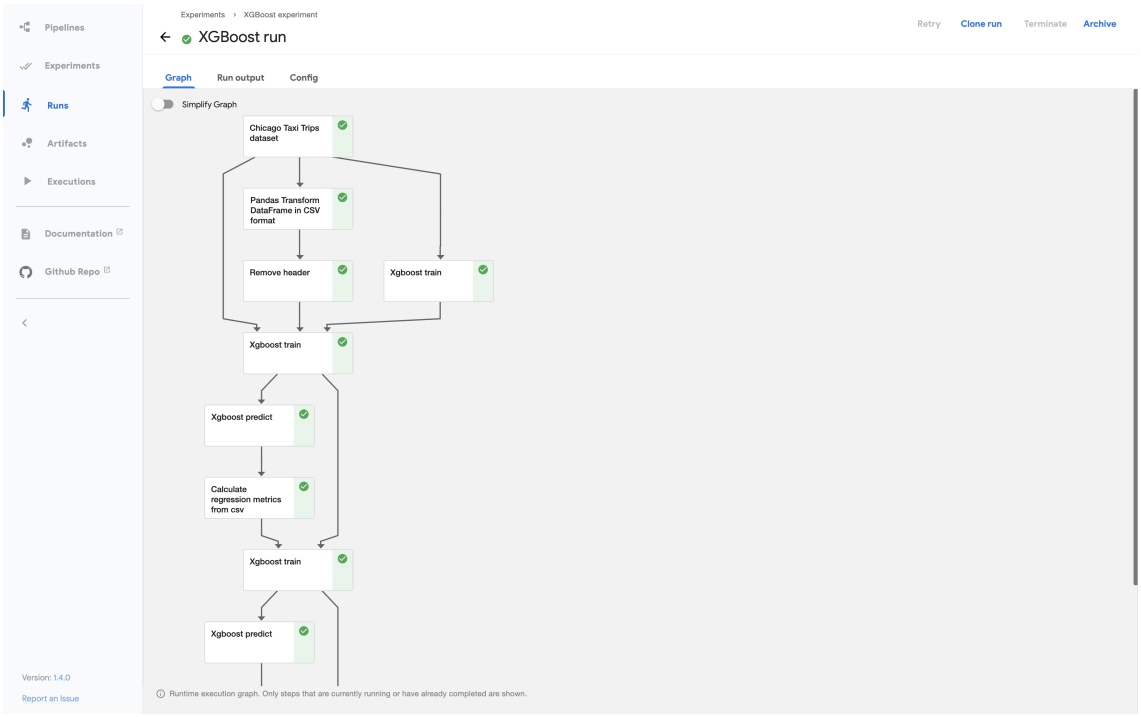
8.3 Αναπαραγωγιμότητα

Ο κώδικας που χρησιμοποιήθηκε για όλα τα στάδια της διπλωματικής έχει αναρτηθεί σε αποθετήριο στο Github.

Παραρτήματα

Παράρτημα

Α΄.1 Παράδειγμα γράφου που περιγράφει μία διοχέτευση



Ο κώδικας που περιγράφει τον παραπάνω γράφο είναι ο ακόλουθος:

```
@dsl.pipeline(
    name='XGBoost Trainer',
    description='A trainer that does end-to-end distributed training for XGBoost
        models.')
def xgb_train_pipeline(
    # options...
):
    output_template = str(output) + '/' + dsl.RUN_ID_PLACEHOLDER + '/data'
    analyze_output = output_template
    transform_output_train = os.path.join(output_template, 'train', 'part-*')
    transform_output_eval = os.path.join(output_template, 'eval', 'part-*')
    train_output = os.path.join(output_template, 'train_output')
    predict_output = os.path.join(output_template, 'predict_output')

    with dsl.ExitHandler(exit_op=dataproc_delete_cluster_op(
        # options...
    )):
        _create_cluster_op = dataproc_create_cluster_op(
            # options...
        )
        _analyze_op = dataproc_analyze_op(
            # options...
        ).after(_create_cluster_op).set_display_name('Analyzer')
        _transform_op = dataproc_transform_op(
            # options...
        ).after(_analyze_op).set_display_name('Transformer')
        _train_op = dataproc_train_op(
            # options...
        ).after(_transform_op).set_display_name('Trainer')
        _predict_op = dataproc_predict_op(
            # options...
        ).after(_train_op).set_display_name('Predictor')
        _cm_op = confusion_matrix_op(
            # options...
        ).after(_predict_op)
        _roc_op = roc_op(
            # options...
        ).after(_predict_op)

    dsl.get_pipeline_conf().add_op_transformer(
        gcp.use_gcp_secret('user-gcp-sa'))
```

A.2 Κώδικας Python που περιγράφει την διοχέτευση KFP

```
import json
import kfp
from kfp.components import func_to_container_op
from kfp import dsl


@func_to_container_op
def print_op(message: str):
    print(message)


def spark_preprocess():
    with open('manifest_preprocess.json') as f:
        manifest_preprocess = f.read()
    op = kfp.dsl.ResourceOp(
      name='Spark Submit Preprocess',
      k8s_resource=json.loads(manifest_preprocess),
      action='create',
      attribute_outputs={"name": "{.metadata.name}"})
    return op.outputs

def spark_main_ml():
    with open('manifest_main_ml.json') as f:
        manifest_main_ml = f.read()
    op = kfp.dsl.ResourceOp(
      name='Spark Submit ML',
      k8s_resource=json.loads(manifest_main_ml),
      action='create',
      attribute_outputs={"name": "{.metadata.name}"})
    return op.outputs


@dsl.pipeline(name='spark-ml-pipeline')
def spark_pipeline():
    spark_pp_out = spark_preprocess()
    poll_op = dsl.ContainerOp(
      name='poll-spark-preprocess-driver',
      image='bitnami/kubectl:1.21.12-debian-10-r32',
      command=['bash', '-c'],
      arguments=[f'while true; \
```

```
41                do OUT="$(kubectl get -n spark-operator sparkapp {spark_pp_out["name
                      "]} -o json | jq ".status.applicationState.state")"; \
42                if [ "$OUT" = "\\"COMPLETED"\\" ]; \
43                then echo "COMPLETED" > /tmp/output.txt; break; \
44                elif [ "$OUT" = "\\"FAILED"\\" ]; then echo "FAILED" > /tmp/output.
                      txt; break; \
45                else sleep 5; fi; done'],
46      file_outputs={
47        'output': '/tmp/output.txt'
48      }
49    )
50    with dsl.Condition(poll_op.outputs['output'] == 'COMPLETED'):
51        spark_ml_out = spark_main_ml()
52    with dsl.Condition(poll_op.output == 'FAILED', name='print-on-failure'):
53        print_op('Unexpected Error')
54    poll_op_2 = dsl.ContainerOp(
55      name='poll-spark-ml-driver',
56      image='bitnami/kubectl:1.21.12-debian-10-r32',
57      command=['bash', '-c'],
58      arguments=[f'while true; \
59                do OUT="$(kubectl get -n spark-operator sparkapp {spark_ml_out["name
                      "]} -o json | jq ".status.applicationState.state")"; \
60                if [ "$OUT" = "\\"COMPLETED"\\" ]; \
61                then echo "COMPLETED" > /tmp/output.txt; break; \
62                elif [ "$OUT" = "\\"FAILED"\\" ]; then echo "FAILED" > /tmp/output.
                      txt; break; \
63                else sleep 5; fi; done'],
64      file_outputs={
65        'output': '/tmp/output.txt'
66      }
67    )
68    with dsl.Condition(poll_op_2.output == 'COMPLETED'):
69        dsl.ContainerOp(
70        name='retrieve-logs-on-success',
71        image='bitnami/kubectl:1.21.12-debian-10-r32',
72        command=['bash', '-c'],
73        arguments=[f'kubectl logs {spark_ml_out["name"]}-driver -n spark-operator \
74                    | tee /tmp/output.txt'],
75        file_outputs={
76        'output': '/tmp/output.txt'
77        }
78    )
79    with dsl.Condition(poll_op_2.output == 'FAILED', name='print-on-failure'):
```

```
80          print_op('Unexpected Error')
81
82  # This is a hacky solution
83  AUTHSERVICE_SESSION_COOKIE="redacted"
84  client = kfp.Client(host="http://10.64.140.43.nip.io/pipeline",
85                      cookies=AUTHSERVICE_SESSION_COOKIE)
86
87  client.create_run_from_pipeline_func(
88      spark_pipeline,
89      arguments={},
90      experiment_name='Spark-Experiment',
91      namespace='admin')
```

Α΄.3 Αποτελέσματα του benchmark

INST	CORE	RAM	Time-q64-v2.4	Time-q70-v2.4	Time-q82-v2.4	Time-Avg
1	1	1	283,192	33,696	60,555	125,814
1	1	2	215,047	25,078	42,246	94,124
1	1	3	228,494	27,059	43,484	99,679
1	2	1	146,270	16,400	30,993	64,554
1	2	2	111,554	13,499	24,093	49,715
1	2	3	116,601	13,418	22,090	50,703
1	3	1	107,072	15,718	21,929	48,240
1	3	2	81,208	10,799	16,295	36,101
1	3	3	83,101	9,677	15,389	36,056
2	1	1	148,291	20,641	30,924	66,619
2	1	2	125,882	14,359	23,864	54,702
2	1	3	116,961	13,541	22,048	50,850
2	2	1	83,688	12,182	17,753	37,874
2	2	2	85,723	12,869	18,113	38,902
2	2	3	61,933	8,186	12,287	27,469
2	3	1	66,532	10,252	13,849	30,211
2	3	2	65,592	9,778	13,137	29,503
2	3	3	51,048	6,526	9,395	22,323
3	1	1	102,904	16,145	20,141	46,397
3	1	2	98,160	13,944	21,947	44,684
3	1	3	78,631	9,685	15,636	34,650
3	2	1	62,175	9,274	12,895	28,115
3	2	2	46,083	6,315	8,404	20,267
3	2	3	61,949	9,888	12,492	28,110
3	3	1	49,496	7,930	10,038	22,488
3	3	2	47,310	7,779	9,407	21,499
3	3	3	41,728	5,440	7,727	18,298
4	1	1	79,931	11,966	17,155	36,351
4	1	2	70,571	8,637	14,086	31,098
4	1	3	73,424	8,268	14,153	31,948
4	2	1	53,642	8,684	10,948	24,425
4	2	2	45,685	5,379	7,792	19,619
4	2	3	43,237	5,530	9,416	19,394
4	3	1	41,573	6,868	8,871	19,104
4	3	2	50,853	6,691	8,213	21,919
4	3	3	56,137	6,252	8,645	23,678

Table A.1: *Πίνακας με τους χρόνους των ερωτημάτων σε δευτερόλεπτα.*

Bibliography

[1] *Kubeflow*. https://www.kubeflow.org/.

[2] *Kubernetes*. https://kubernetes.io/.

[3] *Kubeflow Pipelines*. https://www.kubeflow.org/docs/components/pipelines/introduction/.

[4] *Apache Spark*. https://spark.apache.org/.

[5] *Kubernetes Operator for Apache Spark*. https://github.com/GoogleCloudPlatform/spark-on-k8s-operator.

[6] *I.C. Education, Containerization*. https://www.ibm.com/cloud/learn/containerization.

[7] *Open Container Initiative*. https://opencontainers.org/.

[8] *docker*. https://www.docker.com/.

[9] *Docker Registry*. https://docs.docker.com/registry/.

[10] *Project Calico*. https://www.tigera.io/project-calico/.

[11] *Introduction to SELinux*. https://web.mit.edu/rhel-doc/5/RHEL-5-manual/Deployment_Guide-en-US/ch-selinux.html.

[12] *role-based access control*. https://www.techtarget.com/searchsecurity/definition/role-based-access-control-RBAC#:~:text=Role%2Dbased%20access%20control%20(RBAC)%20is%20a%20method%20of,doesn't%20pertain%20to%20them.

[13] *OAuth 2.0*. https://oauth.net/2/.

[14] *etcd*. https://etcd.io/.

[15] *container-runtime*. https://www.aquasec.com/cloud-native-academy/container-security/container-runtime/.

[16] *Jupyter*. https://jupyter.org/.

[17] *PyTorch*. https://pytorch.org/.

[18] *TensorFlow*. https://www.tensorflow.org/.

[19] *Wikipedia - MapReduce*. https://en.wikipedia.org/wiki/MapReduce.

[20] *Hadoop Yarn*. https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html.

[21] *Apache Mesos*. https://mesos.apache.org/.

[22] *Hadoop Distributed File System*. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[23] *Apache Cassandra*. https://cassandra.apache.org/_/index.html.

[24] *OpenStack Swift*. https://docs.openstack.org/swift/latest/.

[25] *Amazon S3*. https://aws.amazon.com/s3/.

[26] *Openstack*. https://www.openstack.org/.

[27] *Dqlite*. https://dqlite.io/.

[28] Diego Ongaro και John Ousterhout. *In Search of an Understandable Consensus Algorithm*. *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, σελίδα 305–320, USA, 2014. USENIX Association.

[29] *Charmed Kubeflow*. https://charmed-kubeflow.io/.

[30] *Juju*. https://juju.is/.

[31] *spark-on-k8s-operator*. https://github.com/GoogleCloudPlatform/spark-on-k8s-operator.

[32] *Adult Data Set*. https://archive.ics.uci.edu/ml/datasets/Adult/.

[33] *CNI - the Container Network Interface*. https://github.com/containernetworking/cni.

[34] *Juju - Controller Agent*. https://juju.is/docs/olm/controller-agent.

[35] *dex*. https://dexidp.io/.

[36] *OpenID Connect*. https://openid.net/connect/.

[37] *WebHDFS*. https://hadoop.apache.org/docs/r1.0.4/webhdfs.html.

[38] *Hadoop - Cluster Setup*. https://hadoop.apache.org/docs/r2.10.2/hadoop-project-dist/hadoop-common/ClusterSetup.html.

[39] *Helm*. https://helm.sh/.

[40] *Kubernetes Operator for Apache Spark Design*. https://github.com/GoogleCloudPlatform/spark-on-k8s-operator/blob/master/docs/design.md.

[41] *Mutating Admission Webhook*. https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/.

[42] *Kubernetes Volumes*. https://kubernetes.io/docs/concepts/storage/volumes/.

[43] *(Download) UCI Adult Data Set Description*. https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.names.

[44] *Machine Learning Library (MLlib)*. https://spark.apache.org/docs/latest/ml-guide.html.

[45] Raghunath Othayoth Nambiar και Meikel Poess. *The Making of TPC-DS*. *Proceedings of the 32nd International Conference on Very Large Data Bases*, VLDB '06, σελίδα 1049–1058. VLDB Endowment, 2006.

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια

API	Application Programming Interface
CLI	Command Line Interface
CNI	Container Network Interface
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
DNS	Dynamic Name Resolution
GB	Gigabyte
HDFS	Hadoop Distributed File System
IP	Internet Protocol
JSON	JavaScript Object Notation
KFP	Kubeflow Pipelines
ML	Machine Learning
OCI	Open Container Initiative
OS	Operating System
POSIX	Portable Operating System Interface
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory
RBAC	Role-based Access Control
RDD	Resilient Distributed Dataset
REST	Representational State Transfer
SCTP	Stream Control Transmission Protocol
SDK	Software Development Kit
SELinux	Security Enhanced Linux
SQL	Structured Query Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VM	Virtual Machine
YAML	Yet Another Markup Language

Απόδοση ξενόγλωσσων όρων

Απόδοση	Ξενόγλωσσος όρος
αδειοδότηση	licensing
ανάθεση	allocation
αναπτύσσω	develop
ανθεκτικό	resilient
ανοιχτού κώδικα	open source
αντίγραφο	replica
αποθετήριο	repository
αποθηκευτικός χώρος	storage space
απομονωμένη	isolated
αποσφαλμάτωση	debugging
ασφάλεια δικτύου	network security
αφαιρετικά	abstractedly
βάση δεδομένων	database
βρόγχος ελέγχου	control loop
δήλωση	declaration
δαίμων	daemon
διαθεσιμότητας	availability
διεπαφή	interface
διεπαφή-πελάτης	client
διεπαφή προγραμματισμού εφαρμογών	application programming interface
δικαιώματα ασφάλειας	permissions
δικτύωση	networking
διοχέτευση	pipeline
δρομέας υποβολής	submission runner
δρομολογώ	schedule
εικονική μηχανή	virtual machine
εικονικοποίηση	virtualization
εκκινηθεί	boot
εκτελέσιμο	executable
ελαφριά	lightweight
ελεγκτής αναπαραγωγής	replication controller
ελεγκτής	controller
ενδιάμεσο προσωρινό αρχείο	shuffle file
ενορχήστρωση	orchestration

ενσωματωμένη	embedded
επίπεδο δικτύου	network layer
επίπεδο ελέγχου	control plane
εργάτης	worker
εργαλείο γραμμής εντολών	command line tool
εφαρμογή	application
κάψουλα	pod
κακόβουλος κώδικα	malware
καταγραφή συμβάντων	logging
κατανεμημένο σύστημα αποθήκευσης	distributed storage
κλιμάκωση	scaling
κλιμακόσιμων	scalable
κόμβος	node
κόμβος δεδομένων	datanode
κόμβος ονομάτων	namenode
κόμβος σκλάβος	slave
κόμβος του σημείου ελέγχου	checkpoint node
κύκλου ζωής	lifecycle
λειτουργικό σύστημα	operating system
λειτουργικό σύστημα υποδοχής	host operating system
λύση	solution
μειώνει	reduce
μεταβλητή περιβάλλοντος	environment variable
μετασχηματισμός δεδομένων	data transformation
μεταφορτώνω	upload
μετρικές	metrics
μηχάνημα-οικοδεσπότη	host machine
μηχανή εκτέλεσης	engine
μονοπάτι	path
μόνιμη	resilient
οδηγός	driver
πακέτο λογισμικού	package
παράγοντας αναπαραγωγής	replication factor
παραγωγικό περιβάλλον	production environment
παρακολούθηση	monitoring
παρατηρητής	watcher
πηγή αλήθειας	source of truth
πλεονάζουσα συστοιχία ανεξάρτητων δίσκων	redundant array of independent disks
πράκτορας ελέγχου	control agent
πράκτορας	agent
προκαθορισμένη	default
προκαθορισμένο μονοπάτι	default path
πυρήνας	kernel

ροή εργασίας	workflow
ρυθμίσεις	configuration
σύνολο δεδομένων	dataset
συμβατότητα	compatibility
συσσωμάτωση	aggregation
σύστημα-οικοδεσπότη	host system
συστοιχία	cluster
σύνολο εκπαίδευσης	train set
σύνολο ελέγχου	test set
σύστημα αρχείων	file system
σύστημα-υποδομή	infrastructure
τοπικό σύστημα	local system
υπηρεσία	service
υπολογιστικό μοτίβο	programming paradigm
υπολογιστικό νέφος	cloud
υπολογιστικών πόρων	computational resources
φορητό	portable
χαρτογράφηση	mapping
χρόνο εκκίνησης	startup time