ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Containerization vs Bare Metal: distributed computing performance using Apache Spark

Διπλωματική εργασία

Μαργαρίτα Ελένη Τσαρμποπούλου

Επιβλέπων καθηγητής: Γεώργιος Γκούμας

Αναπληρωτής Καθηγητής, Ε.Μ.Π.

Αθήνα, Μάρτιος 2024

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Containerization vs Bare Metal: distributed computing performance using Apache Spark

Διπλωματική εργασία

Μαργαρίτα Ελένη Τσαρμποπούλου

Επιβλέπων καθηγητής: Γεώργιος Γκούμας

Αναπληρωτής Καθηγητής, Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 22η Μαρτίου, 2024.

(Υπογραφή)	(Υπογραφή)	(Υπογραφή)
.....	.....	.....
Γεώργιος Γκούμας	Νεκτάριος Κοζύρης	Διονύσιος Πνευματικάτος
Αναπληρωτής Καθηγητής, Ε.Μ.Π.	Καθηγητής, Ε.Μ.Π.	Καθηγητής, Ε.Μ.Π.

Αθήνα, Μάρτιος 2024

(Υπογραφή)

.....

Μαργαρίτα Ελένη Τσαρμποπούλου

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

στην οικογένειά μου

Περίληψη

Αυτή η έρευνα εξερευνά τις συμβιβαστικές λύσεις απόδοσης μεταξύ των περιβαλλόντων που βασίζονται σε container και τα περιβάλλοντα bare metal για την εκτέλεση εφαρμογών Apache Spark, εστιάζοντας συγκεκριμένα στα dashboard ανίχνευσης τροχαίων περιστατικών. Ο κατανεμημένος υπολογισμός, ένα θεμελιώδες στοιχείο των σύγχρονων εφαρμογών που βασίζονται σε δεδομένα, παρουσιάζει ένα φάσμα επιλογών ανάπτυξης, κάθε μία με διακριτά πλεονεκτήματα και προκλήσεις. Αυτή η έρευνα εμβαθύνει στις θεωρητικές βάσεις του κατανεμημένου υπολογισμού, του containerization και των υλοποιήσεων bare metal, προετοιμάζοντας το έδαφος για μια συγκριτική ανάλυση που βασίζεται σε μετρήσεις απόδοσης, κλιμακωσιμότητας, χρήσης πόρων και λειτουργικής πολυπλοκότητας.

Διεξήχθησαν μια σειρά πειραμάτων χρησιμοποιώντας το Apache Spark για την εκτέλεση μοντέλων μηχανικής μάθησης τόσο σε περιβάλλοντα container όσο και σε περιβάλλοντα bare metal. Τα κριτήρια αξιολόγησης σχεδιάστηκαν έτσι ώστε να αντανακλούν τις απαιτήσεις των εφαρμογών πολλαπλών κατόχων στον πραγματικό κόσμο, τονίζοντας την ανταπόκριση και την κλιμακωσιμότητα των dashboard οπτικοποίησης που είναι κρίσιμα για την ανίχνευση περιστατικών. Τα πειραματικά αποτελέσματα αποκαλύπτουν ότι, ενώ τα περιβάλλοντα με container προσφέρουν βελτιωμένη κλιμακωσιμότητα, ευελιξία στην ανάπτυξη και μειωμένη λειτουργική πολυπλοκότητα, φέρουν ελάχιστη επιβάρυνση απόδοσης σε σύγκριση με τις διαμορφώσεις bare metal. Αντίθετα, τα περιβάλλοντα bare metal επιδεικνύουν ελαφρώς ανώτερη υπολογιστική αποδοτικότητα, αποδίδοντας στην άμεση πρόσβαση στο υλικό, παρόλο που αυτό συνεπάγεται μειωμένη ευελιξία και έλλειψη ανοχής σε σφάλματα.

Η έρευνα καταλήγει ότι η επιλογή μεταξύ containerization και bare metal για τις αναπτύξεις Apache Spark εξαρτάται από τις συγκεκριμένες απαιτήσεις της εφαρμογής και το πλαίσιο χρήσης της. Τα περιβάλλοντα με container προτιμώνται για την προσαρμοστικότητά τους σε σενάρια βασισμένα στο cloud και πολλαπλών κατόχων, όπου η κλιμακωσιμότητα και η λειτουργική αποδοτικότητα είναι καθοριστικής σημασίας. Οι διαμορφώσεις bare metal, ωστόσο, μπορεί να προτιμώνται σε πλαίσια που απαιτούν τη μέγιστη υπολογιστική απόδοση με σταθερά χαρακτηριστικά φορτίου. Αυτή η έρευνα συνεισφέρει στην ευρύτερη κατανόηση των αρχιτεκτονικών κατανεμημένου υπολογισμού, προσφέροντας πληροφορίες για τις επιπτώσεις τους στον σχεδιασμό και τη βελτιστοποίηση υψηλής απόδοσης, κλιμακώσιμων dashboard για διάφορους ενδιαφερόμενους και περιπτώσεις χρήσης.

Λέξεις-κλειδιά — dashboard, μηχανική μάθηση, κατανεμημένος υπολογισμός, Kubernetes, bare metal, Apache Spark, containerization, ανίχνευση περιστατικών, ανάλυση απόδοσης, κλιμακωσιμότητα, χρήση πόρων

Abstract

This research explores the performance trade-offs between containerized and bare metal environments for running Apache Spark applications, specifically focusing on incident detection dashboards. Distributed computing, a cornerstone of modern data-intensive applications, presents a spectrum of deployment options, each with distinct advantages and challenges. This research delves into the theoretical underpinnings of distributed computing, containerization, and bare metal implementations, setting the stage for a comparative analysis grounded in performance metrics, scalability, resource utilization, and operational complexity.

A series of experiments were conducted using Apache Spark to execute machine learning algorithms within both containerized and bare metal settings. The evaluation criteria were designed to reflect real-world multi-tenancy application demands, emphasizing the responsiveness and scalability of visualization dashboards crucial for incident detection. The experimental results reveal that while containerized environments offer enhanced scalability, deployment flexibility, and reduced operational complexity, they incur a minimal performance overhead compared to bare metal setups. Conversely, bare metal environments demonstrate marginally superior computational efficiency, attributable to direct hardware access, albeit at the cost of reduced flexibility and lack of fault tolerance.

The research concludes that the choice between containerization and bare metal for Apache Spark deployments hinges on specific application requirements and context. Containerized environments are favored for their adaptability in cloud-based, multi-tenant scenarios, where scalability and operational efficiency are paramount. Bare metal deployments, however, may be preferred in contexts demanding maximal computational performance with stable workload characteristics. This research contributes to the broader understanding of distributed computing architectures, offering insights into their implications for the design and optimization of high-performance, scalable dashboards for diverse stakeholders and use cases.

Keywords — dashboard, machine learning, distributed computing, Kubernetes, bare metal, Apache Spark, containerization, incident detection, performance analysis, scalability, resource utilization

Ευχαριστίες

Αυτή η διπλωματική εργασία μου προσέφερε την ευκαιρία να μελετήσω βαθύτερα διάφορα θέματα από τον τομέα του Κατανεμημένου υπολογισμού και να λάβω μια γεύση από την ερευνητική διαδικασία. Ευχαριστώ θερμά τον επιβλέποντα καθηγητή μου κ. Γεώργιο Γκούμα, αναπληρωτή καθηγητή Ε.Μ.Π., που κατά τη διάρκεια της φοίτησης μου, μου καλλιέργησε το ενδιαφέρον στα Λειτουργικά Συστήματα και μου έδωσε την ευκαιρία να εκπονήσω αυτήν την διπλωματική υπό την επίβλεψή του.

Σημαντικότεροι στην πορεία μου σε αυτό το πενταετές ταξίδι αλλά και στην υπόλοιποι ζωή μου είναι η οικογένειά μου. Τους ευχαριστώ για την στήριξή τους σε κάθε μου βήμα και τις υπέροχες στιγμές. Ευχαριστώ και τους φίλους μου για τα αξέχαστα φοιτητικά χρόνια.

Μαργαρίτα Ελένη Τσαρμποπούλου

Μάρτιος 2024

Contents

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Εκτεταμένη περίληψη στα Ελληνικά

Εισαγωγή

Αντικείμενο της διπλωματικής

Το αντικείμενο αυτής της έρευνας είναι να αξιολογήσει την απόδοση του Apache Spark όταν αναπτύσσεται σε containerized περιβάλλοντα σε αντίθεση με τις ρυθμίσεις bare metal, ιδιαίτερα στο πλαίσιο της ανίχνευσης περιστατικών. Αυτές οι δύο αρχιτεκτονικές ανάπτυξης αντιπροσωπεύουν διαφορετικές μεθοδολογίες στη διαχείριση εφαρμογών μεγάλης κλίμακας, η καθεμία με διακριτά οφέλη και περιορισμούς.

Θεωρητικό υπόβαθρο

Σύντομη εισαγωγή στις έννοιες

Κατανεμημένος υπολογισμός

Το πεδίο του κατανεμημένου υπολογισμού καλύπτει μια ευρεία γκάμα υπολογιστικών παραδειγμάτων και μοντέλων πρόσβασης πληροφοριών που εκτείνονται σε πολλαπλά στοιχεία επεξεργασίας συνδεδεμένα μέσω διαφόρων μορφών δικτύων επικοινωνίας. Αυτά τα δίκτυα μπορεί να καλύπτουν τοπικά περιβάλλοντα ή να εκτείνονται σε ευρύτερη περιοχή, παρουσιάζοντας ένα ποικίλο τοπίο για τον σχεδιασμό και την υλοποίηση κατανεμημένων συστημάτων. Η κύρια πρόκληση στον κατανεμημένο υπολογισμό είναι ο αποτελεσματικός συντονισμός και διαχείριση των υπολογιστικών εργασιών που διανέμονται σε διάφορους κόμβους, προάγοντας τη συνεργασία και τον παραλληλισμό, ενώ αντιμετωπίζονται ζητήματα όπως η ανοχή σφαλμάτων και η κλιμακωσιμότητα (Chaisawat & Vorakulpipat, 2020).

Apache Spark

Το Apache Spark είναι ένα πλαίσιο κατανεμημένου υπολογισμού ανοιχτού κώδικα που έχει σχεδιαστεί για την επεξεργασία μεγάλων συνόλων δεδομένων και την εκτέλεση πολύπλοκων εργασιών ανάλυσης δεδομένων. Παρέχει μια ευέλικτη πλατφόρμα για την κατανεμημένη επεξεργασία δεδομένων, προσφέροντας χαρακτηριστικά όπως η αποθήκευση δεδομένων στη μνήμη, η ανοχή σφαλμάτων και η υποστήριξη διαφόρων γλωσσών προγραμματισμού.

Το Spark λειτουργεί βασιζόμενο στην έννοια των Resilient Distributed Datasets (RDDs), τα οποία είναι κατανεμημένες συλλογές δεδομένων που μπορούν να επεξεργαστούν παράλληλα σε ένα σύνολο μηχανημάτων. Τα RDD επιτρέπουν στο Spark να διαχειρίζεται αποτελεσματικά τη διαίρεση, τη διανομή και τον υπολογισμό των δεδομένων, καθιστώντας το κατάλληλο για εργασίες που κυμαίνονται από την επεξεργασία δεδομένων πακέτων έως τη ροή δεδομένων σε πραγματικό χρόνο (K & G, 2022).

Bare metal

Η υλοποίηση bare metal περιλαμβάνει την εκτέλεση εφαρμογών απευθείας στο φυσικό υλικό χωρίς κανένα ενδιάμεσο στρώμα. Η ακατέργαστη υπολογιστική ισχύς και το ελάχιστο πρόσθετο φορτίο που σχετίζεται με εφαρμογές bare metal τις καθιστούν μια συναρπαστική επιλογή, ιδιαίτερα για εφαρμογές που απαιτούν υψηλή απόδοση (Lee & Fox, 2019). Ωστόσο, οι προκλήσεις που σχετίζονται με την επεκτασιμότητα (Zhang et al., 2020) και τον κοινόχρηστο χώρο πόρων πρέπει να εξεταστούν προσεκτικά.

Εικονοποίηση

Η εικονοποίηση περιλαμβάνει τη δημιουργία εικονικών περιβαλλόντων υπολογιστικών πόρων εντός ενός μόνο φυσικού υπολογιστή. Η εικονοποίηση επιτρέπει την ταυτόχρονη εκτέλεση πολλαπλών λειτουργικών συστημάτων σε έναν υποκείμενο επεξεργαστή. Στον τομέα της εικονοποίησης, οι συμβιβασμοί μεταξύ απομόνωσης, επιβάρυνσης πόρων και κλιμακωσιμότητας γίνονται κρίσιμες εκτιμήσεις (Campbell & Jeronimo, n.d.).

Containerization

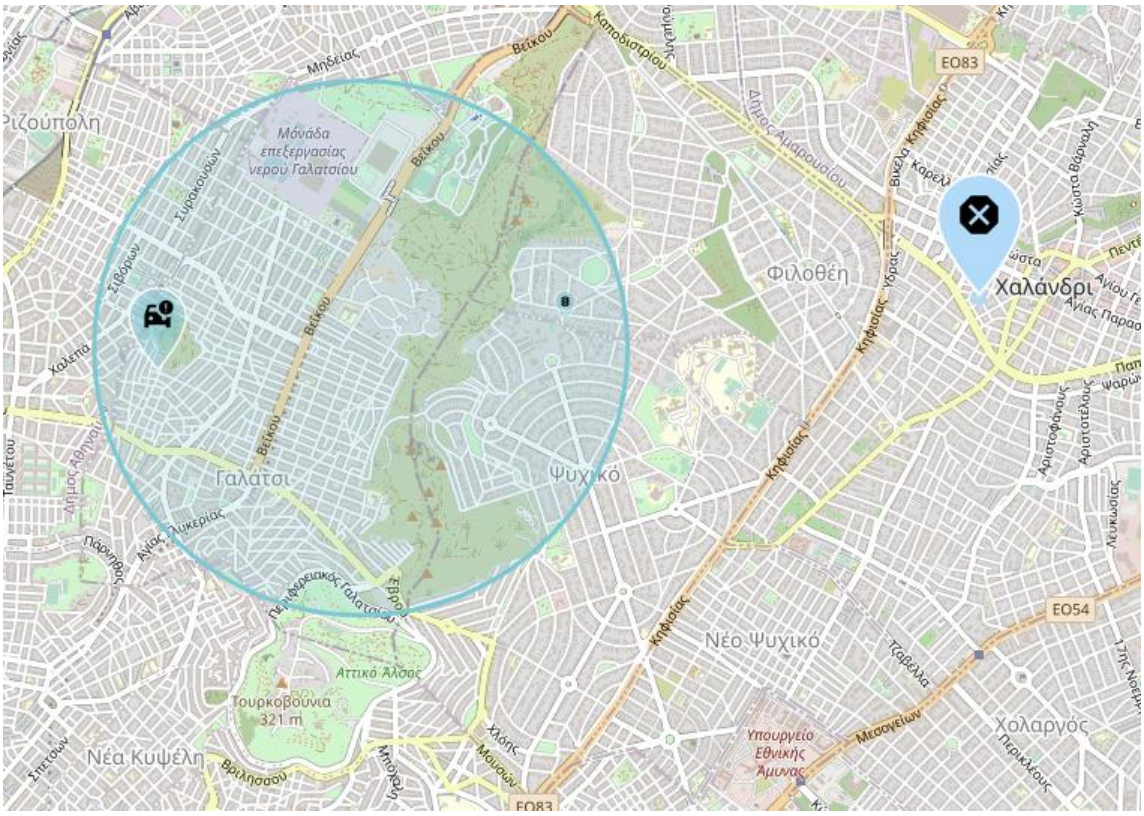
Το containerization έχει αποκτήσει μεγάλη σημασία λόγω της ικανότητάς του να ενθυλακώνει εφαρμογές και τις εξαρτήσεις τους σε ελαφριές, φορητές μονάδες που ονομάζονται containers. Το Docker και το Kubernetes είναι κεντρικά σε αυτό το παράδειγμα, διευκολύνοντας τη δημιουργία, την ανάπτυξη και την κλιμάκωση containerized εφαρμογών. Στις επόμενες ενότητες, θα εξετάσουμε τον αντίκτυπο της containerized αρχιτεκτονικής στο Apache Spark σε κατανεμημένα υπολογιστικά περιβάλλοντα, εξερευνώντας πώς επηρεάζει παράγοντες όπως η απόδοση, η κλιμακωσιμότητα και η ευκολία ανάπτυξης. Κατανοώντας τις λεπτομέρειες της containerized αρχιτεκτονικής, μπορούμε να καταλάβουμε καλύτερα τον ρόλο της στο σχηματισμό multi-user εφαρμογών και τους συμβιβασμούς που συνεπάγεται η υιοθέτηση της τεχνολογίας αυτής.

Πλεονεκτήματα και μειονεκτήματα της κάθε αρχιτεκτονικής

Οι ρυθμίσεις bare metal ξεχωρίζουν στην απόδοση, προσφέροντας άμεση πρόσβαση στο υλικό, γεγονός το οποίο ωφελεί τις εφαρμογές που απαιτούν έντονο υπολογισμό, με ένα πιο απλό στήσιμο για σταθερές απαιτήσεις (USENIX Association., 2003). Ωστόσο, η έλλειψή τους σε ευελιξία και κλιμακωσιμότητα, μπορεί να είναι δαπανηρή για ποίκιλα φορτία εργασίας και μπορεί να έχει υψηλότερους χρόνους συντήρησης. Η εικονοποίηση παρέχει ισχυρή απομόνωση, ασφάλεια και αφαίρεση υλικού, επιτρέποντας τη δυναμική κατανομή πόρων και την εκτέλεση σε διάφορα υλικά, αλλά υποφέρει από ένταση πόρων, πιθανές απώλειες απόδοσης και πολυπλοκότητα στη διαχείριση (Bhardwaj & Krishna, 2021). Το containerization ενθυλακώνει εφαρμογές με τις εξαρτήσεις τους σε ελαφριά, φορητά containers, προσφέροντας αποδοτικότητα πόρων, ταχεία ανάπτυξη και ανοχή σφαλμάτων. Ωστόσο, αντιμετωπίζει κινδύνους ασφαλείας από κοινά πυρήνες ΛΣ, πολυπλοκότητα στη διαχείριση μεγάλης κλίμακας και επιβάρυνση απόδοσης σε σύγκριση με το bare metal. Κάθε ρύθμιση παρουσιάζει έναν συμβιβασμό μεταξύ απόδοσης, ευελιξίας και ευκολίας διαχείρισης, αντιμετωπίζοντας διαφορετικές ανάγκες και περιορισμούς.

Κίνητρο Εργασίας

Η αυξανόμενη ζήτηση για dashboard στον κόσμο που βασίζεται στα δεδομένα, ιδίως για εφαρμογές όπως η πρόβλεψη κυκλοφορίας και η ανίχνευση περιστατικών, καθιστά απαραίτητη την ταχεία επεξεργασία δεδομένων και την αποδοτική λειτουργία μοντέλων μηχανικής μάθησης. Ωστόσο, τα κεντρικοποιημένα συστήματα σε έναν μόνο υπολογιστή συχνά υποφέρουν από ζητήματα απόδοσης, προκαλώντας αργές οπτικοποιήσεις και μεγάλους χρόνους αναμονής. Οι πίνακες ελέγχου χρησιμοποιούν δυναμικές "push" και "pull" για πραγματικού χρόνου ενημερώσεις δεδομένων. Στην πρόβλεψη κίνησης, τα dashboard λειτουργούν σε κατάσταση "pull", όπου κάνουν αίτημα για προβλέψεις σε συγκεκριμένες περιοχές, οδηγώντας στην εκτέλεση αλγορίθμων μηχανικής μάθησης. Από την άλλη πλευρά, στην ανίχνευση περιστατικών, χρησιμοποιείται μια προσέγγιση "push", όπου οι αλγόριθμοι λειτουργούν ανεξάρτητα και συνεχώς, εντοπίζοντας και αναφέροντας περιστατικά καθώς συμβαίνουν. Αντιμετωπίζοντας αυτές τις προκλήσεις, η υιοθέτηση της κατανεμημένης επεξεργασίας βελτιώνει σημαντικά την ανταπόκριση του συστήματος επιτρέποντας την κατανεμημένη εκτέλεση μοντέλων μηχανικής μάθησης, εκμεταλλευόμενη την υποδομή cloud για ευελιξία, κλιμακωσιμότητα και αντοχή σε σφάλματα. Η μετάβαση στον κατανεμημένο υπολογισμό, διευκολυνόμενη από τεχνολογίες όπως το containerization, στοχεύει στη βελτίωση της χρηστικότητας, τη μείωση των χρόνων αναμονής και την αποδοτική διαχείριση των πόρων. Αυτή η προσέγγιση αξιολογεί την εκτέλεση αλγορίθμων μηχανικής μάθησης σε εφαρμογές ανίχνευσης περιστατικών, υποστηρίζοντας τα κατανεμημένα περιβάλλοντα προς βελτιστοποίηση της απόδοσης και της ανταπόκρισης, και τονίζει τη συνεργασία μεταξύ κατανεμημένης επεξεργασίας και τεχνολογιών containerization στην ανάπτυξη υψηλής απόδοσης, κλιμακώσιμων dashboard για διάφορους stakeholders και use cases.



1.1-1: Εικόνα από το dashboard του έργου FRONTIER, όπου φαίνεται η οπτικοποίηση κάποιων ανιχνευμένων τροχαίων ανωμαλιών

Κατανεμημένη Επεξεργασία και Μηχανική Μάθηση

Μέθοδοι για κατανεμημένη επεξεργασία

Τα πλαίσια κατανεμημένης επεξεργασίας είναι πολλά και χωρίζονται σε 3 κύριες κατηγορίες, οι οποίες είναι τα συστήματα database, general και purpose-built types, καθένα με μοναδικά πλεονεκτήματα και πλαίσια εφαρμογής βασισμένα σε χρήσεις, ανάγκες απόδοσης και κλίμακες δεδομένων (Galakatos et al., 2017).

Τα συστήματα βάσεων δεδομένων, ως επεκτάσεις DBMS, προσφέρουν ολοκληρωμένα περιβάλλοντα βελτιστοποιημένα για την έντονη φύση των δεδομένων της μηχανικής μάθησης, επωφελούμενα από προηγμένη διαχείριση δεδομένων και ελέγχους συναλλαγών.

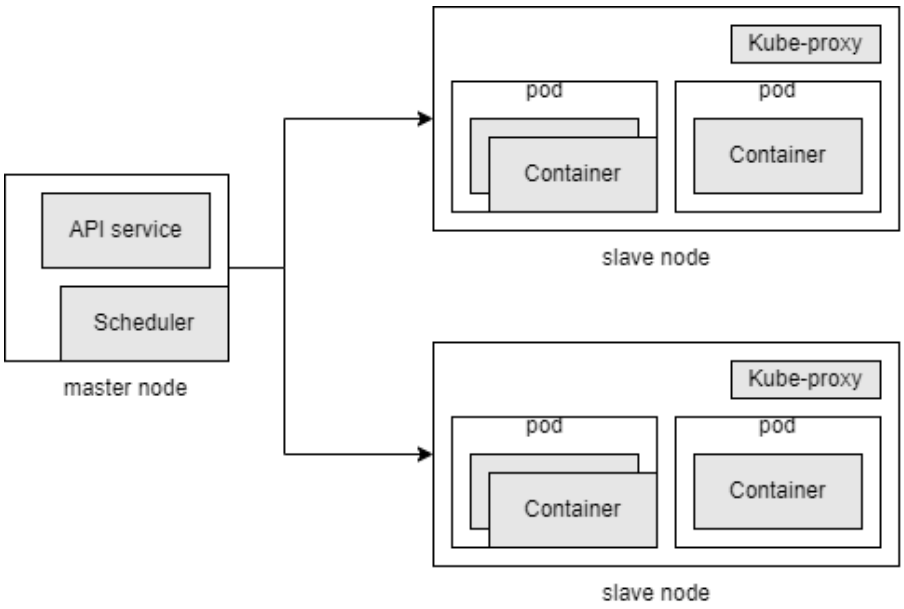
Τα γενικά πλαίσια απλοποιούν την ανάπτυξη αλγορίθμων μηχανικής μάθησης μέσω API υψηλού επιπέδου, υποστηρίζοντας ευελιξία και κλιμακωσιμότητα με εργαλεία όπως το MPI για υπολογισμό υψηλής απόδοσης, το Hadoop για εργασίες MapReduce και το Spark για in-memory επεξεργασία και αποδοτική υποστήριξη επαναληπτικών αλγορίθμων.

Τα purpose-built συστήματα είναι βελτιστοποιημένα για συγκεκριμένες εργασίες, προσφέροντας αποδοτικότητα αλλά περιορισμένη ευελιξία, με παραδείγματα όπως το SystemML για λειτουργίες matrix, το OptiML για γραμμική άλγεβρα και το Hogwild! για στοχαστική καθοδική βελτιστοποίηση.

Αυτή η έρευνα επικεντρώνεται στο Apache Spark, επιλεγμένο για τις δυνατότητες επεξεργασίας στη μνήμη, την αντοχή σε σφάλματα και την εκτεταμένη υποστήριξη της βιβλιοθήκης μηχανικής μάθησης που έχει.

Kubernetes και Kubernetes σε Spark

Το Kubernetes είναι μια δημοφιλής επιλογή για την ανάπτυξη εφαρμογών βασισμένων σε containers, τόσο σε φυσικά μηχανήματα όσο και σε περιβάλλοντα cloud Platform-as-a-Service (PaaS), λόγω της ικανότητάς του να κλιμακώνει δυναμικά εφαρμογές ανταποκρινόμενο σε αλλαγές φορτίου εργασίας. Λειτουργεί βάσει μιας αρχιτεκτονικής master- worker, όπου ο κύριος κόμβος οργανώνει τους εργαζόμενους κόμβους που εκτελούν τις πραγματικές υπηρεσίες. Το Kubernetes εισάγει τα pods ως τις θεμελιώδεις μονάδες ανάπτυξης, τα οποία μπορούν να περιέχουν ένα ή περισσότερα containers που μοιράζονται τον ίδιο χώρο δικτύου, επιτρέποντας αποτελεσματική επικοινωνία μεταξύ pods σε διαφορετικά μηχανήματα μέσω του Kube-proxy. Τα pods σχεδιάζονται να είναι ελαφριά, ζητώντας συνήθως έναν πυρήνα CPU ή λιγότερο, διευκολύνοντας την ευέλικτη ανάπτυξη σε κόμβους για να ενισχύσουν την ευελιξία και την αξιοπιστία της εφαρμογής. Επιπλέον, το Spark μπορεί να ενσωματωθεί με το Kubernetes για τη διαχείριση cluster, όπου οι drivers και οι executors του Spark λειτουργούν μέσα σε pods, διαχειριζόμενοι από τον scheduler του Kubernetes. Αυτή η ρύθμιση επιτρέπει την αποτελεσματική εκτέλεση και καθαρισμό της εφαρμογής, με το pod του οδηγού να διατηρεί τα αρχεία καταγραφής μετά την ολοκλήρωση μέχρι να αφαιρεθεί, επιδεικνύοντας την αποδοτικότητα του Kubernetes στη διαχείριση κατανεμημένων εφαρμογών (Zhu et al., 2020a).



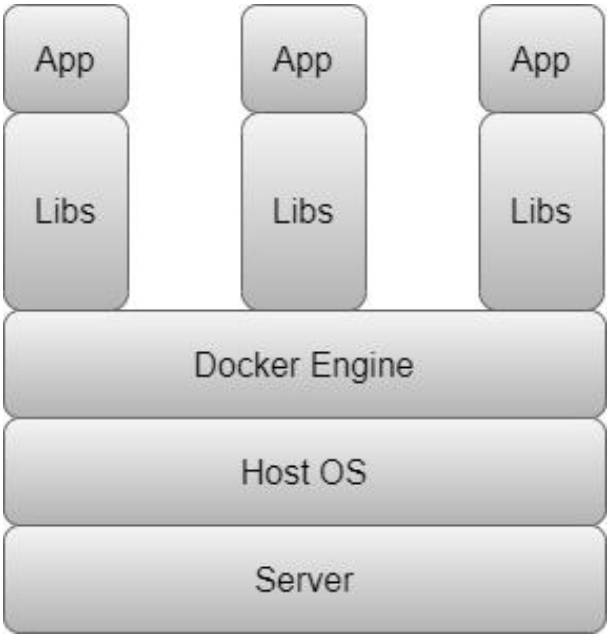
1.1-1: Αρχιτεκτονική Kubernetes

Σύγκριση απόδοσης των διαφόρων αρχιτεκτονικών

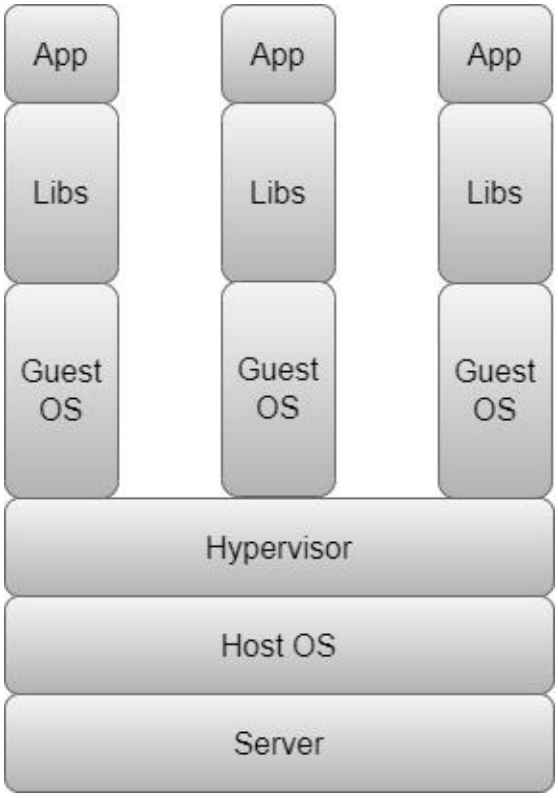
Containerization vs Εικονοποίηση

Το Docker χρησιμοποιεί το containerization για να απομονώνει διεργασίες, επιτρέποντας σε πολλαπλά containers να τρέχουν στον ίδιο υπολογιστικό κόμβο με ελάχιστο αντίκτυπο στην απόδοση. Αυτό συμβαίνει επειδή το containerization λειτουργεί στο επίπεδο του πυρήνα, αποφεύγοντας την επιβάρυνση που συνδέεται με την εικονικοποίηση ολόκληρων συστημάτων. Όταν εκκινείται ένα container του Docker, η ορισμένη διεργασία ή διεργασίες τρέχουν απευθείας στον υπολογιστικό κόμβο, χωρίς την ανάγκη για πλήρη εικονικοποίηση. Αυτό διαφέρει έντονα από τις εικονικές μηχανές (VMs), όπου η ενεργοποίηση ενός VM περιλαμβάνει τον υπολογιστή να εικονικοποιεί ολόκληρα συστήματα, συμπεριλαμβανομένων CPU, RAM και αποθηκευτικού χώρου, απαιτώντας ένα ολόκληρο λειτουργικό σύστημα για κάθε VM (Bhat, 2018).

Αυτή η μέθοδος αυξάνει σημαντικά το φορτίο των πόρων, καθώς η λειτουργία ενός λειτουργικού συστήματος εντός άλλου συνεπάγεται συσσώρευση χρήσης πόρων. Τα containers μοιράζονται τον πυρήνα του λειτουργικού συστήματος του host, εξαλείφοντας την ανάγκη για ξεχωριστές εκδόσεις λειτουργικού συστήματος, ενισχύοντας έτσι την απόδοση, βελτιστοποιώντας τη χρήση πόρων και μειώνοντας τη σπατάλη υπολογιστικών πόρων. Αυτή η αποδοτικότητα στην προσέγγιση του Docker έναντι της πιο εντατικής σε πόρους φύσης των VMs τονίζει τα πλεονεκτήματα του containerization στη διαχείριση και την ανάπτυξη εφαρμογών σε μεμονωμένους υπολογιστικούς κόμβους.



1.1-2: Αναπαράσταση 3 εφαρμογών που εκτελούνται σε 3 διαφορετικά container.



1.1-3: Αναπαράσταση τριών εφαρμογών που εκτελούνται σε τρεις διαφορετικές εικονικές μηχανές.

Containerization vs bare metal

Στη σύγκριση του containerization με το bare metal για τις αναπτύξεις του Apache Spark, κύριες σκέψεις περιλαμβάνουν τη χρήση πόρων, τους χρόνους εκκίνησης, την κλιμακωσιμότητα, την επιβάρυνση απόδοσης, την πολυπλοκότητα λειτουργίας, την ευελιξία και τη φορητότητα. Το containerization ξεχωρίζει σε multi-tenant περιβάλλοντα με την αποδοτική χρήση πόρων και την απομόνωση πολλαπλών εργασιών Spark στο ίδιο υλικό. Προσφέρει ταχύτερους χρόνους εκκίνησης και ενισχυμένη κλιμακωσιμότητα, πλεονεκτικά για περιβάλλοντα cloud με διακυμάνσεις φορτίου εργασίας. Παρόλο που το containerization επιφέρει μια ελαφριά επιβάρυνση απόδοσης λόγω του στρώματος αφαίρεσης του, αυτό είναι γενικά ελάχιστο· ωστόσο, για εργασίες που απαιτούν έντονους πόρους, το bare metal μπορεί να υπερτερεί ελαφρώς λόγω της άμεσης πρόσβασης στο υλικό. Οι λειτουργικές πτυχές ευνοούν επίσης το containerization, προσφέροντας απλούστερες διαδικασίες ανάπτυξης, διαχείρισης και κλιμάκωσης σε σύγκριση με την πιο πολύπλοκη και απαιτητική φύση των ρυθμίσεων bare metal.

Επιπλέον, το containerization κερδίζει σε ευελιξία και φορητότητα, επιτρέποντας στις εφαρμογές να μετακινούνται εύκολα σε διάφορα περιβάλλοντα, σε αντίθεση με τις λύσεις bare metal. Η απόφαση μεταξύ containerization και bare metal για το Spark εξαρτάται από τις συγκεκριμένες απαιτήσεις χρήσης, με το containerization να ταιριάζει σε σενάρια cloud και multi-tenant για την κλιμακωσιμότητα και τη λειτουργική αποδοτικότητα, και το bare metal να είναι πιο κατάλληλο για σταθερές, υψηλής ζήτησης εργασίες που απαιτούν άμεση αλληλεπίδραση με το υλικό.

Συνοπτικά, οι παραπάνω συγκρίσεις έχουν θέσει μια στέρεη βάση για το επίκεντρο αυτής της έρευνας στη σύγκριση μεταξύ containerization και bare metal στο πλαίσιο του Apache Spark. Ενώ η ενότητα 4.3.1 ορίζει τα πλεονεκτήματα του containerization έναντι της εικονοποίησης, ιδιαίτερα όσον αφορά την αποδοτικότητα πόρων, την κλιμακωσιμότητα και τη λειτουργική απλότητα, η ενότητα 4.3.2 τονίζει τους συμβιβασμούς μεταξύ των αναπτύξεων containerization και bare metal. Είναι φανερό από αυτές τις συζητήσεις ότι το containerization, στα περισσότερα σενάρια, προσφέρει μια πιο ισορροπημένη προσέγγιση, υπερτερώντας της εικονοποίησης όσον αφορά τη χρήση πόρων και τη λειτουργική ακρίβεια. Αυτό μας οδηγεί στην άμεση σύγκριση μεταξύ containerization και bare metal, καθώς είναι πιο κατάλληλη για αυτή την έρευνα.

Προσέγγιση

Περίπτωση: πολλαπλές πιθανές διαμορφώσεις για το dashboard οπτικοποίησης σε μια εφαρμογή πολλαπλών κατόχων

Σε multi-tenant εφαρμογές, τα dashboard οπτικοποίησης πρέπει να είναι ανταποκριτικά και κλιμακωτά, επιτρέποντας στους χρήστες να αλληλεπιδρούν αποτελεσματικά με σύνθετα σύνολα δεδομένων. Αυτό απαιτεί μια λεπτή ισορροπία διατήρησης της ακεραιότητας των δεδομένων και της απομόνωσης της υπηρεσίας ενώ παρέχεται μια απρόσκοπτη εμπειρία σε διάφορες βάσεις χρηστών. Για να αντιμετωπιστεί αυτό, η έρευνα αυτή εξερευνά δύο υλοποιήσεις μοντέλων μηχανικής μάθησης: η μία σε ένα containerized περιβάλλον Apache Spark για την κλιμακωσιμότητα και την ευελιξία ανάπτυξης του, και η άλλη σε bare metal για τη μεγιστοποίηση της υπολογιστικής δύναμης εξαλείφοντας το επιπλέον κόστος του containerization.

Η αξιολόγηση αυτών των διαμορφώσεων περιλαμβάνει την ανάλυση της απόδοσης, της κλιμακωσιμότητας και της χρήσης πόρων, μαζί με τη διασφάλιση της απομόνωσης δεδομένων και της

ασφάλειας. Πρακτικές πτυχές όπως η εμπειρία χρήστη, η συντήρηση του συστήματος και η κοστολογική αποδοτικότητα είναι επίσης κρίσιμες, επηρεάζοντας την επιλογή μεταξύ της ευελιξίας μίας containerized διαμόρφωσης και τις δυνατότητες απόδοσης του bare metal. Τελικά, αυτή η απόφαση επηρεάζει σημαντικά την αποτελεσματικότητα των dashboard οπτικοποίησης σε multi-tenant εφαρμογές, καθοδηγώντας προς μια βέλτιστη διαμόρφωση που συνδυάζει απόδοση, κλιμακωσιμότητα και ικανοποίηση χρήστη σε περίπλοκα, προσανατολισμένα στα δεδομένα περιβάλλοντα.

Σύνολο δεδομένων

Το σύνολο δεδομένων «US-Accidents» από το Kaggle, που περιλαμβάνει περίπου 2.97 εκατομμύρια καταγραφές τροχαίων ατυχημάτων στις Ηνωμένες Πολιτείες από τον Φεβρουάριο του 2016, χρησιμεύει ως βάση για την αξιολόγηση της απόδοσης αλγορίθμων μηχανικής μάθησης στην ανίχνευση περιστατικών. Συγκεντρωμένα μέσω διαφόρων πηγών, συμπεριλαμβανομένων των τμημάτων μεταφορών των πολιτειών και των αισθητήρων κυκλοφορίας, αυτό το σύνολο δεδομένων προσφέρει λεπτομερείς πληροφορίες για τα ατυχημάτων, καταγεγραμμένες σε μορφή CSV με 45 χαρακτηριστικά όπως η σοβαρότητα του ατυχήματος, οι συντεταγμένες τοποθεσίας και οι καιρικές συνθήκες. Η εκτεταμένη κάλυψη του συνόλου δεδομένων σε 49 πολιτείες και τα διάφορα σενάρια ατυχημάτων παρέχουν μια στιβαρή βάση για ανάλυση, υποστηρίζοντας τον στόχο της μελέτης να αξιολογήσει διάφορες υπολογιστικές διαμορφώσεις στην ανίχνευση τροχαίων περιστατικών.

Για να επιτραπεί μια ενδελεχής συγκριτική ανάλυση, το σύνολο δεδομένων διαμερίζεται με προσοχή και εμπλουτίζεται με επιπλέον λεπτομέρειες όπως η κατεύθυνση της κυκλοφορίας, ο καιρός και τα σημεία ενδιαφέροντος, επικεντρώνοντας σε πόλεις με διάφορες συνθήκες κυκλοφορίας και καιρού για μια ισορροπημένη εξέταση. Καλύπτοντας την περίοδο από τον Ιούνιο έως τον Δεκέμβριο του 2018 για να ληφθούν υπόψη οι εποχιακές επιδράσεις, το σύνολο δεδομένων προετοιμάζεται με 113 χρονικά αμετάβλητα και πολυάριθμα χρονικά μεταβαλλόμενα χαρακτηριστικά για κάθε καταγραφή, χρησιμοποιώντας αρνητική δειγματοληψία για να αντιμετωπιστεί η ανισορροπία μεταξύ των περιπτώσεων ατυχημάτων και μη ατυχημάτων. Αυτή η στρατηγική προετοιμασία αποτελεί τη βάση για την εκπαίδευση και τη δοκιμή του πλαισίου πρόβλεψης ατυχημάτων, με στόχο να εξερευνήσει και να συγκρίνει διάφορες προσεγγίσεις υλοποίησης σε ένα κατανεμημένο υπολογιστικό περιβάλλον, διευκολύνοντας έτσι την ανάπτυξη προηγμένων, υψηλής απόδοσης dashboard για εφαρμογές υποστήριξης κυκλοφορίας δικτύου (Moosavi, Samavatian, Parthasarathy, Teodorescu, et al., 2019).

1.1-1: Λεπτομέρειες συνόλου δεδομένων με ατυχήματα

Κατηγορία	Χαρακτηριστικά
Χαρακτηριστικά κυκλοφορίας	id, source, TMC, severity, start_time, end_time, start_point, end_point, distance, description
Χαρακτηριστικά Διεύθυνσης	number, street, side (left/right), city, county, state, zip-code, country
Χαρακτηριστικά καιρού	time, temperature, wind_chill, humidity, pressure, visibility, wind_direction, wind_speed, precipitation, condition

Χαρακτηριστικά POI	Amenity, Bump, Crossing, Give_Way, Junction, _Exit, Railway, Roundabout, Station, Stop, Traffic_Calming, Traffic_Signal, Turning_Loop
Περίοδος ημέρας	Sunrise/Sunset, Civil Twilight, Nautical Twilight, Astronomical Twilight

Συνολικά Ατυχήματα	2,974,336
# Ατυχημάτων MapQuest	2,257,521 (75.89%)
# Ατυχημάτων Bing	684,097 (22.99%)
# Ατυχημάτων και από τα δύο	32,718 (1.1%)
Κορυφαίες Πολιτείες	California (485K), Texas (238K), Florida (177K), North Carolina (109K), New York (106K)

Προαπαιτούμενα

Πολλαπλοί αλγόριθμοι

Η ενσωμάτωση ενός ποικίλου σετ αλγορίθμων μηχανικής μάθησης, συμπεριλαμβανομένων του Random Forest, Multilayer Perceptron (MLP), Multiclass Logistic Regression, Decision Tree και Naive Bayes Classifier, είναι καθοριστική σε αυτήν την μελέτη για να κατανοήσουμε τις λεπτομέρειες απόδοσης διαφορετικών διαμορφώσεων κατανεμημένου υπολογισμού. Αυτή η μεθοδολογική επιλογή εξυπηρετεί στην ολιστική αξιολόγηση πώς διάφορες ρυθμίσεις υπολογισμού επηρεάζουν την απόδοση του αλγορίθμου και ενισχύει τη γενικευσιμότητα των ευρημάτων της έρευνας. Οι μοναδικές απαιτήσεις υπολογιστικής δύναμης και χρήσης πόρων κάθε αλγορίθμου προσφέρουν μια ευρεία προοπτική στις απαιτήσεις κατανεμημένης επεξεργασίας και μνήμης, δίνοντας μια ολοκληρωμένη αξιολόγηση των εργασιών ταξινόμησης και παλινδρόμησης σε διάφορα περιβάλλοντα. Συγκρίνοντας την απόδοση διαφορετικών αλγορίθμων, η μελέτη στοχεύει να αποκαλύψει συγκεκριμένα trade-offs και αναγκαίες βελτιστοποιήσεις μέσα σε διάφορες διαμορφώσεις υπολογισμού, παρέχοντας πληροφορίες στην αλληλεπίδραση μεταξύ εργασιών μηχανικής μάθησης και των υποδομών τους.

Cloud Native / φορητότητα

Η υλοποίηση μιας cloud-native και φορητής αρχιτεκτονικής για εφαρμογές Apache Spark είναι καθοριστική για τη διατήρηση της ευελιξίας και αποδοτικότητας σε διάφορες υπολογιστικές ρυθμίσεις. Οι δυνατότητες cloud-native, συμπεριλαμβανομένης της κλιμακωσιμότητας, ανθεκτικότητας και κατανεμημένης επεξεργασίας, επιτρέπουν σε αυτές τις εφαρμογές να εκμεταλλευτούν αποτελεσματικά λειτουργίες του cloud όπως η αυτόματη κλιμάκωση και η γρήγορη παροχή. Ταυτόχρονα, η φορητότητα διασφαλίζει ότι οι εφαρμογές μπορούν να λειτουργούν συνεπώς σε διάφορες υποδομές, από παρόχους cloud έως τοπικά κέντρα δεδομένων, μειώνοντας τους

κινδύνους lock-in του προμηθευτή. Πετυχαίνοντας αυτό, το containerization αναδεικνύεται ως μια βασική στρατηγική, χρησιμοποιώντας τεχνολογίες όπως το Docker για να ενθυλακώνει εφαρμογές και τις εξαρτήσεις τους, διευκολύνοντας τη συνεπή εκτέλεση σε οποιοδήποτε περιβάλλον υποστηρίζει container.

Επιπλέον, εργαλεία ορχήστρωσης όπως το Kubernetes παίζουν κρίσιμο ρόλο στη διαχείριση αυτών των containerized εφαρμογών, ενισχύοντας την κλιμακωσιμότητα και ανθεκτικότητα τους σε διάφορες ρυθμίσεις. Ωστόσο, προκλήσεις όπως η πολυπλοκότητα της ορχήστρωσης, η εξασφάλιση της ασφάλειας στα περιβάλλοντα, η διατήρηση της απόδοσης και η επίτευξη συνεπούς διαχείρισης δεδομένων υπογραμμίζουν την ανάγκη για προηγμένες στρατηγικές και προσεκτικό σχεδιασμό. Η υπέρβαση αυτών των εμποδίων είναι ουσιώδης για την αξιοποίηση του πλήρους δυναμικού μιας ανθεκτικής, cloud-native και φορητής αρχιτεκτονικής σε εφαρμογές βασισμένες στο Spark, απαιτώντας τόσο τεχνική όσο και στρατηγική επάρκεια.

Μη λειτουργικές απαιτήσεις / απαιτήσεις απόδοσης

Η διασφάλιση των μη-λειτουργικών απαιτήσεων και των απαιτήσεων απόδοσης των εφαρμογών που χρησιμοποιούν κατανεμημένο υπολογισμό με το Apache Spark είναι καθοριστική, και περιλαμβάνουν την κλιμακωσιμότητα (Choi et al., 2021) για την αντιμετώπιση μεταβαλλόμενων φορτίων, και την προσαρμοστικότητα για δυναμική κατανομή πόρων. Η υψηλή διαθεσιμότητα και αξιοπιστία του συστήματος είναι ουσιώδης για την ελαχιστοποίηση του χρόνου αδράνειας και τη διατήρηση συνεχούς, συνεπούς απόδοσης (Thiruvathukal et al., 2019). Η εφαρμογή πρέπει να παρέχει υψηλή ανταπόκριση και χαμηλή καθυστέρηση (Chang et al., 2016) στην επεξεργασία και οπτικοποίηση δεδομένων για να υποστηρίξει άρτιες εμπειρίες χρήστη, μαζί με την αποδοτικότητα πόρων για οικονομικά αποδοτική κλιμάκωση σε περιβάλλοντα cloud. Τα μέτρα ασφάλειας, συμπεριλαμβανομένης της κρυπτογράφησης δεδομένων και των ασφαλών ελέγχων πρόσβασης, είναι κρίσιμα για την προστασία από μη εξουσιοδοτημένη πρόσβαση και παραβιάσεις σε multi-tenancy πλαίσια (Neves & Bernardino, 2015). Η διαλειτουργικότητα σε διάφορα υπολογιστικά περιβάλλοντα (Lokuciejewski et al., 2021), η εύκολη συντήρηση του συστήματος, η ενημερωσιμότητα, και η συμμόρφωση με τα βιομηχανικά πρότυπα είναι κρίσιμα για τη μακροπρόθεσμη βιωσιμότητα και τη νόμιμη διαχείριση δεδομένων (Suneetha et al., 2020). Η ισχυρή παρακολούθηση, η καταγραφή για βελτιστοποίηση της απόδοσης, και ένα ολοκληρωμένο σχέδιο ανάκαμψης από καταστροφές (Nagar, 2017), συμπεριλαμβανομένων τακτικών αντιγράφων ασφαλείας και σαφών στρατηγικών αποκατάστασης, είναι απαραίτητα για τη διατήρηση της ακεραιότητας του συστήματος και της επιχειρησιακής αριστείας.

Πειραματική Ρύθμιση

Περιγραφή

Στο πείραμά μας, εκτελέσαμε τους πέντε επιλεγμένους αλγορίθμους σε ένα bare metal και τρεις διαμορφώσεις Kubernetes για να αξιολογήσουμε την απόδοση του κατανεμημένου υπολογισμού χρησιμοποιώντας το Apache Spark. Αυτές οι διαμορφώσεις είναι ουσιώδεις στην υποστήριξη της προσπάθειας μας να αναλύσουμε τις διαφορές και ομοιότητες, τα υπέρ και τα κατά των προσεγγίσεων containerization και bare metal στον τομέα του κατανεμημένου υπολογισμού.

Spark

Καθ' όλη τη διάρκεια των πειραμάτων μας, το Apache Spark λειτουργεί ως το κοινό νήμα, επιτρέποντάς μας να αξιολογήσουμε την απόδοση και τη συμπεριφορά του σε διαμορφώσεις containerization και bare metal. Εξετάζοντας πώς το Spark αλληλεπιδρά με καθένα από αυτά τα πλαίσια, στοχεύουμε να κατανοήσουμε τις ανταλλαγές, τα οφέλη και τις αδυναμίες που συνδέονται με κάθε προσέγγιση στον τομέα του κατανεμημένου υπολογισμού. (Salloum et al., 2016)

Μοντέλα μηχανικής μάθησης

Τα επιλεγμένα μοντέλα μηχανικής μάθησης έχουν επιλεγεί λόγω της σχετικότητάς τους με την ανίχνευση περιστατικών και την ποικιλία τους όσον αφορά τις κατηγορίες μηχανικής μάθησης. Περιλαμβάνουν μια ποικιλία τεχνικών μηχανικής μάθησης, επιτρέποντάς μας να αξιολογήσουμε πώς διάφορες διαμορφώσεις containerization και bare metal επηρεάζουν την απόδοσή τους.

Random Forest

Ο αλγόριθμος Random Forest, εκμεταλλευόμενος την κλάση RandomForestClassifier στο Apache Spark, ξεχωρίζει στην ανίχνευση περιστατικών χρησιμοποιώντας πολλαπλά δέντρα αποφάσεων για να χειριστεί αποτελεσματικά πολύπλοκα, μη ισορροπημένα σύνολα δεδομένων. Μειώνει το υπερπροσαρμογή και ενισχύει την ακρίβεια, καθιστώντας τον πολυμορφικό για διάφορες κατανομές δεδομένων. Η αποδεδειγμένη επιτυχία του στην ανίχνευση τροχαίων περιστατικών μέσω ανάλυσης παραγόντων και βαρυτικών προσεγγίσεων υπογραμμίζει την πρακτική του χρησιμότητα στη βελτίωση των αποτελεσμάτων ανίχνευσης (Jiang & Deng, 2020).

Multilayer Perceptron

Το Multilayer Perceptron (MLP), υλοποιημένο στο Apache Spark μέσω της κλάσης MultilayerPerceptronClassifier, είναι ένα νευρωνικό δίκτυο ικανό να μαθαίνει από μεγάλα σύνολα δεδομένων για την ανίχνευση περιστατικών. Ξεχωρίζει στην αναγνώριση περίπλοκων μοτίβων, αποδεικνύοντας τη χρησιμότητά του για εργασίες ταξινόμησης και παλινδρόμησης, ιδιαίτερα σε λεπτομερή κυκλοφοριακά σενάρια (Kongkhaensarn & Piantanakulchai, 2018).

Multiclass Logistic Regression

Ο Multiclass Logistic Regression, διαθέσιμο στο Apache Spark, επεκτείνει τον αλγόριθμο logistic regression για προβλέψεις πολλαπλών κλάσεων, ιδανικός για την ανίχνευση περιστατικών που περιλαμβάνουν πολλαπλούς τύπους περιστατικών. Η απλότητα και ερμηνευσιμότητά του, όπως φαίνεται σε αναφορές ασφάλειας, προσφέρουν πρακτικά πλεονεκτήματα για διάφορες εργασίες ανίχνευσης (Wang et al., 2017).

Decision Tree

Ο αλγόριθμος Decision Tree, που παρουσιάζεται στο Apache Spark, προσφέρει ένα απλό αλλά αποτελεσματικό μοντέλο για την ανίχνευση περιστατικών κατατάσσοντας τα δεδομένα με βάση τις

δοκιμές γνωρισμάτων. Η ερμηνευσιμότητα και η εφαρμογή του σε κυκλοφοριακά σενάρια υπογραμμίζουν τη χρησιμότητά του στην κατανόηση διαφόρων τύπων περιστατικών (Chen & Wang, 2009).

Nayve Bayes Classifier

Ο Naive Bayes Classifier, που υλοποιείται από την κλάση NaiveBayes του Apache Spark, είναι αποδοτικός για την ανίχνευση περιστατικών, ιδιαίτερα με κατηγορικά δεδομένα. Παρά την απλότητά του, είναι αποτελεσματικός στη γρήγορη επεξεργασία δεδομένων και έχει χρησιμοποιηθεί σε μεθόδους σύνθεσης περιστατικών κυκλοφορίας για βελτιωμένη ανίχνευση (Q. Liu et al., 2014).

Αρχιτεκτονικές

Bare metal – single PC

Στη διαμόρφωση Bare Metal με έναν κόμβο, χρησιμοποιείται ένας μόνο προσωπικός υπολογιστής ως ο υπολογιστικός κόμβος για το spark cluster. Αυτή η διάταξη αντιπροσωπεύει την παραδοσιακή προσέγγιση της εκτέλεσης του Spark απευθείας σε αφιερωμένο υλικό χωρίς κανένα επίπεδο αφαίρεσης. Τα κύρια χαρακτηριστικά αυτής της διάταξης περιλαμβάνουν:

- **Hardware**: Ένας μόνος υπολογιστής (PC) με αφιερωμένους πόρους CPU, RAM και αποθήκευσης.
- **Spark Cluster**: Ένα σύμπλεγμα Spark με έναν κόμβο, που λειτουργεί σε bare metal υλικό.
- **Λειτουργικό Σύστημα**: Ο υπολογιστικός κόμβος λειτουργεί απευθείας στο τοπικό λειτουργικό του σύστημα.

Συμπεριλαμβάνουμε αυτήν τη διαμόρφωση για να καθιερώσουμε μια βασική μέτρηση απόδοσης που αντιπροσωπεύει μια παραδοσιακή προσέγγιση χωρίς επίπεδα αφαίρεσης. Αυτό μας βοηθά να αξιολογήσουμε την ακατέργαστη, φυσική απόδοση του Apache Spark σε αφιερωμένο υλικό.

Kubernetes – spark cluster 1/2/4 κόμβων

Στη διαμόρφωση Kubernetes (K8s) με έναν, δύο και τέσσερις κόμβους Spark Cluster, χρησιμοποιήθηκε το Kubernetes, μια δημοφιλής πλατφόρμα ορχήστρωσης δοχείων, για τη διαχείριση ενός συστήματος Spark. Το Kubernetes παρέχει ένα υψηλό επίπεδο αφαίρεσης και απομόνωσης πόρων, καθιστώντας το μια κατάλληλη επιλογή για εργασίες που εκτελούνται σε φορτία εργασίας. Τα κύρια χαρακτηριστικά αυτής της διάταξης περιλαμβάνουν:

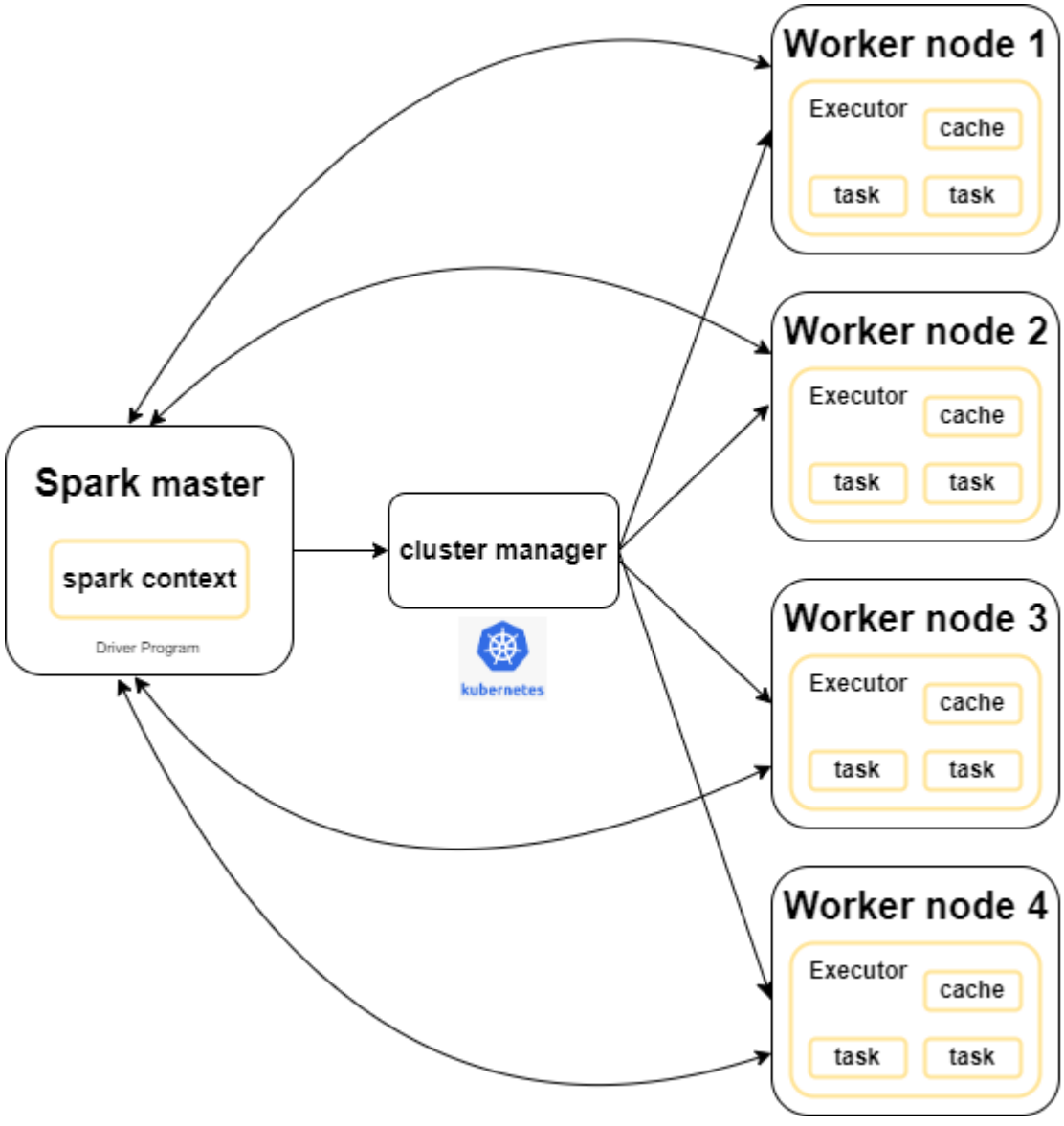
- **Kubernetes**: Ένα σύμπλεγμα Spark με έναν κόμβο που διαχειρίζεται από το Kubernetes.
- **Docker Containers**: Το Apache Spark και οι απαιτούμενες εξαρτήσεις του είναι δοχεία που διαχειρίζονται από το Docker και ορχηστρώνονται από το Kubernetes.
- **Διαχείριση Πόρων**: Το Kubernetes διαχειρίζεται δυναμικά τους πόρους CPU και μνήμης που κατανέμονται στα δοχεία Spark.

Για τις διαμορφώσεις δύο και τεσσάρων κόμβων:

- **Κλιμάκωση Κόμβων**: Οι εργασίες Spark μπορούν να κατανεμηθούν σε 2/4 κόμβους, βελτιώνοντας πιθανώς την κατανεμημένη εκτέλεση και απόδοση.

- **Διακόμβεια Επικοινωνία**: Οι κόμβοι Spark επικοινωνούν μέσω του δικτύου Kubernetes.

Καθώς η πολυπλοκότητα του περιβάλλοντος Kubernetes αυξάνεται, από τον έναν στους δύο και στη συνέχεια στους τέσσερις κόμβους, στοχεύουμε να αξιολογήσουμε πώς οι εφαρμογές Spark εκτελούνται όταν κατανέμονται σε πολλαπλούς κόμβους που διαχειρίζονται από το Kubernetes. Αυτή η διαμόρφωση μας βοηθά να κατανοήσουμε τα οφέλη της κλιμακωσιμότητας στην ορχήστρωσης των container.



1.1-4: Kubernetes - spark cluster τεσσάρων κόμβων: Ροή εκτέλεσης

Αποτελέσματα

Πίνακας με αλγορίθμους και αποδώσεις από τις διάφορες αρχιτεκτονικές

Τα KPIs που μετρήθηκαν ήταν ο χρόνος εκτέλεσης του training και του testing των μοντέλων μηχανικής μάθησης, όπως και η μνήμη που κατανάλωσαν.

1.1-2: Πίνακας αποτελεσμάτων

Αλγόριθμοι	Αρχιτεκτονική	Απόδοση		
		Χρόνος	Μνήμη (ανά κόμβο)	Μνήμη (συνολικά)
Random forest	Bare metal – ένας υπολογιστής	717.85 s	1190.64 MiB	1190.64 MiB
	K8s – spark cluster ενός κόμβου	729.86 s	1190.86 MiB	1191.18 MiB
	K8s – spark cluster δύο κόμβων	453.06 s	704.07 MiB	1191.51 MiB
	K8s – spark cluster τεσσάρων κόμβων	334.00 s	427.56 MiB	1192.03 MiB
MLP	Bare metal – ένας υπολογιστής	1237.64 s	1082.91 MiB	1082.91 MiB
	K8s – spark cluster ενός κόμβου	1309.26 s	1082.83 MiB	1083.46 MiB
	K8s – spark cluster δύο κόμβων	896.01 s	607.97 MiB	1085.14 MiB
	K8s – spark cluster τεσσάρων κόμβων	574.47 s	419.85 MiB	1085.63 MiB
Multiclass Logistic regression	Bare metal – ένας υπολογιστής	126.42 s	438.85 MiB	438.85 MiB
	K8s – spark cluster ενός κόμβου	137.73 s	438.67 MiB	438.78 MiB
	K8s – spark cluster δύο κόμβων	88.40 s	259.61 MiB	439.49 MiB
	K8s – spark cluster τεσσάρων κόμβων	62.89 s	169.90 MiB	440.29 MiB

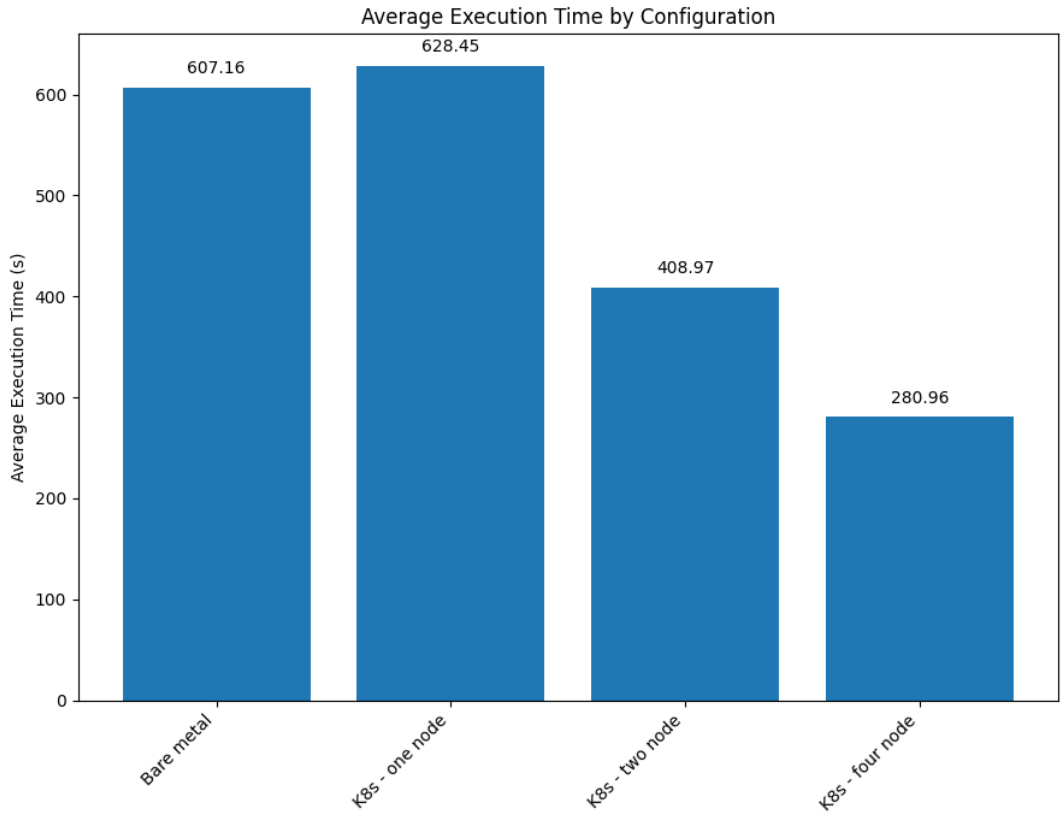
Decision tree	Bare metal – ένας υπολογιστής	542.44 s	1010.77 MiB	1010.77 MiB
	K8s – spark cluster ενός κόμβου	549.47 s	1011.34 MiB	1011.47 MiB
	K8s – spark cluster δύο κόμβων	344.128 s	598.15 MiB	1012.13 MiB
	K8s – spark cluster τεσσάρων κόμβων	249.46 s	391.46 MiB	1012.75 MiB
Naive Bayes	Bare metal – ένας υπολογιστής	411.47 s	762.99 MiB	762.99 MiB
	K8s – spark cluster ενός κόμβου	415.95 s	761.92 MiB	762.27 MiB
	K8s – spark cluster δύο κόμβων	263.25 s	451.05 MiB	763.36 MiB
	K8s – spark cluster τεσσάρων κόμβων	183.96 s	295.20 MiB	763.42 MiB

Ανάλυση και ερμηνεία

Σύγκριση απόδοσης

Αναλύοντας την απόδοση διαφόρων αλγορίθμων κατά την μετάβαση από ένα μεμονωμένο υπολογιστή σε ένα cluster τεσσάρων κόμβων Kubernetes (K8s), παρατηρείται σημαντική μείωση στον χρόνο εκτέλεσης. Αυτή η βελτίωση της απόδοσης είναι συνεπής στα διάφορα μοντέλα, με το εύρος της μείωσης στον χρόνο εκτέλεσης να κυμαίνεται από 50.2% έως περίπου 54.0%. Αυτή η σημαντική μείωση στον χρόνο δείχνει τα κέρδη απόδοσης που επιτυγχάνονται μέσω των διατάξεων κατανεμημένου υπολογισμού, ιδιαίτερα σε περιβάλλοντα πολλαπλών κόμβων. Αυτές οι βελτιώσεις δεν υποδηλώνουν μόνο τη δύναμη της κατανεμημένης επεξεργασίας αλλά επίσης τονίζουν την αποτελεσματικότητα των Kubernetes clusters στην βελτιστοποίηση υπολογιστικών εργασιών μέσω διαφόρων μοντέλων.

Αντιθέτως, κατά την μετάβαση από ένα μεμονωμένο υπολογιστή σε μία διάταξη ενός κόμβου K8s, παρατηρείται μια ήπια αύξηση στον χρόνο εκτέλεσης. Αυτή η αύξηση κυμαίνεται από περίπου 1,67% έως 8%, ανάλογα με τον συγκεκριμένο αλγόριθμο που αναλύεται. Αυτή η ήπια αύξηση στον χρόνο εκτέλεσης μπορεί να αποδοθεί στο overhead που εισάγεται από τις διαδικασίες containerization που είναι ουσιώδεις στη διάταξη του ενός κόμβου K8s. Αν και αυτή η αύξηση είναι σχετικά μικρή, τονίζει τις πολυπλοκότητες και τις πιθανές αναποτελεσματικότητες που μπορούν να προκύψουν κατά την μετάβαση σε ένα περιβάλλον κατανεμημένου υπολογισμού χωρίς να αξιοποιηθούν πλήρως οι δυνατότητες των διαμορφώσεων πολλαπλών κόμβων.



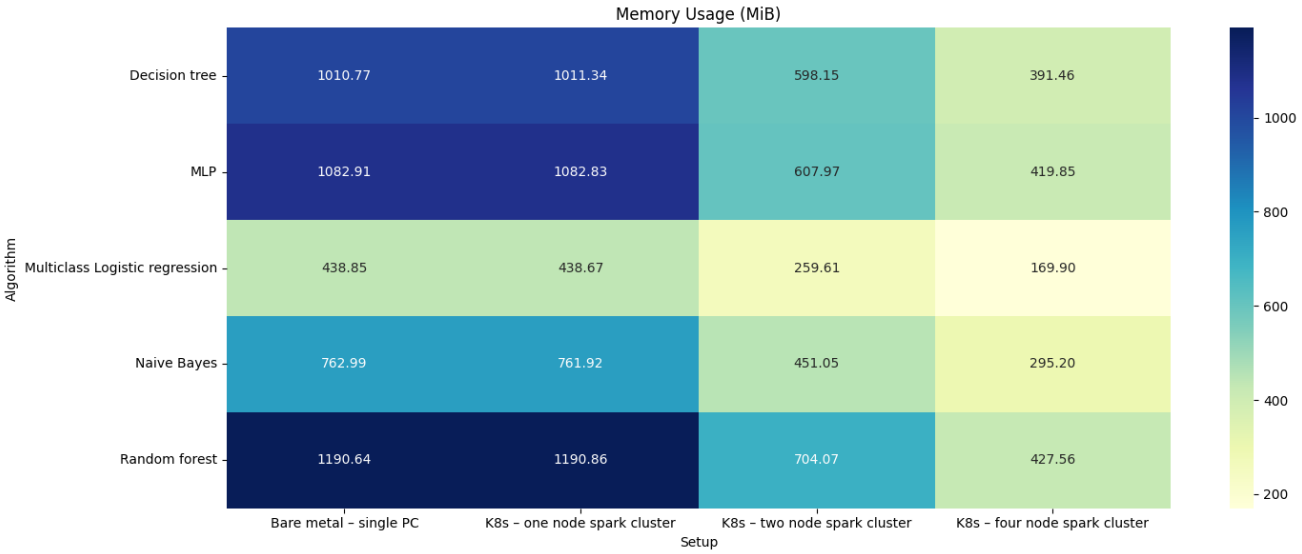
1.1-2: Μέσος χρόνος εκτέλεσης των μοντέλων μηχανικής μάθησης στις τέσσερις διαμορφώσεις

Σχετικά με τη χρήση μνήμης, μια συνεπής και σημαντική μείωση είναι εμφανής στη σύγκριση των διατάξεων bare metal με τις διαμορφώσεις τεσσάρων κόμβων K8s. Η χρήση μνήμης μειώνεται σημαντικά, δείχνοντας την ανώτερη αποδοτικότητα μνήμης των διατάξεων κατανεμημένου υπολογισμού. Αυτή η μείωση στην κατανάλωση μνήμης όχι μόνο αντανακλά τη βελτιωμένη ικανότητα των Kubernetes clusters να διαχειρίζονται τους πόρους πιο αποτελεσματικά αλλά επίσης τονίζει τα πλεονεκτήματα της κατανομής υπολογιστικών φορτίων σε πολλαπλούς κόμβους. Η μείωση στη χρήση μνήμης στους αλγορίθμους περαιτέρω υποστηρίζει την κατεύθυνση προς διανεμημένα περιβάλλοντα υπολογισμού, ιδιαίτερα για εφαρμογές όπου η αποδοτική χρήση πόρων είναι κρίσιμη. Αυτά τα ευρήματα τονίζουν τα οφέλη των βασισμένων σε Kubernetes διαμορφώσεων πολλαπλών κόμβων στην επίτευξη βέλτιστης απόδοσης και αποδοτικότητας πόρων.

Ερμηνεία και συνέπειες

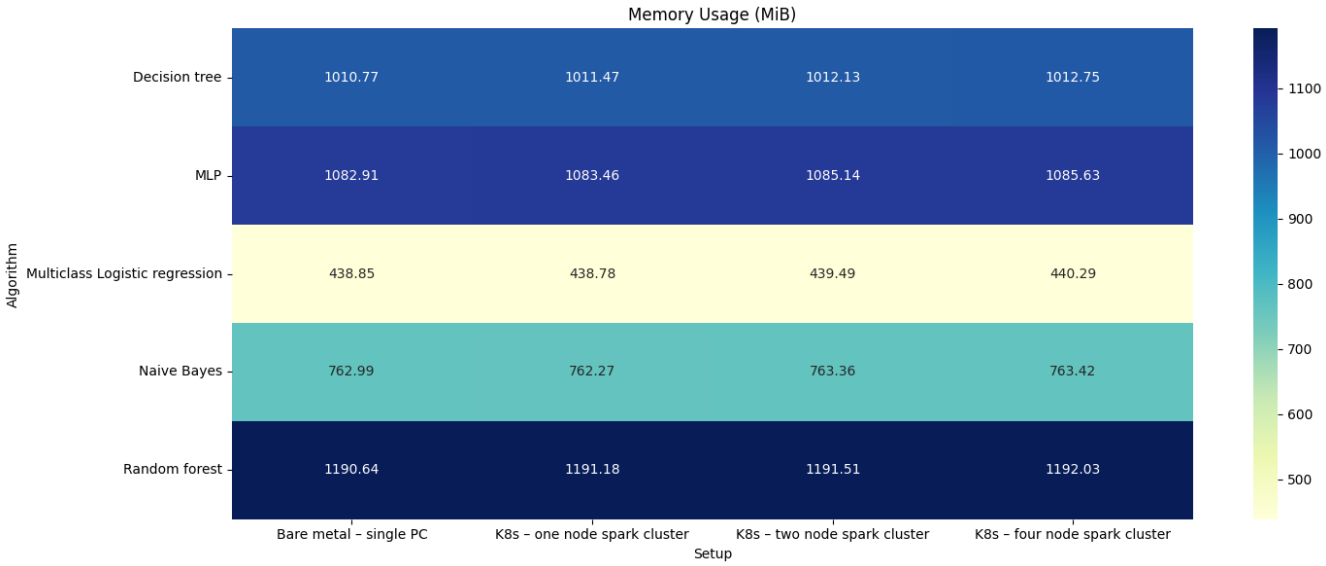
Μια παρατήρηση από την ανάλυσή μας είναι η σημαντική πτώση στη χρήση μνήμης σε όλους τους αλγορίθμους κατά τη χρήση μιας διαμόρφωσης πολλαπλών κόμβων Kubernetes (K8s) σε σύγκριση με μια διάταξη μεμονωμένου υπολογιστή, όπως απεικονίζεται στο heatmap στο σχήμα 1.1-5. Αυτή η μείωση στη χρήση μνήμης αποδίδεται στις αρχές του κατανεμημένου υπολογισμού, όπου οι εργασίες διαιρούνται μεταξύ πολλαπλών κόμβων, επιτρέποντας πιο αποδοτική χρήση της μνήμης. Η κατανεμημένη επεξεργασία σε clusters επιτρέπει την ταυτόχρονη εκτέλεση εργασιών, βελτιώνοντας την ταχύτητα υπολογισμού και τη χρήση πόρων. Ως αποτέλεσμα, κάθε κόμβος επεξεργάζεται μικρότερο μέρος των δεδομένων, μειώνοντας τη συνολική κατανάλωση μνήμης. Προηγμένες τεχνικές διαχείρισης μνήμης, όπως η έξυπνη προσωρινή αποθήκευση δεδομένων και η ισορροπία φορτίου, συμβάλλουν περαιτέρω σε αυτή την αποδοτικότητα. Η αποδοτικότητα είναι ιδιαίτερα αξιοσημείωτη

στις διαμορφώσεις δύο κόμβων και τεσσάρων κόμβων K8s Spark cluster, οι οποίες συνεπώς εκθέτουν χαμηλότερη χρήση μνήμης από τις διαμορφώσεις ενός κόμβου και μεμονωμένου υπολογιστή.



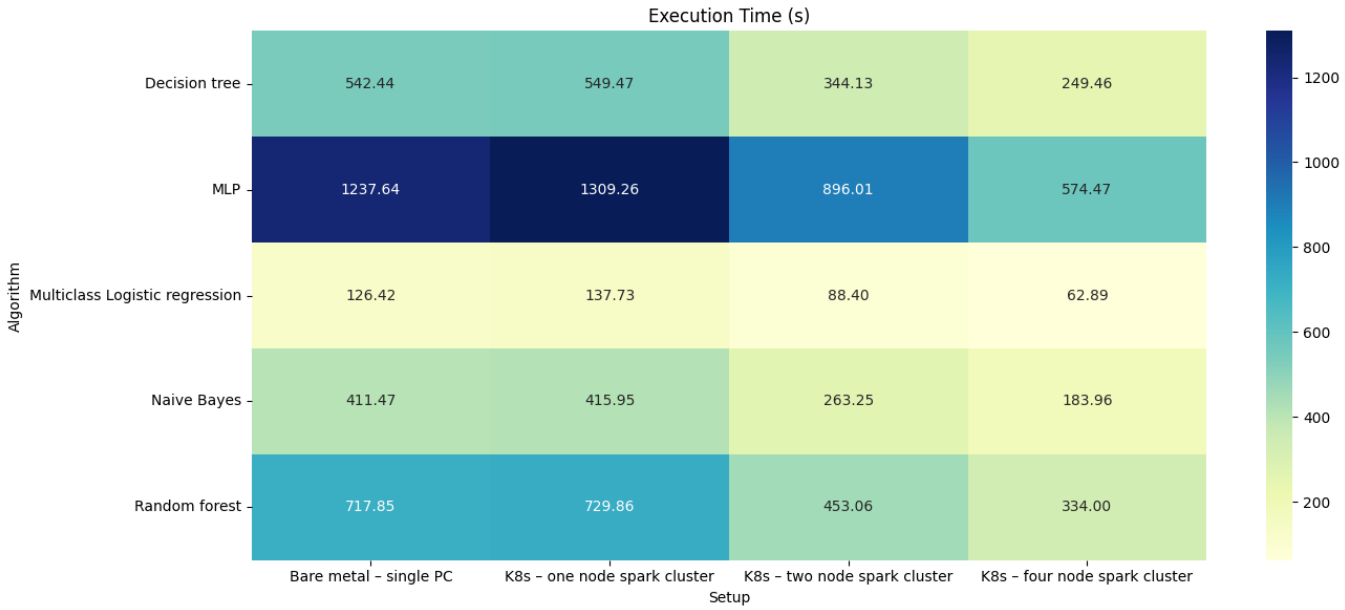
1.1-5: Heatmap χρήσης μνήμης ανά κόμβο

Επιπλέον, η στήλη "συνολική μνήμη" στον πίνακα αποτελεσμάτων μας δείχνει ότι η συνολική χρήση μνήμης παραμένει σταθερή σε διαφορετικές διαμορφώσεις, από έναν έως τέσσερις κόμβους, υποδηλώνοντας ότι η προσέγγιση κατανεμημένου υπολογισμού του K8s δεν εισάγει επιπλέον overhead μνήμης. Αυτή η σταθερότητα δείχνει την ικανότητα του K8s για οριζόντια κλιμάκωση χωρίς να προστίθενται επιπλέον κόστη μνήμης, που είναι κρίσιμη για τη διαχείριση μεγάλης κλίμακας δεδομένων ή ροών εργασίας μηχανικής μάθησης. Η συνεπής χρήση μνήμης σε διάφορες διαμορφώσεις τονίζει την αποδοτικότητα του K8s στη διαχείριση πόρων, υποστηρίζοντας τη χρήση του για εφαρμογές κλιμακωτής υψηλής απόδοσης χωρίς αυξημένη κατανάλωση πόρων. Αυτή η βελτιστοποίηση είναι κρίσιμη σε τομείς όπως το big data και η μηχανική μάθηση, όπου η αποτελεσματική διαχείριση μνήμης μπορεί να ενισχύσει σημαντικά την απόδοση και να διευκολύνει την επεξεργασία μεγαλύτερων συνόλων δεδομένων πιο αποτελεσματικά.



1.1-3: Heatmap συνολικής χρήσης μνήμης

Η παρατηρούμενη μείωση στον χρόνο εκτέλεσης στις διαμορφώσεις πολλαπλών κόμβων K8s σηματοδοτεί επίσης ένα σημαντικό βήμα προόδου στην αποδοτικότητα και την κλιμακωσιμότητα των υπολογιστικών διαδικασιών. Αυτή η μείωση επιτρέπει την ολοκλήρωση πιο πολύπλοκων υπολογισμών σε σύντομες περιόδους, τονίζοντας τον ρόλο του Kubernetes στην βελτιστοποίηση των υπολογιστικών πόρων. Παρά την ήπια αρχική αύξηση στον χρόνο εκτέλεσης κατά τη μετάβαση από μεμονωμένο υπολογιστή σε μια διάταξη ενός κόμβου K8s—λόγω των χρόνων εκκίνησης container και του επιπέδου αφαίρεσης που εισάγει το Kubernetes—τα οφέλη των μειωμένων χρόνων εκτέλεσης σε μεγαλύτερες διαμορφώσεις είναι σαφή. Αυτά τα οφέλη αντανακλούν την κλιμακωσιμότητα και την αποδοτικότητα του Kubernetes, επιτρέποντας τη δυναμική ανάθεση πόρων για να ανταποκριθούν σε αυξανόμενες υπολογιστικές απαιτήσεις χωρίς να παρατείνουν τους χρόνους εκτέλεσης. Αυτή η κλιμακωσιμότητα διασφαλίζει την οικονομικά αποδοτική διαχείριση των υπολογιστικών πόρων, ισορροπώντας την ανάθεση πόρων με τις απαιτήσεις φορτίου για να αποφευχθεί η υποχρησιμοποίηση ή η υπερπροβολή.



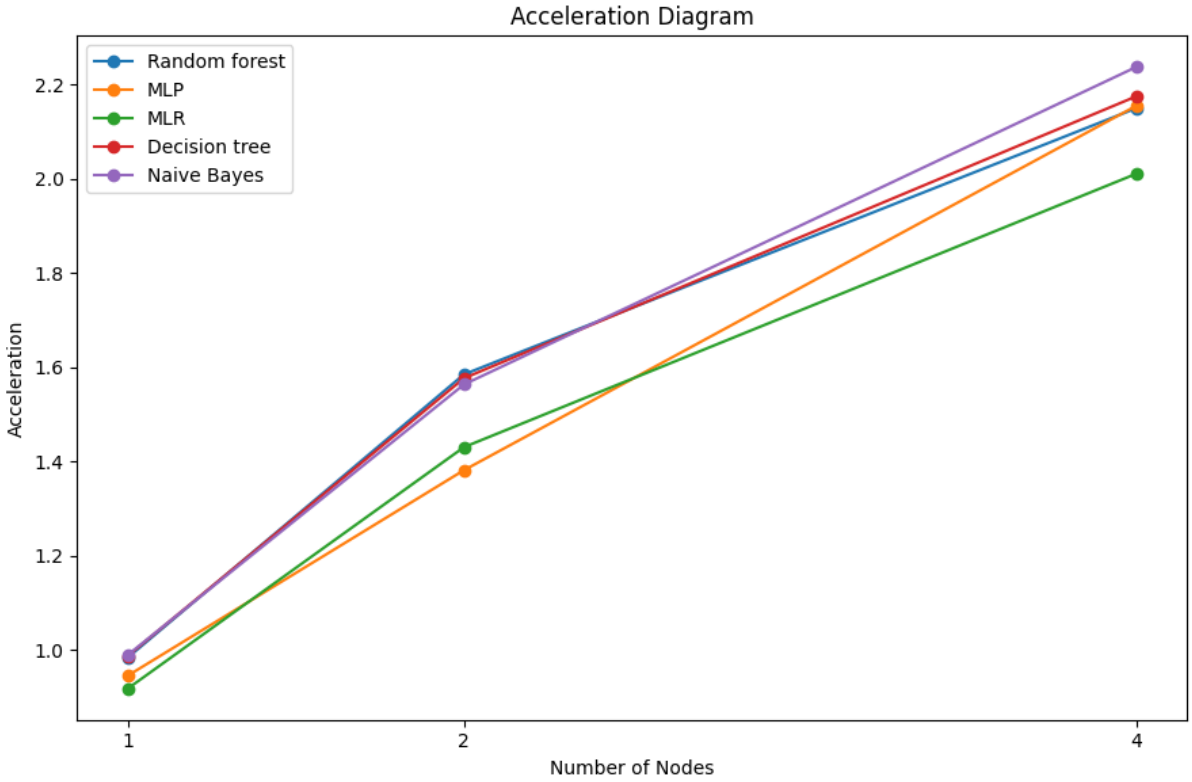
1.1-6: Heatmap χρόνου εκτέλεσης

Σύγκριση με βάση τον τύπο των μοντέλων μηχανικής μάθησης

Στο παρακάτω διάγραμμα επιτάχυνσης μπορούμε να δούμε ότι στο σύμπλεγμα 4 κόμβων Kubernetes έχουμε επιτάχυνση από περίπου 2 έως 2,2 ανάλογα με το μοντέλο. Ο χρόνος δεν υποτετραπλασιάζεται γιατί η αύξηση του αριθμού των κόμβων συνεπάγεται κόστος συντονισμού και επικοινωνίας. Αυτό είναι γνωστό ως νόμος του Amdahl, ο οποίος δηλώνει ότι η μέγιστη επιτάχυνση ενός προγράμματος περιορίζεται από το χρόνο εκτέλεσης του μη κατανεμημένου τμήματος του κώδικα. Πιο συγκεκριμένα, ακόμα κι αν προσθέσουμε περισσότερους κόμβους, ορισμένες εργασίες πρέπει να εκτελούνται σειριακά και έτσι υπάρχει ένα όριο στην επιτάχυνση που μπορεί να ολοκληρωθεί. Πέρα από αυτό, τα γενικά έξοδα από την ενορχήστρωση των κοντέινερ αυξάνουν περαιτέρω τον περιορισμό της κατανεμημένης εκτέλεσης.

Οι παρατηρήσιμες διακυμάνσεις στην επιτάχυνση των διάφορων αλγορίθμων μηχανικής μάθησης μπορούν να αποδοθούν στα μοναδικά χαρακτηριστικά και υπολογιστικές απαιτήσεις του καθενός. Ο αλγόριθμος Decision Tree, είναι λιγότερο ευαίσθητος στην ορχήστρωση, παρουσιάζοντας αισθητή επιτάχυνση στην εκτέλεση. Παρομοίως, ο Naive Bayes, ο οποίος είναι υπολογιστικά λιγότερο απαιτητικός, δείχνει επίσης σημαντική βελτίωση. Αντιθέτως, ο Multilayer Perceptron και η Multiclass

Logistic Regression, που έχουν υψηλότερες υπολογιστικές απαιτήσεις και βασίζονται σε περίπλοκες επαναληπτικές διαδικασίες, εμφανίζουν μικρότερη επιτάχυνση, με την τελευταία να είναι η λιγότερο ευνοημένη από την κλιμάκωση των κόμβων.



1.1-7: Διάγραμμα επιτάχυνσης

Συζήτηση

Απόδοση

Η μελέτη μας δείχνει τη βελτιωμένη απόδοση των clusters πολλαπλών κόμβων του Kubernetes έναντι των παραδοσιακών ρυθμίσεων Bare Metal στην εκτέλεση διαφόρων μοντέλων μηχανικής μάθησης με το Apache Spark. Συγκεκριμένα, παρατηρήσαμε μια σημαντική μείωση στον χρόνο εκτέλεσης σε όλους τους αλγορίθμους κατά τη μετάβαση από Bare Metal σε συστάδες Kubernetes σε πολλαπλές διαμορφώσεις κόμβων. Για παράδειγμα, αλγόριθμοι όπως ο Multilayer Perceptron και ο Decision Tree παρατήρησαν μειώσεις χρόνου εκτέλεσης πάνω από 50% σε μια διαμόρφωση τεσσάρων κόμβων Kubernetes. Αυτή η βελτίωση οφείλεται στις δυνατότητες κατανεμημένης επεξεργασίας του Kubernetes, που επιτρέπει την ταυτόχρονη εκτέλεση εργασιών σε πολλαπλούς κόμβους και βελτιστοποιεί την υπολογιστική αποδοτικότητα. Ωστόσο, η μετάβαση σε μια διαμόρφωση ενός κόμβου Kubernetes από Bare Metal δείχνει μια οριακή μείωση στην ταχύτητα εκτέλεσης, με αυξήσεις που κυμαίνονται από 1,6% έως 8,95%.

Επιπλέον, τα clusters του Kubernetes επιδεικνύουν ανώτερη αποδοτικότητα μνήμης, μειώνοντας συνεχώς τη χρήση μνήμης κατά τη μετάβαση από Bare Metal σε πολλαπλές διαμορφώσεις κόμβων Kubernetes. Αυτό υποδηλώνει την αποτελεσματική διαχείριση και κατανομή των πόρων μνήμης του

Kubernetes μέσω της τεχνολογίας της δυναμικής διαχείρισης πόρων. Η κλιμακωσιμότητα και η βελτιστοποίηση πόρων που προσφέρουν οι συστάδες πολλαπλών κόμβων του Kubernetes είναι ιδιαίτερα πλεονεκτικές για εργασίες υψηλής απόδοσης υπολογιστικών εργασιών και εφαρμογές που βασίζονται σε μεγάλους όγκους δεδομένων, τονίζοντας την ικανότητα του Kubernetes να διανέμει αποτελεσματικά τα φορτία εργασίας σε πολλαπλούς κόμβους. Παρόλο που η μελέτη επιβεβαιώνει την ικανότητα του Kubernetes να βελτιστοποιεί τη χρήση πόρων και να ενισχύει την απόδοση του συστήματος, ιδιαίτερα στη διαχείριση μνήμης, τονίζει επίσης τη σημασία της επιλογής της κατάλληλης διαμόρφωσης και της κατανόησης των συγκεκριμένων απαιτήσεων της εφαρμογής για να αξιοποιηθεί πλήρως το Kubernetes.

Απόδοση – Φορητότητα

Το Kubernetes, που είναι γνωστό για τη φορητότητα και την κλιμακωσιμότητά του στη διαχείριση εμπλεκομένων εφαρμογών, εισάγει σημαντικά πλεονεκτήματα, ιδιαίτερα για εφαρμογές που βασίζονται στο cloud, ενισχύοντας την ευκολία με την οποία οι εφαρμογές μπορούν να μετακινηθούν σε διάφορα υπολογιστικά περιβάλλοντα. Αυτή η φορητότητα, αποτέλεσμα του containerization, διασφαλίζει ότι οι εφαρμογές λειτουργούν συνεπώς σε οποιαδήποτε υποδομή, είτε πρόκειται για διαφορετικούς παρόχους cloud είτε για μεταβάσεις μεταξύ τοπικών και cloud ρυθμίσεων. Ωστόσο, αυτή η ευελιξία μπορεί να έρθει με κόστος απόδοσης. Το containerization απομακρύνει τις εφαρμογές από το υλικό, δημιουργώντας επιβάρυνση που θα μπορούσε να επηρεάσει την ταχύτητα και την αποδοτικότητα. Συγκεκριμένα, στις διαμορφώσεις ενός κόμβου του Kubernetes, η ευκολία κλιμάκωσης και ανάπτυξης αυξάνει ελαφρώς τους χρόνους εκτέλεσης και τις ανάγκες πόρων, παρουσιάζοντας έναν σημαντικό trade-off στον υπολογισμό υψηλής απόδοσης όπου η ταχύτητα είναι κρίσιμη.

Η απόφαση μεταξύ Kubernetes και bare metal επηρεάζεται από τις μοναδικές απαιτήσεις μιας εφαρμογής. Εάν η κλιμακωσιμότητα και η ευκολία ανάπτυξης υπερτερούν της ακατέργαστης απόδοσης, το Kubernetes είναι προτιμητέο, παρά την πολυπλοκότητα διαχείρισης που φέρνει, συμπεριλαμβανομένης της εξειδικευμένης γνώσης στην ορχήστρωση των container. Αντιθέτως, το bare metal μπορεί να ταιριάζει καλύτερα σε εφαρμογές με σταθερές απαιτήσεις απόδοσης. Αυτή η απόφαση επηρεάζει όχι μόνο την άμεση απόδοση αλλά και τη μακροπρόθεσμη οργανωτική στρατηγική και βιωσιμότητα, ισορροπώντας τις πιθανές οικονομίες κόστους και τη μειωμένη περιβαλλοντική επιβάρυνση του Kubernetes έναντι των άμεσων

Γενικά έξοδα διαχείρισης

Το Kubernetes, γνωστό για την ενίσχυση της κλιμακωσιμότητας και της φορητότητας, συμπεριλαμβάνει μια πολυπλοκότητα και έναν διοικητικό φόρτο που μπορεί να επηρεάσει το κόστος, την αποδοτικότητα και την πρακτική εφαρμογή του. Η πολυπλοκότητα προκύπτει από τον συντονισμό των containers, τη διαχείριση των πόρων του cluster, τη ρύθμιση δικτύων και την εξασφάλιση υψηλής διαθεσιμότητας και ανοχής σε σφάλματα. Η αποτελεσματική διαχείριση του Kubernetes απαιτεί εξειδικευμένες δεξιότητες για να κατανοηθούν και να πλοηγηθούν οι περίπλοκες σχέσεις μεταξύ αυτών των συστατικών. Αυτή η πολυπλοκότητα αποτελεί πρόκληση για τους διαχειριστές, ειδικά στη δυναμική κατανομή πόρων και την παρακολούθηση, απαιτώντας συνεχή επιτήρηση για την βελτιστοποίηση της επίδοσης, τη διαχείριση του προγραμματισμού των pods και την εξασφάλιση της υγείας του cluster.

Η διαχείριση των περιβαλλόντων Kubernetes επεκτείνεται στην εξασφάλιση υψηλής διαθεσιμότητας, αποτελεσματικής ισορροπίας φορτίου και πολυπλοκότητας της ρύθμισης δικτύου. Το Kubernetes παρέχει μηχανισμούς για την ισορροπία φορτίων και τη διατήρηση της διαθεσιμότητας της εφαρμογής, ακόμη και κατά τη διάρκεια αποτυχιών κόμβων. Ωστόσο, η ρύθμιση και η διαχείριση αυτών των χαρακτηριστικών απαιτεί προσεκτικό σχεδιασμό και συνεχή επιτήρηση. Επιπλέον, η ασφάλιση της επικοινωνίας μεταξύ των pods, η διαχείριση των ελέγχων πρόσβασης και η συμμόρφωση με τα πρότυπα ασφάλειας δεδομένων εισάγουν περαιτέρω πολυπλοκότητα, απαιτώντας υψηλότερο επίπεδο εμπειρίας στην ορχήστρωση containers, τη διαχείριση δικτύων και την ασφάλεια. Αυτή η περιπλοκότητα απαιτεί σημαντική επένδυση σε εκπαίδευση ή πρόσληψη ειδικών, προσθέτοντας στον διοικητικό φόρτο.

Η απόφαση να υιοθετηθεί το Kubernetes αντί του Bare Metal δεν βασίζεται μόνο στην απόδοση και τη φορητότητα, αλλά και στην ικανότητα μιας οργάνωσης να διαχειριστεί και να συντηρήσει μια τέτοια υποδομή αποτελεσματικά. Ενώ το Kubernetes προσφέρει λειτουργικά οφέλη, ο διοικητικός φόρτος και τα συναφή κόστη - που περιλαμβάνουν εξειδικευμένο προσωπικό, εκπαίδευση, επιλογή εργαλείων και ένταξη, καθώς και τακτικές ενημερώσεις συστήματος - επηρεάζουν το συνολικό κόστος κατοχής. Οι οργανισμοί πρέπει να αξιολογήσουν αυτά τα κόστη έναντι των δυνητικών μακροπρόθεσμων αποδοτικοτήτων και στρατηγικών πλεονεκτημάτων μιας κλιμακώσιμης, ανθεκτικής πλατφόρμας διανεμημένων υπολογισμών. Αυτή η αξιολόγηση πρέπει να λαμβάνει υπόψη τη δυνατότητα διαχείρισης της οργάνωσης και το συνολικό κόστος κτήσης για να καθορίσει την καταλληλότητα του Kubernetes για τις συγκεκριμένες εφαρμογές τους, διασφαλίζοντας ότι η επιλογή συνάδει με τις ανάγκες της εφαρμογής και τους στρατηγικούς στόχους της οργάνωσης.

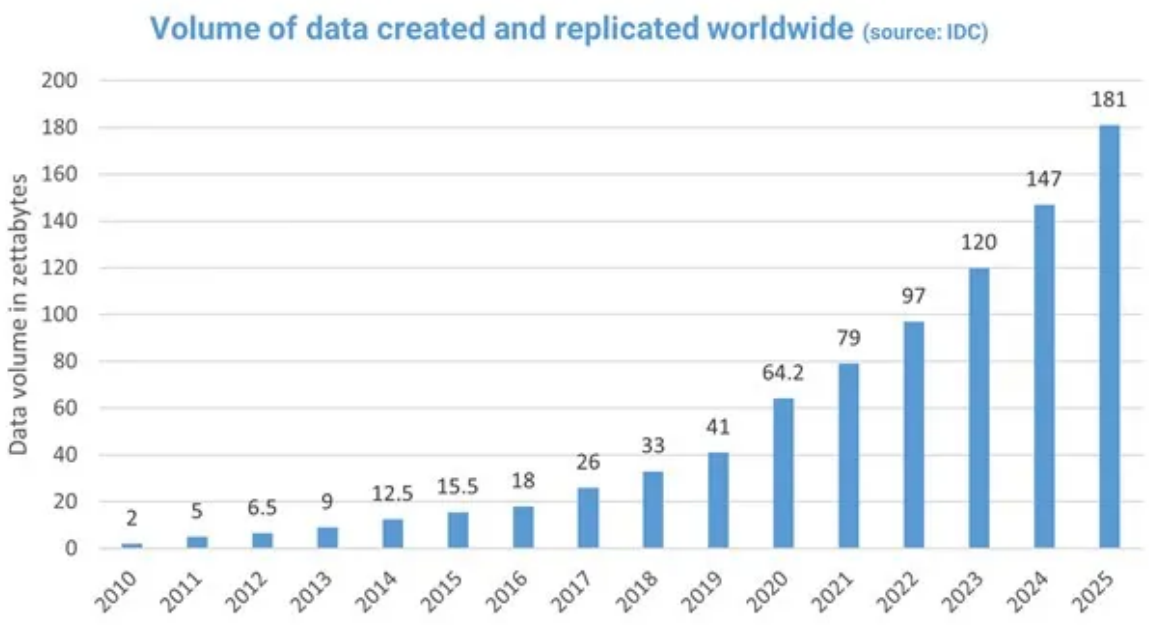
1 Introduction

In an era marked by exponential data growth, the ability to efficiently process and analyze large datasets is crucial for technological innovation. This thesis explores the field of distributed computing, focusing on Apache Spark, a leading engine for big data processing. Specifically, it examines Spark's performance in two operational environments: containerized setups and bare metal configurations.

The motivation for this study stems from the evolving landscape of distributed computing and the necessity for organizations to make informed decisions regarding their infrastructure. With businesses and researchers facing the challenge of handling large-scale data, particularly in the context of incident detection, the deployment architecture of data processing platforms like Apache Spark becomes critical. This thesis critically examines how containerization and bare metal setups impact the performance, scalability, and efficiency of distributed computing systems, with a special focus on their application in incident detection scenarios.

The objective is to provide a detailed comparative analysis of these two deployment strategies, elucidating their merits and limitations. This analysis is intended to aid in determining the most suitable architectural choice for specific computing needs, particularly in environments where efficient incident detection is paramount. This inquiry is not just academically significant but also vitally important for industry practitioners managing these technologies.

The aim is to offer valuable insights for navigating the complexities of big data processing, guiding decisions in a rapidly evolving technological landscape. The subsequent chapters explore the performance implications of containerized versus bare metal deployments in distributed computing using Apache Spark.



[1]

1.1-1: Bar chart showing the volume of data created and replicated worldwide.

[1] https://medium.com/@mwaliph/exponential-growth-of-data-2f53df89124

1.1 Subject

The subject of this thesis is to evaluate the performance of Apache Spark when deployed in containerized environments as opposed to bare metal setups, particularly in the context of incident detection. These two deployment architectures represent divergent methodologies in managing large-scale, data-intensive applications, each with distinct benefits and limitations.

Containerization, facilitated by tools like Docker and orchestrated systems like Kubernetes, is a lightweight and scalable method to package and run applications, offering high portability and operational efficiency. Bare metal deployment, which involves installing applications directly onto hardware, is traditionally preferred for its direct access to hardware resources, enhancing performance by eliminating virtualization overheads.

This thesis seeks to empirically and theoretically assess the performance differences between containerized and bare metal deployments of Apache Spark. This assessment includes examining resource utilization, data processing speed, and system management complexity. The goal is to inform about the trade-offs between these strategies, aiding in infrastructure decision-making in the context of distributed computing.

The research will explore various configurations of Spark deployment, from single-node to multi-node clusters, to comprehensively analyze how different strategies influence the performance of distributed computing environments, with a particular focus on Apache Spark.

1.2 Structure

The thesis is organized into nine chapters, each focusing on a specific aspect of the comparative study between containerization and bare metal environments in the context of distributed computing performance using Apache Spark. The structure is designed to provide a comprehensive understanding of the subject, starting from basic concepts, and gradually moving towards detailed experimental analysis and conclusions.

It begins with the 'Introduction', setting the stage for the study and highlighting its relevance in today's technological landscape. Here, readers are acquainted with the thesis's objectives and the significance of the research within the realm of distributed computing. Next, 'Theoretical Background' forms the second chapter which delves into essential topics like distributed computing, Apache Spark, containerization, virtualization, and bare metal, showing the merits and demerits of containerization against virtualization and Kubernetes against bare metal. It further enriches the reader's understanding with a survey of related studies.

Moving to the third chapter, 'Motivation', discussions illuminate the escalating importance of efficient distributed computing and the imperative for comprehensive studies comparing various architectures. 'Distributed Processing and Machine Learning', the fourth chapter, ventures into the methodologies for distributed processing, emphasizing the role of Kubernetes, particularly in conjunction with Spark.

In the fifth chapter, 'Approach', the thesis takes a more practical orientation. It outlines the methodologies employed in the study, detailing the case study, dataset, and specific requirements, including an array of algorithms and performance considerations. 'Experimental Setup', chapter six, offers a thorough breakdown of the experimental framework, on the Spark configuration, the algorithms deployed, and the variety of setups examined, ranging from a single PC bare metal to diverse Kubernetes cluster configurations.

Chapter seven, 'Results', is where the outcomes of the experiments are methodically presented. This includes performances of algorithms across different setups, accompanied by an analysis and interpretation of these findings. 'Discussion', the eighth chapter, dives into the implications of these results, discussing aspects such as performance, the balance between performance and portability, and the management overhead associated with each setup.

Concluding the thesis, the 'Conclusions' chapter there is a summary of the research. It reflects on the trade-offs between performance, portability, and management overhead in varying configurations. This final chapter also paves the way for future research, proposing directions and opportunities for further exploration in this field.

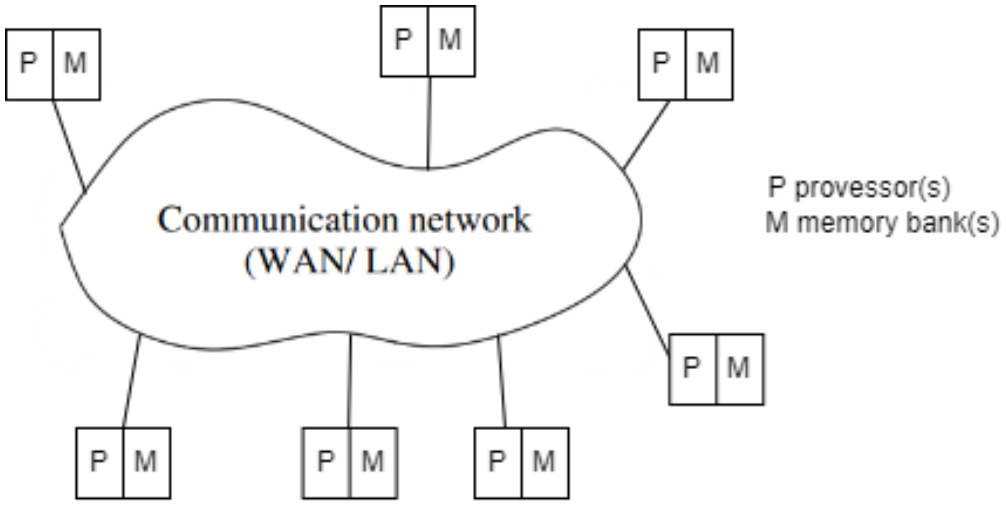
2 Theoretical background

2.1 Brief introduction to the concepts

2.1.1 Distributed Computing

The field of distributed computing covers a wide range of computational paradigms and information access models that extend across multiple processing elements connected through various forms of communication networks. These networks may cover local environments or extend to a wider area, presenting a diverse landscape for the design and implementation of distributed systems. The main challenge in distributed computing is the efficient coordination and management of computational tasks distributed across various nodes, promoting collaboration and parallelism, while addressing issues such as fault tolerance and scalability (Chaisawat & Vorakulpipat, 2020).

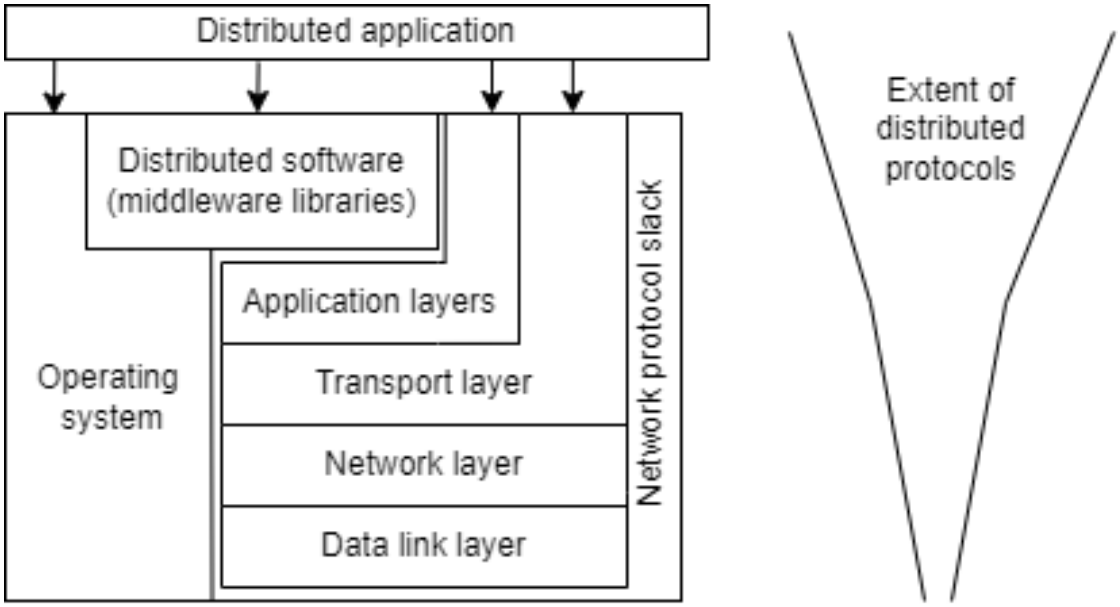
A distributed system, in this context, can be understood as a collection of nearly autonomous processors engaged in communication through a network. Key features of distributed systems include the absence of a common physical clock, the lack of shared memory, the geographical separation of processing elements, the autonomy of individual nodes, and the intermediate heterogeneity in terms of hardware and software settings. The complexity introduced by these characteristics requires innovative solutions and reliable algorithms to ensure optimal coordination among the distributed components, ultimately shaping the landscape for applications ranging from large-scale data processing to resilient and scalable cloud infrastructures (Eze & Akujuobi, 2022).



2.1-1: A distributed system connects processors by a communication network.

A typical distributed system is presented in the figure above. Each computer has a memory-processing unit, and the computers are connected through a communication network. Figure 2.1-2 schematically shows the relationship of the software components running on each computer, using the local operating system and the network protocol stack for their operation. The distributed software is also referred to as middleware (Salkenov & Bagchi, 2019). A distributed execution means an execution of processes throughout the distributed system for the collective achievement of a common goal.

The distributed system uses a multi-layered architecture to analyse the complexity of its design. The middleware drives the distributed system, providing platform-level heterogeneity transparency (Xiang et al., 2019). Figure 2.1-2 schematically shows the interaction of this software with the system components on each processor. We assume that the middleware level does not include the traditional functions of the application layer of the network protocol stack, such as http, mail, ftp, and telnet. Various primitives and function calls defined in various middleware-level libraries are embedded in the user program code. (Kshemkalyani & Singhal, n.d.)



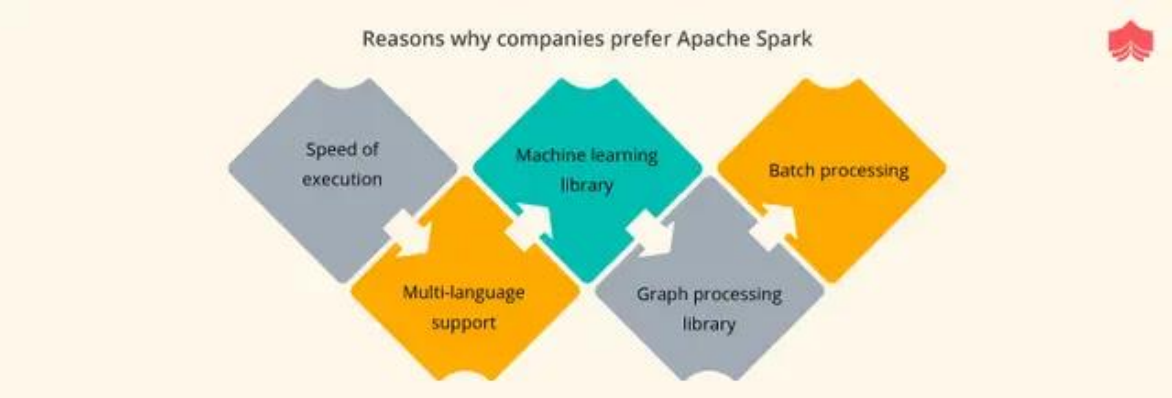
2.1-2 Interaction of the software components at each processor.

2.1.2 Apache Spark

Apache Spark is an open-source distributed computing framework designed for processing large data sets and executing complex data analysis workloads. It provides a flexible platform for distributed data processing, offering features such as in-memory data storage, fault tolerance and support for various programming languages.

Spark operates based on the concept of Resilient Distributed Datasets (RDDs), which are distributed data collections that can be processed in parallel in a cluster of machines. RDDs allow Spark to efficiently manage the division, distribution, and computation of data, making it suitable for tasks ranging from packet data processing to real-time data streaming (K & G, 2022).

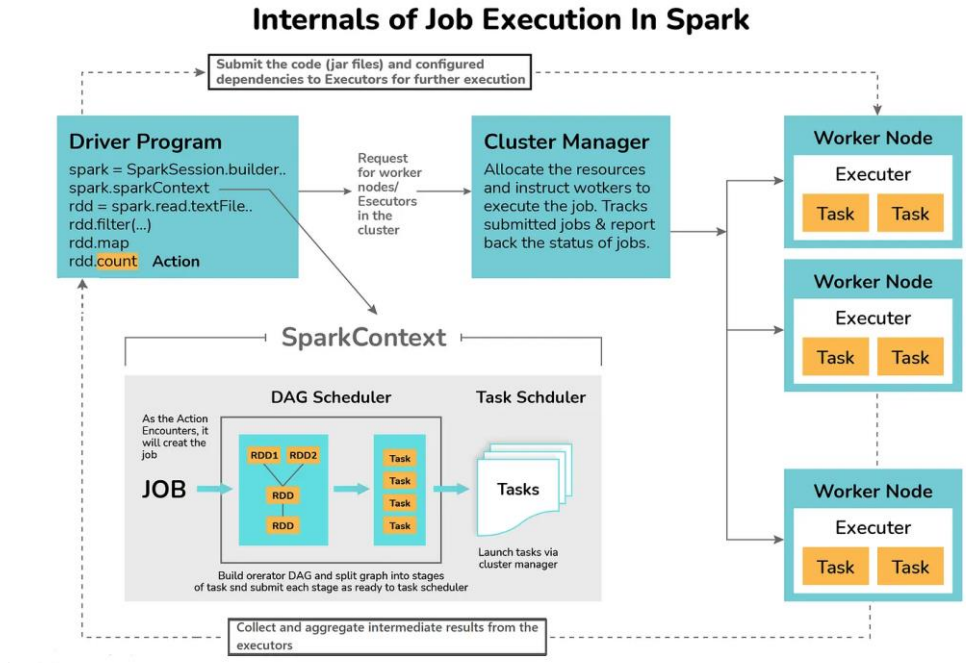
One of the main advantages of Spark is its compatibility with various deployment options (Hou et al., 2019), including containerization, virtualization, and bare metal capabilities. Spark can be comfortably integrated into various infrastructures, adapting to the specific requirements of each configuration (Qureshi et al., 2019a).



[2]

2.1-3: Benefits of having Apache Spark for Individual Companies

The architecture of Spark is shown in Figure 2.1-4, displaying a typical master-worker architecture. A Spark cluster has a single master and any number of workers (Neciu et al., 2021). The driver and the executors run their separate Java processes. The driver, running on the master node of the Spark cluster, creates a logical flow of operations according to all jobs, divides the flow into multiple stages, and ultimately schedules the tasks of the stages to the executors. It also stores the metadata for all RDDs. On the other hand, an executor is a distributed agent, responsible for executing tasks. Each Spark application has its own executors. Executors read from external sources, store the data of computational results in memory, cache, or hard disk drives, perform all data processing, and write the results to external sources. (Zhu et al., 2020a)



[3]

2.1-4 Spark Architecture

[2] https://www.knowledgehut.com/blog/big-data/spark-use-cases-applications

[3] https://medium.com/@shubham02gupta/demystifying-the-internal-workings-of-apache-spark-architecture-2918ec59fc5

2.1.3 Bare Metal

Bare metal implementation, as opposed to virtualization and containerization, involves running applications directly on the physical hardware without any intermediate layer. Analyzing the trade-offs between bare metal, virtualization, and containerization (Salehi et al., 2019) becomes critical for understanding the complex dynamics affecting the performance (Duplyakin et al., 2020) and portability of machine learning applications in multi-tenancy environments.

The raw computational power and minimal additional load associated with bare metal applications make them an exciting choice, particularly for applications requiring high performance (Lee & Fox, 2019). However, challenges related to scalability (Zhang et al., 2020) and shared resource space must be carefully considered.

2.1.4 Virtualization

Virtualization involves creating virtual environments of computing resources within a single physical computer. Virtualization allows the simultaneous execution of multiple operating systems on an underlying processor. By examining specific virtualization settings, we aim to discover the complex relationship between its choices and the overall performance of machine learning algorithms in the context of multi-user applications. In the field of virtualization, trade-offs between isolation, resource overhead, and scalability become critical considerations.(Campbell & Jeronimo, n.d.)

2.1.5 Containerization

Containerization has gained timely significance for its ability to encapsulate applications and their dependencies on lightweight, portable units called containers. Docker and Kubernetes are central to this paradigm, facilitating the creation, deployment, and scaling of containerized applications. In the upcoming sections, we will examine the impact of containerization on Apache Spark and distributed computing settings, exploring how it affects factors such as performance, scalability, and ease of deployment. Understanding the details of containerization, we can better comprehend its role in shaping multi-user applications and the trade-offs involved in adopting this technology.

The benefits of containerization go beyond resource efficiency and simplification of deployment; it introduces a level of abstraction that promotes cohesion in diverse environments. However, challenges such as the complexity of orchestration and potential overhead require a penetrating evaluation. The following sections will analyze specific containerization settings, offering a comprehensive analysis of their impact on the performance of machine learning algorithms in a multi-user environment. This exploration is critical for understanding the optimal balance between the advantages of containerization and the specific requirements of applications based on Apache Spark.(Bhat, 2018)

2.2 Advantages and disadvantages of each configuration

Bare metal configurations offer optimal performance. This direct interface with the hardware ensures that applications can advantage from the full computational power available, without any overhead from hypervisors or container runtimes (USENIX Association., 2003). Such performance efficiency is particularly critical for applications demanding high computational resources, like certain machine learning tasks. Bare metal systems also provide simplicity in certain contexts. For applications with specific, stable requirements, the straightforward nature of bare metal – without the need for

managing additional virtual layers – can be beneficial (Clements et al., n.d.). However, this configuration also presents several disadvantages. The lack of flexibility is a significant drawback; bare metal systems do not offer the same scalability and rapid provisioning capabilities as virtualized or containerized environments. This can be a limitation in dynamic computing scenarios where quick scaling up or down is required. Additionally, for diverse workloads, bare metal can prove pricey, as it might necessitate dedicated hardware for each type of application. Another consideration is the potential for increased maintenance downtime. Unlike virtualized environments where maintenance tasks can often be performed with minimal disruption, updates and maintenance on bare metal systems may require taking the entire system offline, leading to potential service interruptions.

Virtualization offers flexibility and efficiency. Its primary advantage lies in the strong isolation it provides. By running each application in a separate virtual machine (VM), complete with its own operating system, virtualization ensures robust security and fault tolerance (Seznec et al., 2010). This isolation is particularly beneficial in multi-tenant environments where the isolation of different workloads is essential. Additionally, virtual machines offer hardware abstraction, allowing applications to run on various hardware configurations without modification, a feature particularly useful in heterogeneous computing environments. Another benefit of virtualization is the flexibility in resource allocation. Resources such as memory, and storage can be dynamically allocated and reallocated among VMs, facilitating efficient utilization of physical resources (Misevičienmiseviˇmisevičien et al., 2012). However, these advantages come with certain drawbacks. Virtual machines are resource-intensive, as each VM runs its own operating system, leading to higher storage and processing overhead. This can result in slower performance compared to running applications directly on physical hardware (bare metal). The complexity in setting up and maintaining VMs, particularly in large-scale deployments, also poses a challenge, requiring significant expertise and resources.

Containerization offers a unique set of advantages. Its primary strength is its ability to encapsulate applications along with their dependencies into lightweight, portable entities known as containers. This portability ensures consistent performance across diverse computing environments, effectively addressing compatibility issues often encountered in software deployment. Furthermore, containers are recognized for their resource efficiency (Bhardwaj & Krishna, 2021). By sharing the host operating system's kernel and avoiding the overhead of virtual machines, they allow for more efficient use of system resources. Another significant advantage is the rapid deployment and scaling capability of containers (Bellavista & Zanni, 2017). The sharpness with which containers can be started, stopped, and replicated makes them particularly suited for environments where scalability and responsiveness are critical. Moreover, the isolation provided by containerization enhances security, as applications operate independently within their respective containers, minimizing the risk of inter-application conflicts and potential security breaches. Despite these benefits, containerization is not without its challenges. Security remains a concern, primarily because containers share the host operating system's kernel. If not properly secured, this shared environment can become a vector for security vulnerabilities. Additionally, the management of containers, particularly in large-scale deployments, introduces a level of complexity that necessitates sophisticated orchestration tools like Kubernetes (Casalicchio & Iannucci, n.d.). Lastly, while containers are more resource-efficient than virtual machines, they still introduce a performance overhead compared to bare metal configurations due to the additional layers involved in container runtime and image storage.

2.3 Previous studies

The exploration into containerized environments, especially using Kubernetes, has been extensive. Watts et al. (2021) provided crucial insights into the effectiveness of Kubernetes in a bare-metal cloud, employing introspection tools like Prometheus in conjunction with Apache Spark. Their findings suggest certain advantages in introspection capabilities offered by Kubernetes (Watts et al., 2021). Complementing this, Zhu et al. (2020) focused on performance aspects, noting Spark's superior

performance on bare metal, attributed mainly to better data locality. These studies collectively underscore the efficiency of bare metal in certain aspects but also highlight the advanced monitoring and management capabilities offered by Kubernetes in containerized setups (Zhu et al., 2020b).

In the realm of data analytics, Li et al. (2021) introduced ShadowVM, combining bare metal CPUs and GPUs. This approach, diverging from traditional JVM-based environments, achieved significant speedups, indicating the untapped potential of bare metal resources in high-performance data analytics. However, it's crucial to recognize the specific hardware dependencies in their methodology, which might limit the generalizability of these results (Li et al., 2021).

Container orchestration's impact on performance has been a main area. Horchulhack et al. (2022) discussed performance degradation in containerized Apache Spark due to multi-tenancy and hardware over-commitment (Horchulhack et al., 2022). This is complemented by Qureshi et al. (2019), who proposed a dynamic container-based resource management framework for Spark, aiming to enhance performance. The contrast between these two studies highlights a critical aspect: while containerization introduces certain complexities, innovative management frameworks can mitigate these issues (Qureshi et al., 2019b).

Further, the comparative performance of container orchestration mechanisms, explored by Beltre et al. (2019), reveals nuanced differences between Kubernetes and Docker Swarm. Both were found capable of nearing bare metal performance, yet Kubernetes might introduce overheads in specific applications. These findings are pivotal for the present research, as they offer a clear perspective on choosing appropriate orchestration tools based on application requirements (Beltre et al., 2019).

Liu and Guitart's (2020) study takes a critical look at the performance implications of multi-container deployment schemes, particularly relevant in high-performance computing (HPC) environments. Their research is important in understanding how different container technologies and deployment granularities affect the performance of distributed applications. By exploring various container configurations, Liu and Guitart provide a view of the trade-offs involved in container orchestration and resource allocation. Their work is insightful when considering the balance between container overhead and the benefits of isolation and scalability that containerization offers. The study examines how the granularity of container deployments - from lightweight, single-application containers to more complex, multi-service containers - impacts overall system performance, latency, and resource utilization (P. Liu & Guitart, 2021).

Finally, Kumar and Kaur's (2022) study talks about the performance of containerized Message Passing Interface (MPI) applications, particularly in the context of InfiniBand-based HPC systems. This empirical investigation compares the performance of containerized MPI applications with those running on bare metal setups, providing insights for deploying demanding computational workloads in containerized environments. Their research is relevant for understanding the overheads and benefits associated with containerization in environments where interprocess communication and network efficiency are critical. The study's findings highlight how containerization, despite its abstraction layer, can closely match the performance of bare metal setups, especially when optimized for network-intensive tasks. Kumar and Kaur's work is instrumental for this thesis as it offers a direct comparison between containerized and bare metal environments in handling high-performance, network-centric applications. It marks the potential of containerized environments to achieve near bare metal performance, while also illuminating the challenges and considerations necessary for optimizing such deployments in the context of Apache Spark (Kumar & Kaur, 2022).

These studies collectively provide a comprehensive understanding of the performance dynamics between containerized (especially Kubernetes orchestrated) and bare metal environments in the realm of distributed computing with Apache Spark. The insights gained from these studies serve as a foundation for further investigation into the optimal deployment strategies for specific computing needs in distributed environments.

3 Motivation

In the modern data-driven world, the demand for dashboards is continuously increasing. These dashboards play a critical role in providing information from complex data sources, particularly in areas such as traffic forecasting and incident detection. They also require fast data processing in the backend, which includes training and inference of machine learning models.

However, when these dashboards operate in environments based on a single computer, they often face performance and response issues, as they are centralized. Creating visualizations can be slow, leading to low usability and long waiting times for their users. This problem is particularly intense in scenarios such as traffic forecasting, where predictions must cover the entire network, or in incident detection, which involves processing data from many sensors in real-time. This problem is particularly intense in scenarios such as traffic forecasting, where predictions must cover the entire network, or in incident detection, which involves processing data from many sensors in real-time.

In the operation of dashboards, there are two fundamental dynamics: "push" and "pull". In traffic forecasting, dashboards operate in a "pull" state, where they request predictions for specific areas, leading to the execution of machine learning algorithms. On the other hand, in incident detection, a "push" approach is used, where algorithms operate independently and continuously, detecting and reporting incidents as they occur.

In the dashboard space, "push" refers to a process where the system automatically provides data or information to users without requiring action on their part. This means that users do not need to request updates but receive them automatically. For example, in traffic forecasting, "push" means that the system can automatically offer traffic predictions or information about traffic changes, allowing users to receive this information without actively utilizing the dashboard. This approach helps improve performance and response in critical situations, enhancing the system's effectiveness.

To address these challenges, the adoption of a distributed computing environment is proposed. In such an environment, machine learning algorithms can be executed distributed, improving system response (Verbraeken et al., 2020). Additionally, flexibility and scalability are offered when using cloud infrastructure, while improving the system's resilience to faults.

This transition to distributed computing is driven by the desire to improve usability and reduce waiting times for dashboard users. When machine learning algorithms operate in a distributed environment, performance is significantly improved. This improvement ensures that visualizations are created instantly, resulting in a more user-friendly experience.

While the transition to distributed computing is pursued to improve usability and reduce waiting times for dashboard users, it must be recognized that this transition is not always easy. It requires the activation and management of multiple servers that will run the machine learning algorithms, ensuring compatible versions of libraries and other required components. However, various technologies are proposed to reduce the time and effort required for the installation and maintenance of the backend for the execution of machine learning algorithms. Technologies such as containerization and virtualization provide flexibility, resource management, and application isolation, aiding in the efficient development and maintenance of the distributed computing environment. These technologies work together with distributed computing to offer a comprehensive solution that improves performance, response, and scalability, providing a path for the future development of dashboards tailored to various stakeholders and use cases.

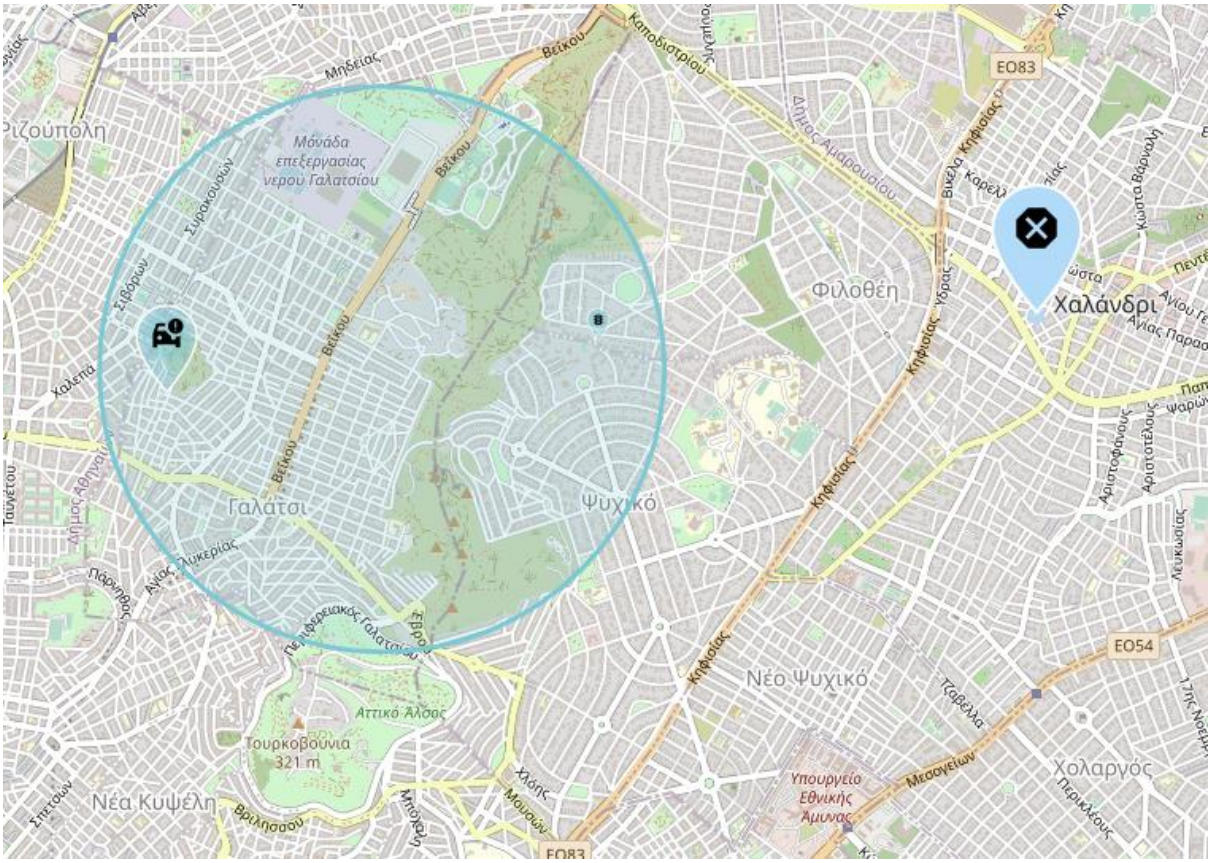
The main contribution of this work is to evaluate, in a structured and objective manner, different approaches to the installation and execution of ML algorithms within a traffic support application on

the road network. These tables should prioritize distributed environments to optimize performance and provide more immediate response to users. This proposal can be used by technology solution providers to choose the approach that will benefit all stakeholders as much as possible, from end-users seeking improved usability to dashboard providers and development and maintenance teams seeking improved management and greater flexibility.

The thesis evaluates two approaches to executing machine learning algorithms in a traffic support environment on the road network. The approaches include the use of "Bare Metal" for direct execution on physical computers, and the use of container technology, such as Kubernetes (K8s), for flexibility and resource management. Each approach has its advantages, with the choice depending on the specific needs of the environment and stakeholders.

Additionally, it is worth noting that the work addresses not only improvements in performance but also the need for portability across various infrastructures. This consideration is becoming increasingly important in the context of developing dashboards in various organizations and regions.

This thesis supports the development of high-performance dashboards that leverage machine learning algorithms for data visualization. In this way, the benefits of distributed computing are highlighted, as well as its synergy with containerization and virtualization technologies, which collectively have a positive impact on usability, response, and scalability, providing a path for the future development of dashboards, tailored to various stakeholders and use cases.



2.3-1: Image from the dashboard of the FRONTIER project, showing the visualization of some detected traffic anomalies.

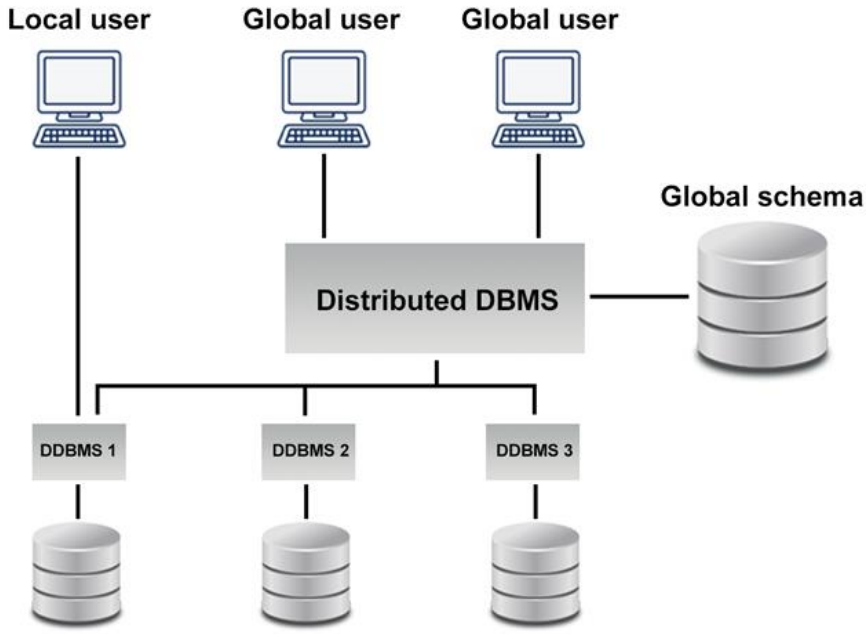
4 Distributed Processing and Machine Learning

Distributed machine learning refers to multi-node machine learning algorithms and systems designed to improve performance, increase accuracy, and scale to larger input data sizes. Increasing the size of input data for many algorithms can significantly reduce learning error and often be more effective than using more complex methods (E X P E R T O P I N I O N 8, 2009). Distributed machine learning allows companies, researchers, and individuals to make informed decisions and draw significant conclusions from large amounts of data.

4.1 Frameworks for distributed processing

There are many systems for executing machine learning tasks in a distributed environment. These systems are divided into three basic categories: database, general, and purpose-built systems (Galakatos et al., 2017). Each type of system has distinct advantages and disadvantages, but all are used in practice depending on individual use cases, performance requirements, input data size, and corresponding implementation effort.

Database systems are typically extensions or modifications of traditional database management systems (DBMSs), offering a more integrated environment for executing machine learning algorithms. These systems can include specific optimizations for data access patterns and storage layouts, which are essential in managing large datasets typically used in machine learning. They also benefit from the robust transaction and concurrency control mechanisms inherent in DBMSs.



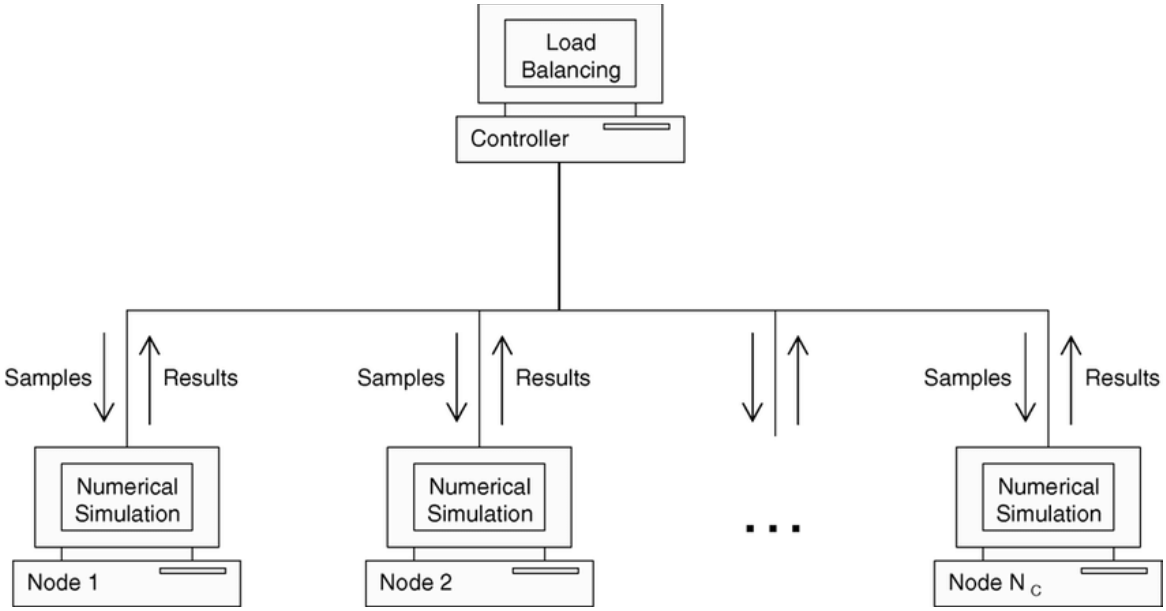
[4]

4.1-1: Database system architecture for distributed computing

[4] https://phoenixnap.com/kb/distributed-database

General frameworks, on the other hand, provide a level of abstraction that allows users to design and implement machine learning algorithms without the need to manage the details of the underlying distributed system. They offer APIs in high-level languages, enabling the development of complex algorithms in a more user-friendly manner. These frameworks are widely used because of their flexibility, scalability, and the extensive ecosystem of libraries and tools they support. Examples include:

- MPI (Message Passing Interface): A low-level framework designed for high-performance distributed computation.
- Hadoop: An open-source MapReduce implementation ideal for executing workflows on large clusters of commodity machines.
- Spark: Known for its in-memory computation capabilities, Spark allows users to compose workflows using predefined API operators and extends the MapReduce paradigm by supporting iterative algorithms efficiently.

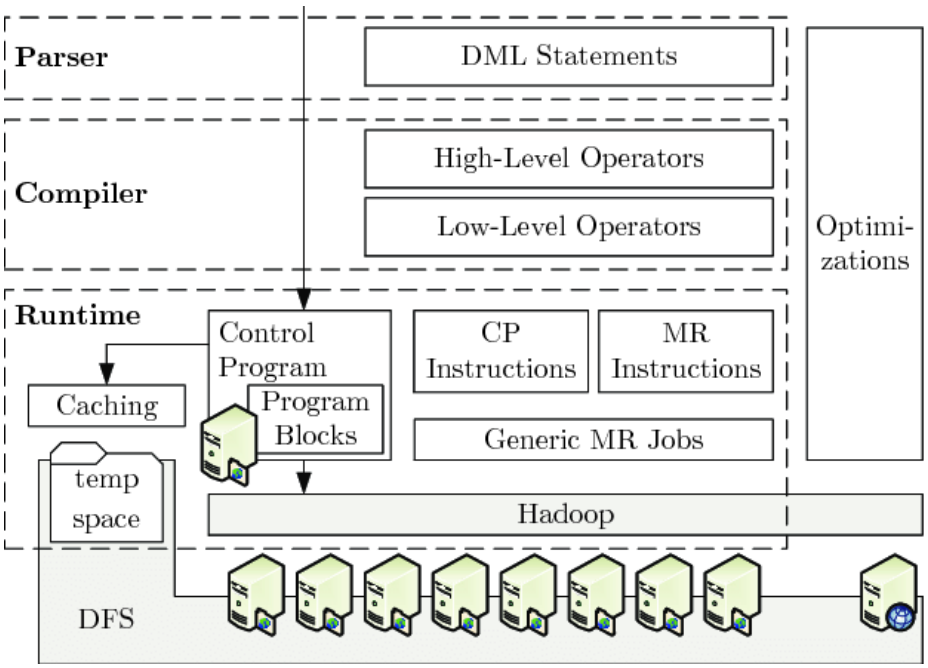


[5]

4.1-2: Schematic of a general framework for distributed computing.

Purpose-built systems, designed exclusively for specific machine learning tasks, offer highly optimized implementations. These systems tend to be less flexible but more efficient for their target tasks. They often include optimizations for particular data structures, computational patterns, and hardware configurations. SystemML, for instance, provides a high-level language with R-like syntax for matrix operations, while OptiML offers a Scala-embedded domain-specific language based on linear algebra operations. Hogwild! presents a lock-free implementation of stochastic gradient descent, optimizing for shared memory systems.

[5] https://www.researchgate.net/publication/220285046_Examination_of_load-balancing_methods_to_improve_efficiency_of_a_composite_materials_manufacturing_process_simulation_under_uncertainty_using_distributed_computing/figures?lo=1



[6]

4.1-3: SystemML Architecture

Apache Spark, the framework chosen for this research, stands out in the category of general frameworks. Spark's in-memory processing capabilities significantly reduce the time for iterative algorithms, a common pattern in machine learning. Additionally, Spark's resilient distributed datasets (RDDs) provide a fault-tolerant way to handle distributed data, an essential feature for robust distributed processing. Its rich ecosystem, including libraries like MLlib for machine learning, makes Spark a comprehensive tool for distributed machine learning. This research will utilize Spark's capabilities to evaluate distributed computing performance in containerized versus bare-metal environments.

4.2 Kubernetes and Kubernetes in Spark

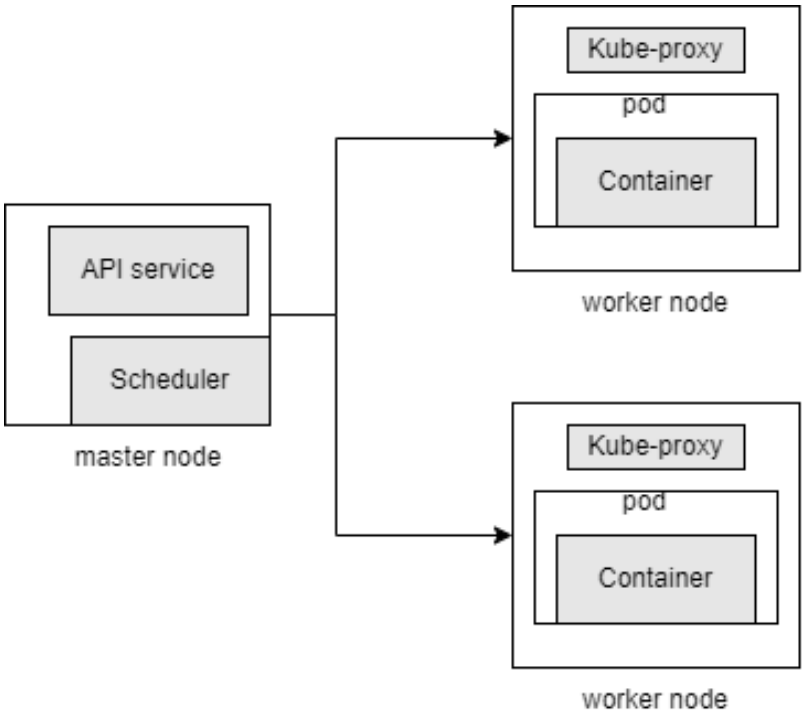
Kubernetes is increasingly used for the development of web applications based on containers, on physical computers within Platform-as-a-Service (PaaS) clouds, allowing the scale-out of an application with dynamic workload changes. Kubernetes also follows the master- worker architecture, as shown in Figure 4.2-1. The master node is responsible for managing the Kubernetes system. This is an entry point for all administrative tasks. The master node takes care of coordinating the worker nodes, where the actual services are executed.

In Kubernetes, pods, rather than containers, are the smallest computational development units, running on worker nodes. A pod can encapsulate one or multiple containers and is assigned a unique IP address. Each container in a pod shares the network namespace, including the IP address and network ports. Pods running on different physical computers can communicate via the Kube-proxy, a

[6] https://www.researchgate.net/publication/260592097_Hybrid_Parallelization_Strategies_for_LargeScale_Machine_Learning_in_SystemML/figures

component of Kubernetes. In Kubernetes, a pod's CPU request is usually one CPU core or less, so pods can be flexibly deployed on various nodes to maintain the flexibility and reliability of applications.

Spark can also use Kubernetes as its cluster manager, as well as other managers. In Kubernetes, all Spark drivers and executors run in pods and are scheduled by the native Kubernetes scheduler. Once a Spark application is submitted to a Kubernetes cluster, a Spark driver is created and initially runs within a pod, and then the driver creates Spark executors, which also run within pods, connect to them, and execute the application code. After the application is completed, the executors' pods are terminated and cleaned up, but the driver's pod retains the log files and remains in a "completed" state in the Kubernetes API until it is finally "garbage collected" or manually deleted. (Zhu et al., 2020a).

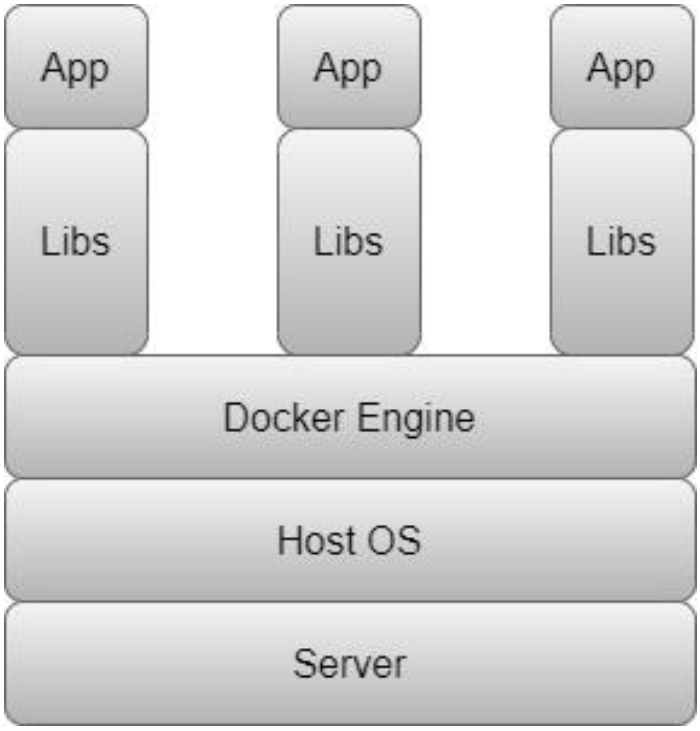


4.2-1 Kubernetes Architecture

4.3 Performance comparison of different architectures

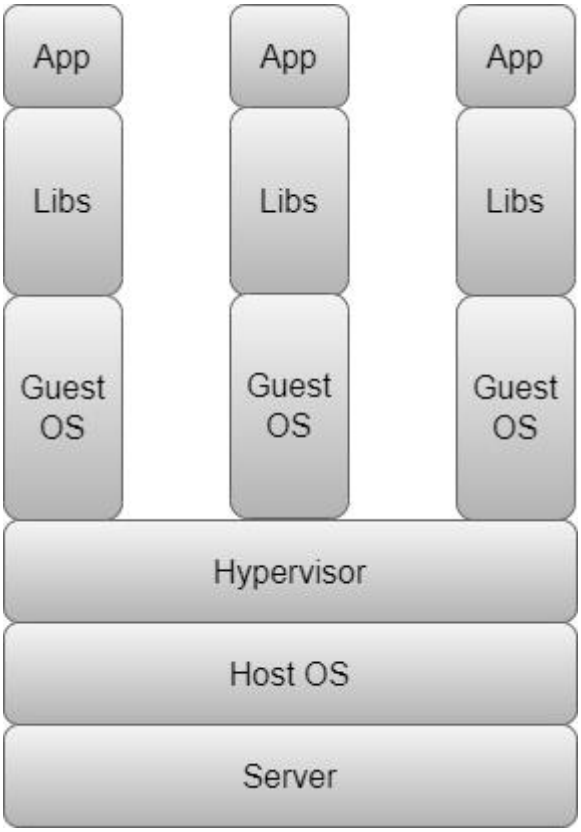
4.3.1 Containerization vs Virtualization

Docker isolates only one process (or a group of processes, depending on how the image is built), and all containers run on the same computing node. As isolation is applied at the kernel level, the execution of containers does not significantly burden the computing node compared to virtual machines. When a container is activated, the selected process or group of processes continues to run on the same computing node, without the need for virtualization or simulation of anything. Figure 4.3-1 shows three applications running in three different containers on a single physical computing node.



4.3-1 Representation of three apps running on three different containers.

In contrast, when a virtual machine is activated, the hypervisor virtualizes the entire system - from the processing unit to RAM and storage space. To support this virtualized system, an entire operating system must be installed. The virtualized system is an entire computer running within a computer. The load required to run a single operating system is already significant, so if an operating system is executed within another, it is even greater. Figure 4.3-2 shows a representation of three applications running in three different virtual machines on a single physical computing node.



4.3-2 Representation of three apps running on three different virtual machines.

Figures 4.3-1 and 4.3-2 give an indication of three different applications running on a single computing node. In the case of a virtual machine, we need not only the dependent libraries of the application but also an operating system to run the application. In comparison, with containers, the fact that they share the kernel of the operating system of the computing node with the application means that the additional load of an extra operating system is discarded. This not only significantly improves performance but also allows us to optimize resource usage and minimize untapped computational power. (Bhat, 2018)

4.3.2 Containerization vs bare metal

When comparing containerization with bare metal in the context of Apache Spark, several factors are critical:

1. **Resource Utilization**: Containerization allows for better resource utilization and efficiency, especially in multi-tenant environments. It enables running multiple isolated Spark jobs on the same physical hardware without interference.
2. **Startup Time and Scalability**: Containers have a significantly lower startup time compared to setting up new bare metal environments. This rapid scalability is beneficial in cloud environments where workloads can be volatile.
3. **Performance Overhead**: While containerization introduces some overhead due to the additional layer of abstraction, this is often minimal. However, in extremely resource-intensive tasks, bare metal can provide marginally better performance due to direct hardware access.

4. **Operational Complexity**: Managing bare metal deployments can be more complex and resource-intensive compared to containerized environments. Containerization offers easier deployment, management, and scaling, which is crucial in agile and dynamic computing environments.
5. **Flexibility and Portability**: Containerization provides higher flexibility and portability. Applications packaged in containers can be easily moved across different environments, a feature not as easily achievable with bare metal solutions.

The choice between containerization and bare metal in Apache Spark deployments hinges on the specific requirements of the use case. Containerization offers significant advantages in terms of scalability, resource efficiency, and operational simplicity, making it suitable for cloud and multi-tenant environments. Bare metal, while offering potentially higher performance, is more suited to stable, resource-intensive tasks where direct hardware access is critical.

In conclusion, the discussions in sections 4.3.1 and 4.3.2 have laid a solid foundation for the focus of this research on the comparison between containerization and bare metal in the context of Apache Spark. While section 4.3.1 defines the advantages of containerization over virtualization, particularly in terms of resource efficiency, scalability, and operational simplicity, section 4.3.2 emphasizes the trade-offs between containerization and bare metal deployments. It is evident from these discussions that containerization, in most scenarios, offers a more balanced approach, outperforming virtualization in terms of resource utilization and operational sharpness. This leads to the direct comparison between containerization and bare metal, as it is more appropriate for this research.

5 Approach

5.1 Case: multiple possible configurations for visualisation dashboard in a multi-tenancy app

In multi-tenancy applications, visualization dashboards serve as the interface for diverse users to interact with complex datasets, facilitating data-driven decision-making processes. These dashboards need to be highly responsive, scalable, and capable of handling concurrent requests without significant performance degradation. The underlying challenge is to maintain the integrity and isolation of data and services for each client, while offering a unified, seamless user experience.

For this research, the machine learning models will be implemented in two distinct configurations. The first is a containerized Apache Spark environment, where Apache Spark is deployed within containers and orchestrated by Kubernetes. This approach leverages the benefits of containerization, including scalability, resource efficiency, and deployment agility, which are essential in a multi-tenancy context. The second configuration is Apache Spark on bare metal, involving a direct deployment on physical servers. This setup aims to maximize computational power and memory efficiency by eliminating the overhead associated with containerization. The second setup is Apache Spark on bare metal. This configuration involves deploying Apache Spark directly on physical servers. The expectation here is to utilize the maximum computational power and memory efficiency, given the absence of containerization overhead.

The effectiveness of these configurations is assessed on several fronts. One of them is performance, which is measured in terms of responsiveness, data processing speed, and the capacity to handle concurrent user requests. Next, scalability is evaluated by the system's ability to adapt to fluctuating workloads, a frequent occurrence in multi-tenancy applications. Moreover, resource utilization is another critical factor, particularly in cost-constrained environments, assessing how efficiently the system uses hardware resources. Additionally, isolation and security are imperative to ensure that the data and processing of one tenant do not interfere with another.

Practical considerations of these configurations include various aspects. User experience is paramount, with the dashboard's responsiveness and reliability directly impacting the user's ability to make timely, data-informed decisions. Maintenance and upgradability are also crucial, especially considering the diverse needs of multiple tenants, dictating how easily new features can be deployed and existing ones maintained. Lastly, cost efficiency is a key consideration, seeking a balance between performance and resource utilization to minimize operational costs.

The choice of configuration has significant implications for the overall effectiveness of a multi-tenancy app. A containerized environment, while offering scalability and ease of management, might introduce performance overheads. On the other hand, bare metal setups, though potentially more performant, could pose challenges in scalability and rapid deployment.

In conclusion, it is necessary to select an appropriate computational configuration for visualization dashboards in multi-tenancy applications. The evaluation of containerized versus bare metal environments, specifically using Apache Spark, is important in determining the optimal setup that balances performance, scalability, and cost efficiency, with ensuring an effective, responsive, and user-friendly dashboard experience in multi-tenant scenarios.

5.2 Dataset

To evaluate the performance of various configurations for machine learning algorithms in the context of incident detection, a comprehensive dataset from Kaggle, named "US-Accidents", is used. This dataset includes approximately 2.97 million traffic accident incidents that occurred in the United States from February 2016. The Kaggle dataset is continuously collected through various sources, with multiple APIs providing traffic event data streams. These APIs, managed by entities such as the US and state departments of transportation, police departments, traffic cameras and traffic sensors on road networks, record vital information for assessing traffic accidents and their impacts.

The dataset covers 49 states of the United States. The data is provided in CSV format, with 45 features describing various aspects of each accident record. Significant features include a unique identifier (ID), accident severity ('Severity'), start and end times of the accident ('Start_Time' and 'End_Time'), geographical coordinates of the starting and ending points ('Start_Lat', 'Start_Lng', 'End_Lat', 'End_Lng'), distance affected by the accident ('Distance(mi)'), physical description of the accident ('Description'), and weather characteristics such as temperature, wind speed, rainfall, and weather condition.

5.2-1 US-Accidents Dataset Details

Category	Features
Traffic Attributes	id, source, TMC, severity, start_time, end_time, start_point, end_point, distance, description
Address Attributes	number, street, side (left/right), city, county, state, zip-code, country
Weather Attributes	time, temperature, wind_chill, humidity, pressure, visibility, wind_direction, wind_speed, precipitation, condition
POI Attributes	Amenity, Bump, Crossing, Give_Way, Junction, _Exit, Railway, Roundabout, Station, Stop, Traffic_Calming, Traffic_Signal, Turning_Loop
Period-of-Day	Sunrise/Sunset, Civil Twilight, Nautical Twilight, Astronomical Twilight

Total Accidents	2,974,336
# MapQuest Accidents	2,257,521 (75.89%)
# Bing Accidents	684,097 (22.99%)
# Reported by Both	32,718 (1.1%)
Top States	California (485K), Texas (238K), Florida (177K), North Carolina (109K), New York (106K)

The dataset includes a variety of accidents, with various levels of severity and geographical locations. The dataset covers 49 states of the United States. The data is provided in CSV format, with 45 features describing various aspects of each accident record. Significant features include a unique identifier (ID), accident severity ('Severity'), start and end times of the accident ('Start_Time' and 'End_Time'), geographical coordinates of the starting and ending points ('Start_Lat', 'Start_Lng', 'End_Lat', 'End_Lng'), distance affected by the accident ('Distance(mi)'), physical description of the accident ('Description'), and weather characteristics such as temperature, wind speed, rainfall, and weather condition.

Table 5.2-1 provides an overview of the dataset, showing the total number of accidents, the percentage of accidents reported from various sources (MapQuest, Bing), and the states with the highest number of accidents.

To facilitate comparative analysis of various implementation approaches, the dataset is detailed further regarding features related to traffic, direction, weather, points of interest (POI), and time of day. Additionally, the dataset is segmented for specific cities, including Atlanta, Austin, Charlotte, Dallas, Houston, and Los Angeles, with an emphasis on achieving diversity in traffic and weather conditions. The temporal aspect is considered by sampling data from June 2018 to December 2018, covering 12 weeks, to reduce the impact of seasonality on weather and traffic patterns. (Moosavi, Samavatian, Parthasarathy, Teodorescu, et al., 2019)

For training and testing the accident prediction framework, all data were used, and each entry is represented by 113 time-invariant features and 8 × 24 time-varying features. Due to the rarity of accidents in the dataset, negative sampling was used to balance the frequency of samples between accident and non-accident categories. (Moosavi, Samavatian, Parthasarathy, & Ramnath, 2019)

This comprehensive dataset, covering a wide range of features related to traffic accidents, provides a powerful basis for evaluating the performance of machine learning algorithms in the proposed distributed computing environment. The richness and diversity of the data supports the exploration of various configurations and the comparison of implementation approaches to enhance the development of high-performance dashboards for traffic support applications on the road network.

5.3 Requirements

5.3.1 Multiple algorithms

The choice of multiple machine learning algorithms within the experimental framework is significant. The objective is to gain a comprehensive understanding of the performance implications across various distributed computing configurations when executing a diverse set of algorithms. The selected algorithms for this study include:

- Random Forest
- Multilayer Perceptron (MLP)
- Multiclass Logistic Regression
- Decision Tree
- Naive Bayes Classifier.

Including multiple machine learning algorithms serves a dual purpose. Firstly, it allows a holistic evaluation of computing environment configurations by examining how each algorithm performs under various conditions. Secondly, it contributes to the generalizability of the findings, facilitating a more reliable interpretation of the results. Various algorithms have unique features, and their performance can be affected by the underlying infrastructure. Therefore, using a diverse set of algorithms ensures a detailed evaluation of the configurations.

The variety of the algorithms aims to capture a broader spectrum of computational patterns and resource usage scenarios. Each algorithm brings in focus distinct computational requirements, varying in terms of distributed processing demands and memory usage. By incorporating a mix of classification and regression algorithms, we aim to evaluate the impact of computing configurations on various aspects of machine learning tasks.

The analysis of the performance of multiple algorithms contributes to enhancing the interpretability of the results. The comparative evaluation provides insights into how the strengths and weaknesses of each algorithm manifest in various computing environments. This, in turn, helps draw more detailed conclusions about the trade-offs and optimizations that may be necessary for specific algorithm-environment combinations.

5.3.2 Cloud Native / portable

The implementation of a cloud native and portable architecture in applications utilizing Apache Spark for distributed computing is crucial. This approach is essential in ensuring that applications remain flexible and efficient across various computational environments.

Cloud native applications embrace the dynamic capabilities of cloud computing (Nr & Rezzakul Haider, 2016). They are constructed and deployed to fully utilize cloud features such as scalability, resilience, and distributed processing. For applications relying on Apache Spark, being cloud native implies the ability to efficiently utilize cloud functionalities like auto-scaling, rapid provisioning, and the integration of microservices.

Portability, on the other hand, refers to the ability of the application to run across different computing environments without significant changes (Pop et al., 2014). This is crucial in a scenario where the underlying infrastructure might vary between cloud providers, on-premises data centers, or hybrid configurations. A portable setup ensures that the application can be seamlessly migrated or replicated across different environments, offering flexibility, and reducing vendor lock-in risks.

To successfully implement a cloud-native and portable architecture, especially for applications utilizing Apache Spark, certain key strategies are employed. The first strategy is containerization. This involves using container technologies, such as Docker (Bhimani et al., 2017), which package the application along with its dependencies into a container image. This method is fundamental in achieving portability, ensuring that the application runs consistently across any computing environment that supports containerization. The second strategy involves the use of orchestration tools. Kubernetes is a prime example of such a tool, which is instrumental in managing containerized applications across various environments. Orchestrating with tools like Kubernetes helps in maintaining consistency, enabling scaling, and ensuring resilience of the application. This is particularly vital in environments that vary between cloud and on-premises setups (Kratzke, 2018).

However, implementing a cloud-native and portable architecture is not without its challenges. One significant challenge is the complexity that arises in orchestrating, monitoring, and ensuring security across diverse environments (Ugwuanyi et al., 2020). This complexity necessitates advanced strategies and tools for effective management. Another hurdle is maintaining optimal performance (Mkandla & Chikohora, 2021). Each environment may require specific optimizations and fine-tuning to achieve

desired performance levels. Lastly, a major challenge lies in achieving consistency in data management. Ensuring uniform data handling and processing across different cloud and on-premises environments poses a significant challenge, requiring diligent planning and execution. These challenges necessitate a comprehensive approach, incorporating both technical expertise and strategic foresight, to successfully implement and maintain a robust, cloud-native, and portable architecture in Apache Spark-based applications.

5.3.3 Non-functional / performance requirements

The non-functional and performance requirements are crucial in ensuring the efficiency of applications leveraging distributed computing with Apache Spark. These requirements form the foundation of a system that supports optimal operational performance.

The scalability of the system is a primary concern (Choi et al., 2021). The system must be capable of both vertical and horizontal scaling to manage fluctuating loads in a multi-tenancy context, where the number of users and data volume can change significantly. This adaptability includes the ability to dynamically add or remove computational resources in response to real-time demand, which is essential for maintaining consistent performance. Alongside scalability, the system's availability and reliability are important. High availability minimizes downtime and ensures continuous access to the dashboard. The system should be designed with redundancy and failover mechanisms to gracefully handle potential failures. Reliability, in terms of consistent and accurate performance, is equally critical (Thiruvathukal et al., 2019).

Another key aspect is responsiveness and latency. The dashboard must provide a high level of responsiveness, with minimal latency in data processing and visualization rendering, to ensure a seamless user experience. Any delay or sluggishness can significantly impact usability and decision-making (Chang et al., 2016). Furthermore, resource efficiency is vital for cost-effective operation, especially when scaling in cloud environments (Zaharia, 2019). The system should optimize the use of computational resources (memory, storage) to run intensive tasks efficiently, thereby reducing unnecessary resource consumption.

Security and data isolation are critical in a multi-tenancy environment (Neves & Bernardino, 2015). The system must incorporate robust security measures, including data encryption, secure access controls, and isolation of tenant data, to prevent unauthorized access and data breaches. In addition, the system should be interoperable, designed to function across different cloud providers and on-premises environments (Lokuciejewski et al., 2021). It must seamlessly integrate with various data sources and be compatible with other systems, tools, and services.

Maintenance and upgradability of the system are essential for its long-term viability. The system should facilitate easy maintenance, allow for disruption-free upgrades, and support automated deployment and updates, without significant downtime or disruption (Nagar, 2017). This also includes capabilities for efficient troubleshooting and monitoring. Additionally, the system must adhere to relevant industry standards and compliance requirements, especially regarding data handling and privacy. The system should be designed to meet these regulations, ensuring legal and ethical use of data (Suneetha et al., 2020).

Lastly, robust monitoring and logging capabilities are imperative for tracking performance metrics, system health, and user activities. These features are critical for proactive maintenance, performance optimization, and security auditing. Furthermore, a comprehensive disaster recovery plan is necessary, including regular backups and a clear strategy for service restoration in the event of significant failures or disasters (Alnafessah & Casale, 2020).

6 Experimental Setup

6.1 Description

In our experiment, we run the five selected algorithms in one bare metal and three Kubernetes configurations to evaluate the performance of distributed computing using Apache Spark. A comprehensive understanding of the configurations in both the hardware and software domains is provided. These configurations are essential in supporting the effort to analyse the differences and similarities, the pros, and cons of the approaches of containerization and bare metal in the field of distributed computing.

6.2 Spark

Throughout our experiments, Apache Spark serves as the common thread, allowing us to evaluate its performance and behaviour in containerization, and bare metal configurations. The spark version used is 3.1.1. By examining how Spark interacts with each of these configurations, we aim to understand the trade-offs, benefits, and drawbacks associated with each approach in the field of distributed computing. (Salloum et al., 2016)

6.3 Algorithms

The selected machine learning algorithms have been chosen due to their relevance to incident detection and their variety in terms of machine learning categories. These algorithms are Random Forest, Multilayer Perceptron (MLP), Multiclass Logistic Regression, Decision Tree, and Naive Bayes. The rationale behind this choice is to facilitate a comprehensive comparison of different configurations, whilst considering the unique characteristics of each algorithm.

These algorithms include a variety of machine learning techniques, allowing us to evaluate how various containerization, and bare metal configurations affect their performance. Through them, we aim to gain valuable insights into the trade-offs between performance, portability, and management overhead in distributed computing environments during their application in incident detection tasks.

6.3.1 Random Forest

The Random Forest algorithm is an ensemble learning method ideal for incident detection due to its aggregation of multiple decision trees to create a more accurate and reliable predictive model. This method is capable at handling large, complex datasets with numerous feature interactions, a common characteristic in incident detection scenarios. Within Apache Spark, Random Forest is implemented using the RandomForestClassifier class. It constructs numerous decision trees during training and uses their collective predictions for the final decision, thus reducing the risk of overfitting and enhancing prediction accuracy. The algorithm's ability to manage various data distributions, including unbalanced datasets often encountered in incident detection tasks, makes it highly versatile. In traffic incident detection, Random Forest has been shown to perform effectively, particularly in scenarios with unbalanced data, by combining factor analysis with a weighted approach to improve detection accuracy (Jiang & Deng, 2020).

6.3.2 Multilayer Perceptron

The Multilayer Perceptron (MLP) is a form of neural network that consists of multiple layers of interconnected neurons, capable of learning complex patterns within large datasets. This versatility makes MLP suitable for both classification and regression tasks in incident detection. Within Apache Spark, the MLP can be implemented using the MultilayerPerceptronClassifier class. The ability of MLP to notice complex patterns and relationships in data allows it to detect subtle and detailed incident patterns, which can be beneficial in complex incident detection scenarios. MLP's effectiveness in large datasets as demonstrated in traffic incident detection highlights its applicability in diverse incident detection contexts (Kongkhaensarn & Piantanakulchai, 2018).

6.3.3 Multiclass Logistic Regression

Multiclass Logistic Regression extends the logistic regression model to handle scenarios where predictions are needed for more than two classes. This method predicts the probability of each class, making it particularly suitable for incident detection tasks that require distinguishing among multiple types of incidents. Apache Spark offers a flexible implementation of Multiclass Logistic Regression through the LogisticRegression class. The simplicity and interpretability of this model are beneficial, especially in understanding and communicating incident detection results. Its application in patient safety incident reports, for example, demonstrates the model's practical utility in real-world incident detection scenarios (Wang et al., 2017).

6.3.4 Decision Tree

The Decision Tree algorithm is a versatile and easy-to-understand machine learning technique. It creates a model that predicts the value of a target variable based on several input variables. Each internal node of the tree represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label (decision). Apache Spark provides an implementation of the Decision Tree Classifier algorithm. This simplicity of structure makes it highly interpretable, which is vital in incident detection for understanding the factors leading to an incident. In traffic incident detection, decision trees have been successfully applied, showing their strength in classifying and understanding different types of traffic incidents based on various traffic parameters (Chen & Wang, 2009).

6.3.5 Nayve Bayes Classifier

The Naive Bayes Classifier, based on Bayes' theorem, is a straightforward yet powerful algorithm for classification tasks, particularly effective in incident detection involving categorical data. Despite its fundamental assumption of independence among features, it has shown a good performance in various applications. It is known for its speed and efficiency in handling large datasets, making it suitable for incident detection tasks where rapid processing of large volumes of data is required. Apache Spark enables the implementation of Naive Bayes through the NaiveBayes class. In the context of traffic incident detection, the Naive Bayes Classifier has been effectively used in ensemble settings to improve detection performance and stability, particularly in scenarios involving large datasets and complex feature sets (Q. Liu et al., 2014).

6.4 Configurations

The specific configurations used for our experimental tasks represent various infrastructure approaches, each with its unique characteristics and trade-offs. The choice of configurations is important, as it directly affects performance, scalability, and portability of Apache Spark applications. We have chosen four distinct configurations for our experimental tasks:

- Bare Metal - Single PC
- Kubernetes – one node spark cluster
- Kubernetes – two node spark cluster
- Kubernetes – four node spark cluster

6.4.1 Bare Metal – Single PC

In the Bare Metal configuration with a single node, a single personal computer is used as the computing node for the Spark cluster. This setup represents the traditional approach of running Spark directly on dedicated hardware without any abstraction layer. The main features of this setup include:

- **Hardware**: A single computer (PC) with dedicated CPU, RAM, and storage resources.
- **Spark Cluster**: A Spark cluster with a single node, running on bare metal hardware.
- **Operating System**: The computing node operates directly on its native operating system.
- **Cores**: 4

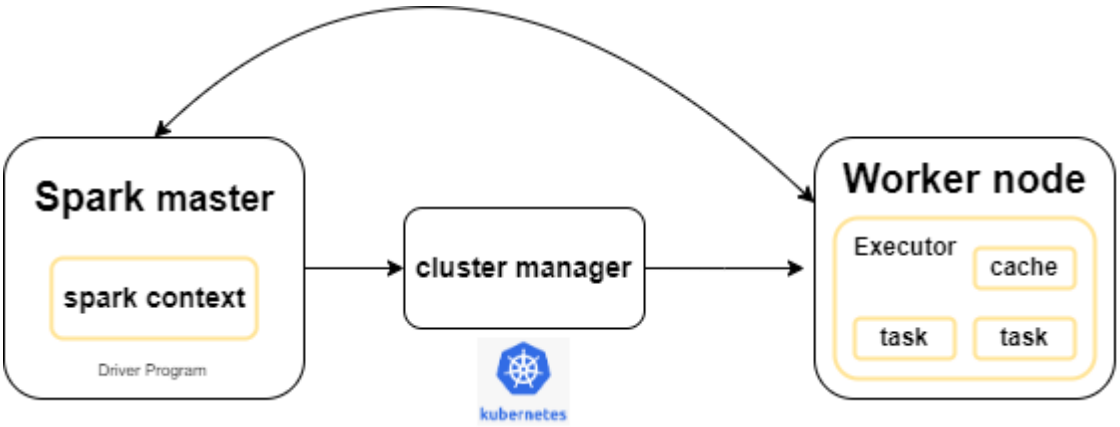
We include this configuration to establish a baseline performance measurement that represents a traditional approach to distributed computing without any abstraction layers. This helps us evaluate the raw, physical performance of Apache Spark on dedicated hardware.

6.4.2 Kubernetes – one node spark cluster

In the Kubernetes (K8s) configuration with a one-node Spark Cluster, Kubernetes was used, a popular container orchestration platform, to manage a Spark cluster consisting of a single node. Kubernetes provides a high level of abstraction and resource isolation, making it a suitable choice for tasks running on workloads. The main features of this setup include:

- **Kubernetes**: A Spark cluster with one node managed by Kubernetes.
- **Docker Containers**: Apache Spark and its required dependencies are containers managed by Docker and orchestrated by Kubernetes.
- **Resource Management**: Kubernetes dynamically manages the CPU and memory resources allocated to the Spark containers.
- **Cores**: 4

Kubernetes is a popular choice for container orchestration and offers enhanced resource management capabilities. We aim to evaluate how running Spark on Kubernetes affects performance and resource usage compared to the bare metal configuration.



6.4-1: Kubernetes - one node spark cluster: Execution flow

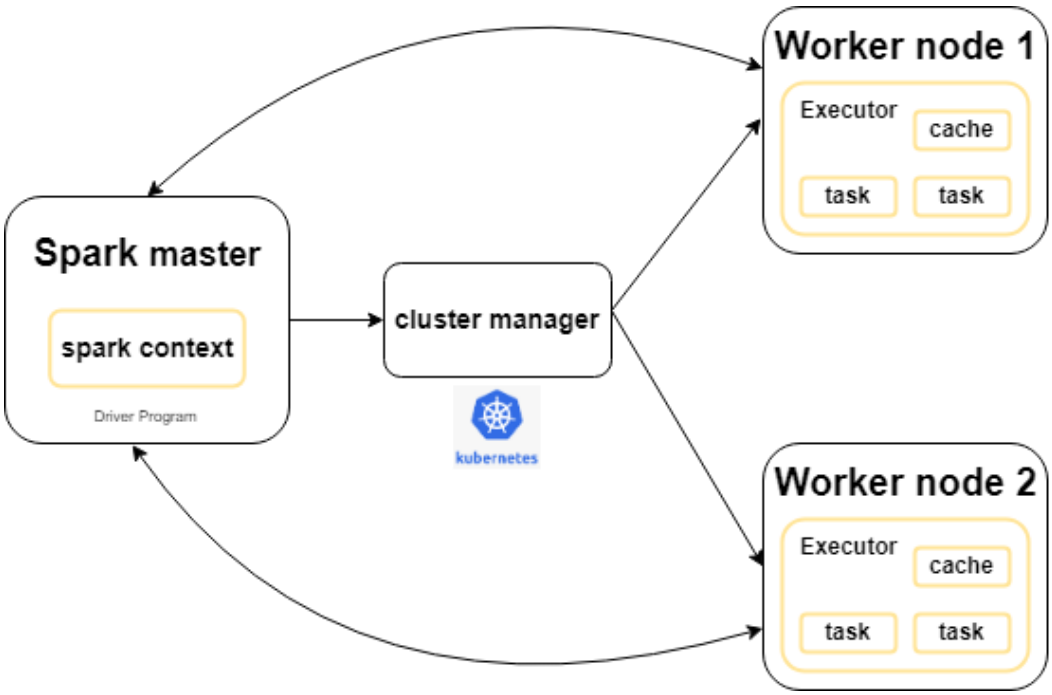
6.4.3 Kubernetes – two node spark cluster

In the Kubernetes (K8s) configuration with a two-node Spark Cluster, the Kubernetes setup was expanded to include a Spark cluster with two nodes. This setting introduces additional complexity by distributing the workload across multiple nodes.

The main features of this setup are:

- **Kubernetes**: A Spark cluster with two nodes managed by Kubernetes.
- **Node Scaling**: Spark tasks can be distributed across two nodes, potentially improving parallel distributed execution and performance.
- **Inter-node Communication**: Spark nodes communicate via the Kubernetes network.
- **Cores**: 4

With the increased complexity of the Kubernetes environment, we aim to evaluate how Spark applications perform when distributed across multiple nodes managed by Kubernetes. This configuration helps us understand the benefits of scalability in container orchestration.



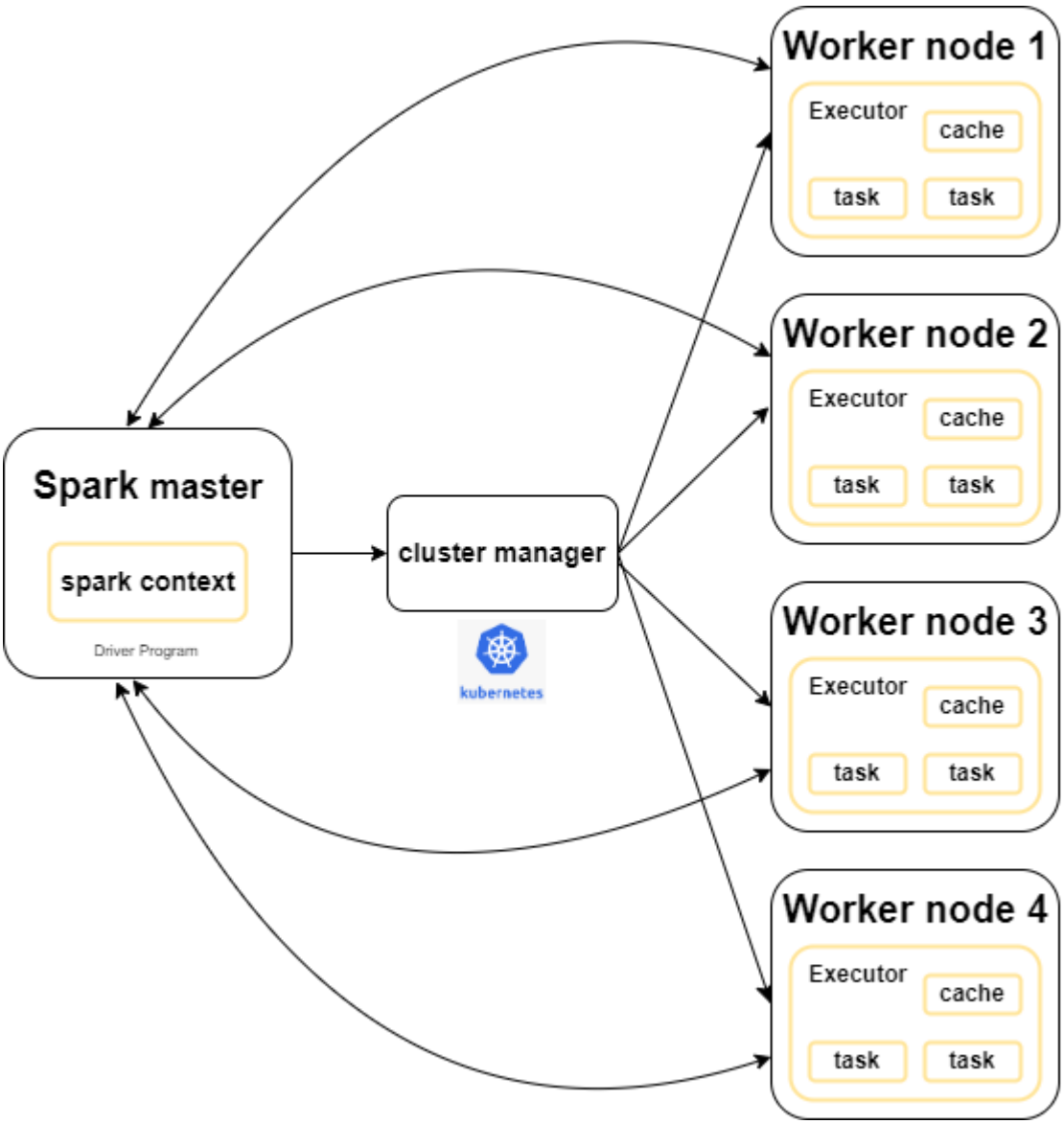
6.4-2 Kubernetes - two node spark cluster: Execution flow

6.4.4 Kubernetes – four node spark cluster

In the Kubernetes configuration with a four-node Spark Cluster, we further extend the Kubernetes setup to include a Spark cluster consisting of four nodes. This setting introduces additional complexity by distributing the workload across multiple nodes. The main features of this setup include:

- **Kubernetes:** A Spark cluster with four nodes managed by Kubernetes.
- **Node Scaling:** Spark tasks can be distributed across four nodes.
- **Inter-node Communication**: Spark nodes communicate via the Kubernetes network.
- **Cores**: 4

As the complexity of the Kubernetes environment increases, we aim to evaluate how Spark applications perform when distributed across multiple nodes managed by Kubernetes. This configuration helps us understand the benefits of scalability in container orchestration.



6.4-3 Kubernetes - four node spark cluster: Execution flow

These architectures aim to provide a comprehensive picture of how various computing environments affect the performance of Apache Spark. The results emerging from each of these settings will be analyzed and compared in the following sections, offering conclusions on the trade-offs between performance, portability, and management overhead associated with each composition.

7 Results

7.1 Table with algorithms and performances, for multiple configurations

Now, a comprehensive table and accompanying figures that detail the performance metrics of various machine learning algorithms under multiple configurations is presented. These configurations include both bare metal and Kubernetes-based distributed environments, with differing node counts in the Apache Spark clusters. The table showcases key performance indicators (KPIs) for each algorithm such as:

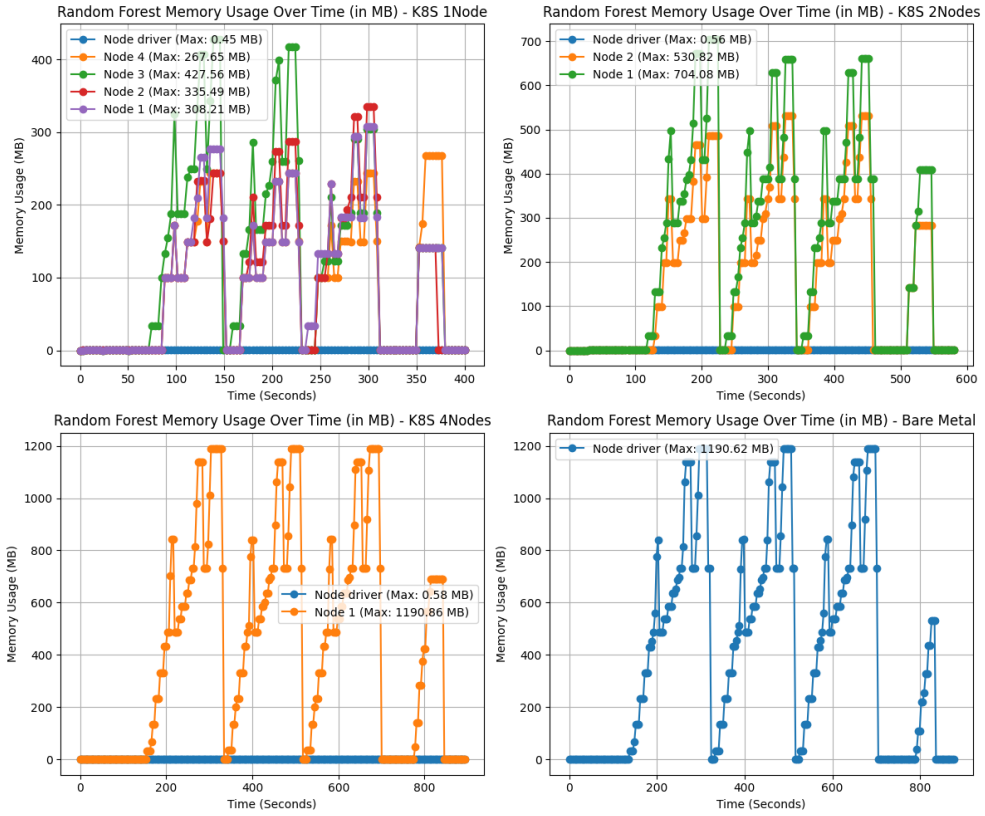
- Execution time (both for training and testing)
- Memory consumption per node
- Memory consumption in all

This section serves as a visual and data-driven foundation for the subsequent analytical discussion, allowing for a clear comparison between containerized and bare metal environments in distributed computing.

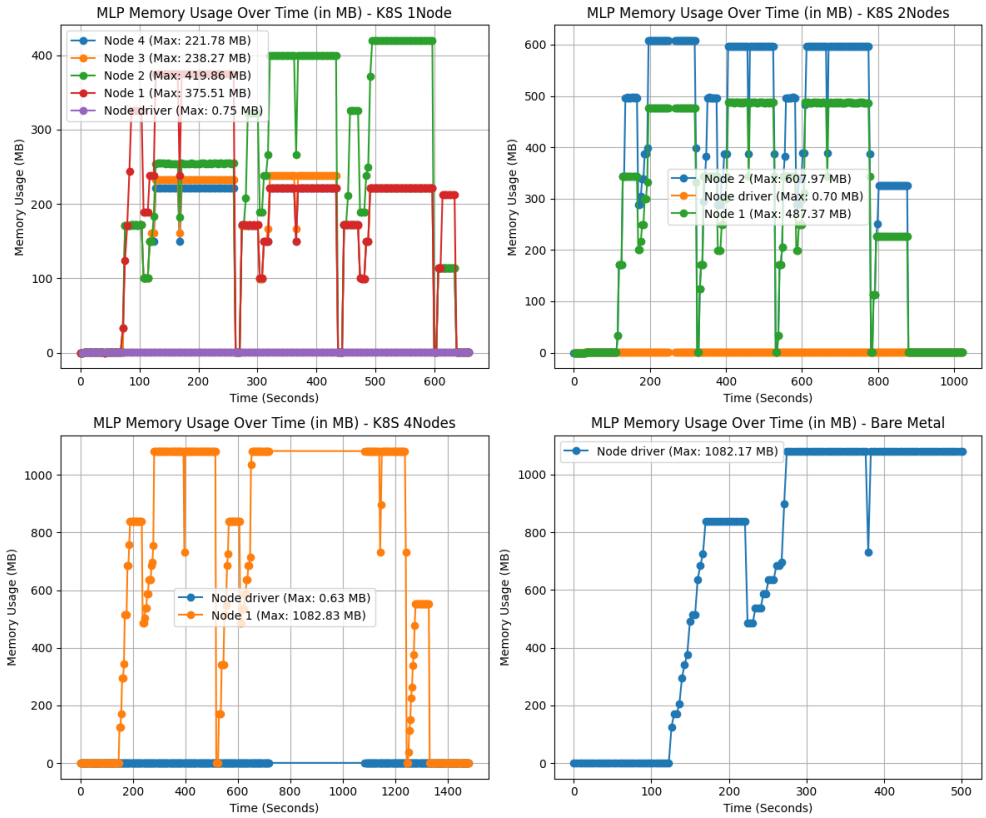
7.1-1 Results Table

Algorithms	Configuration / Setup	Performance (KPIs)		
		Time	Memory (per node)	Memory (across all nodes)
Random forest	Bare metal – single PC	717.85 s	1190.64 MiB	1190.64 MiB
	K8s – one node spark cluster	729.86 s	1190.86 MiB	1191.18 MiB
	K8s – two node spark cluster	453.06 s	704.07 MiB	1191.51 MiB
	K8s – four node spark cluster	334.00 s	427.56 MiB	1192.03 MiB
MLP	Bare metal – single PC	1237.64 s	1082.91 MiB	1082.91 MiB
	K8s – one node spark cluster	1309.26 s	1082.83 MiB	1083.46 MiB
	K8s – two node spark cluster	896.01 s	607.97 MiB	1085.14 MiB

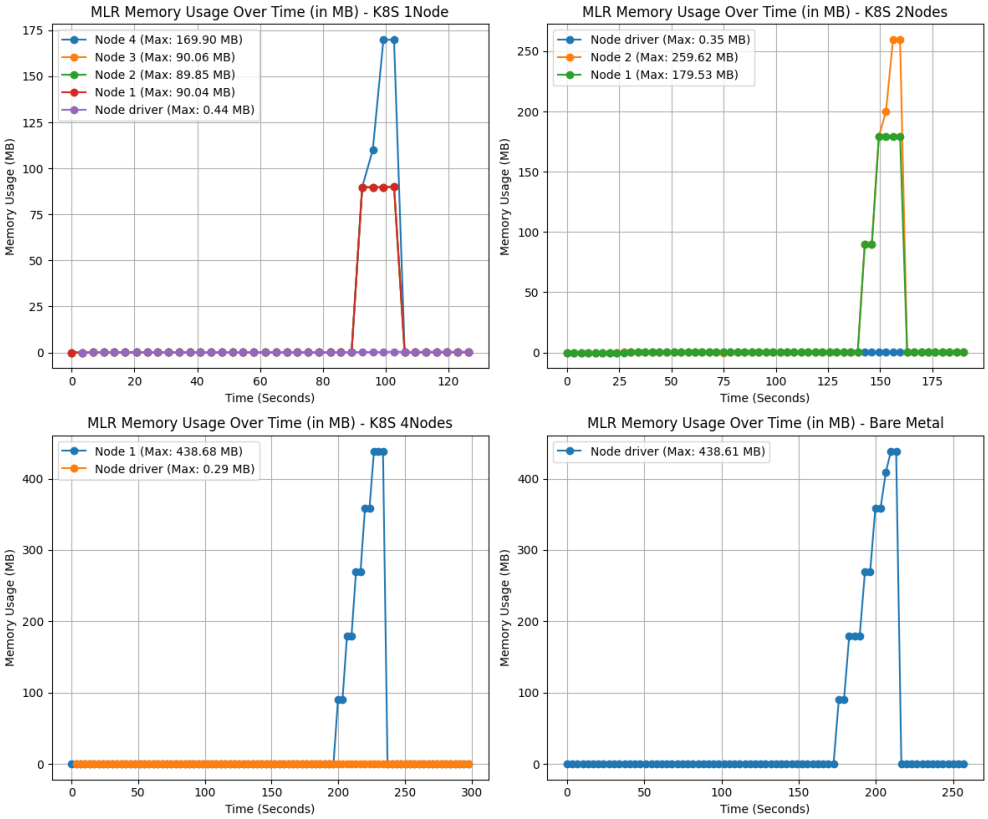
	K8s – four node spark cluster	574.47 s	419.85 MiB	1085.63 MiB
Multiclass Logistic regression	Bare metal – single PC	126.42 s	438.85 MiB	438.85 MiB
	K8s – one node spark cluster	137.73 s	438.67 MiB	438.78 MiB
	K8s – two node spark cluster	88.40 s	259.61 MiB	439.49 MiB
	K8s – four node spark cluster	62.89 s	169.90 MiB	440.29 MiB
Decision tree	Bare metal – single PC	542.44 s	1010.77 MiB	1010.77 MiB
	K8s – one node spark cluster	549.47 s	1011.34 MiB	1011.47 MiB
	K8s – two node spark cluster	344.128 s	598.15 MiB	1012.13 MiB
	K8s – four node spark cluster	249.46 s	391.46 MiB	1012.75 MiB
Naive Bayes	Bare metal – single PC	411.47 s	762.99 MiB	762.99 MiB
	K8s – one node spark cluster	415.95 s	761.92 MiB	762.27 MiB
	K8s – two node spark cluster	263.25 s	451.05 MiB	763.36 MiB
	K8s – four node spark cluster	183.96 s	295.20 MiB	763.42 MiB



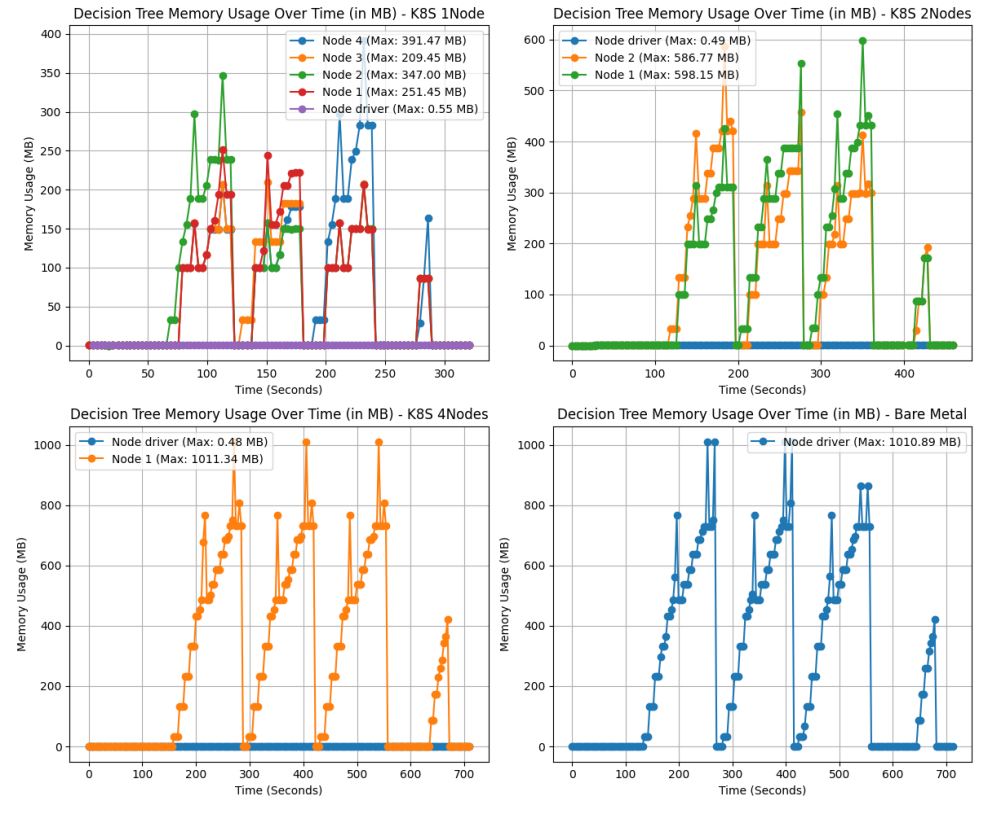
7.1-1: Random forest memory usage



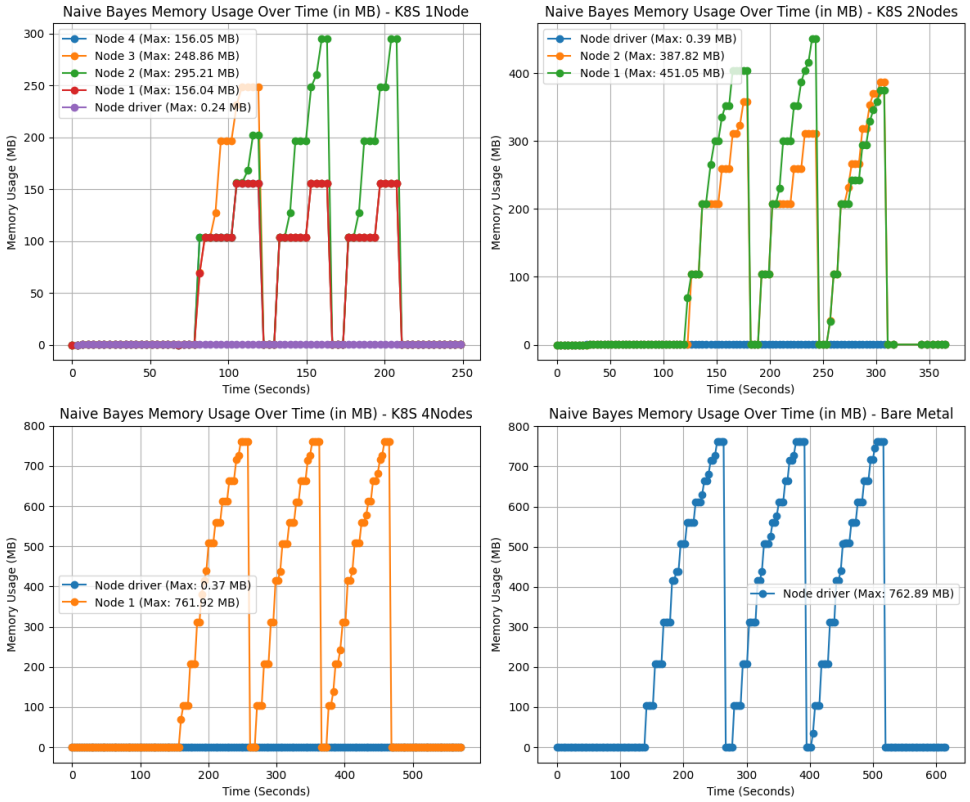
7.1-2: MLP memory usage



7.1-3: MLR memory usage



7.1-4: Decision tree memory usage



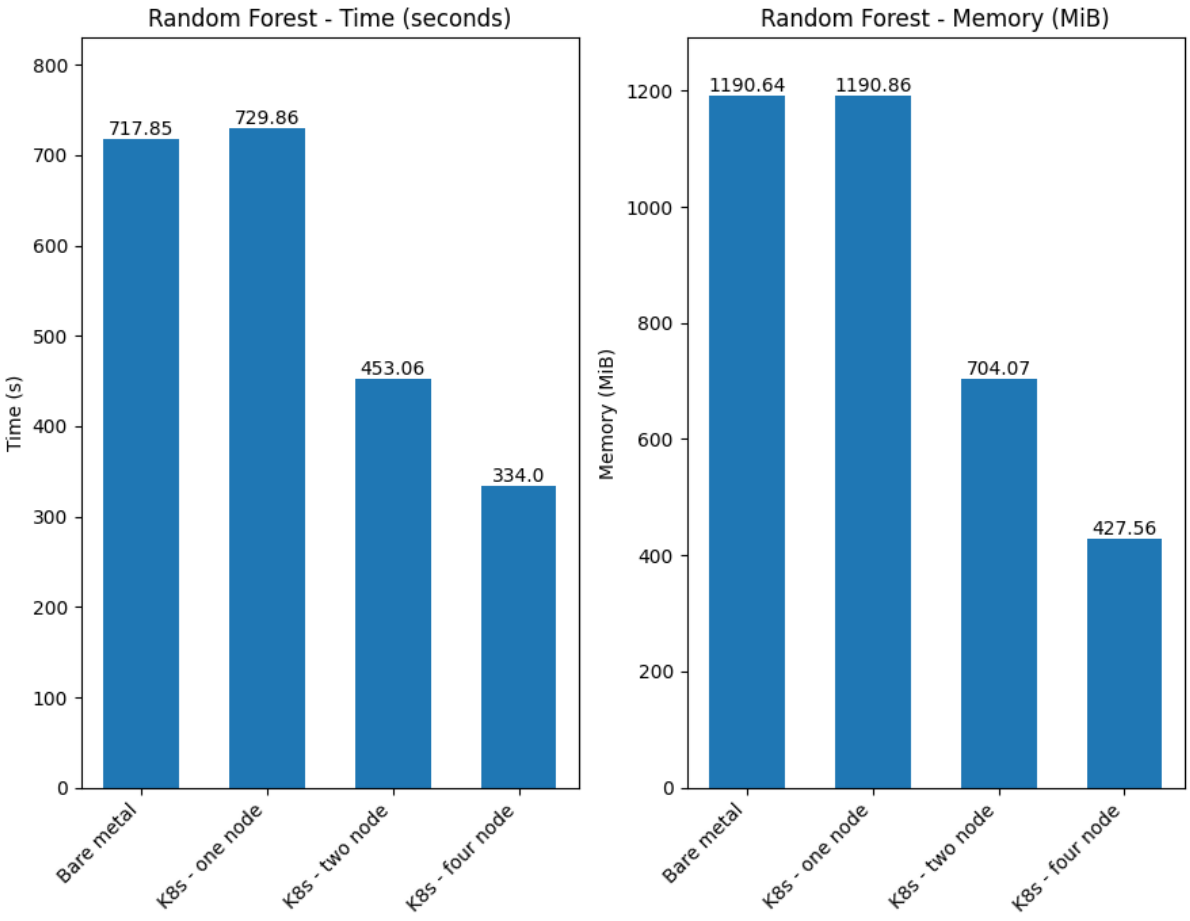
7.1-5: Naive bayes memory usage

The previous table and figures provide a detailed quantitative analysis of the performance differences between containerized (Kubernetes) and bare metal setups in executing various machine learning algorithms on Apache Spark. Observations from these data, including trends in time efficiency, and memory consumption across different configurations, set the stage for a deeper analysis in the following sections.

7.2 Analysis and interpretation

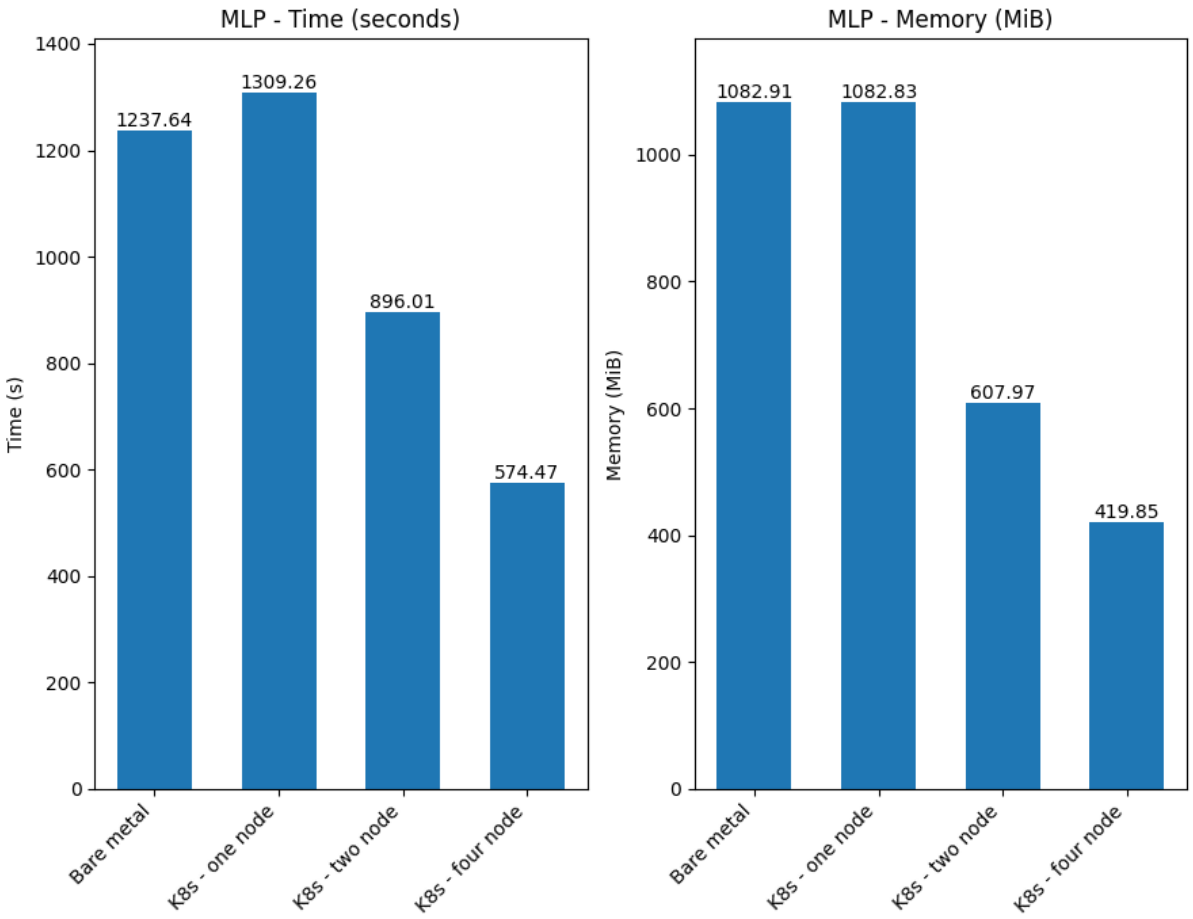
7.2.1 Performance comparison

Starting with the Random Forest algorithm, a noticeable performance enhancement is observed when transitioning from a single PC bare metal setup to various Kubernetes (K8s) configurations. This enhancement is particularly pronounced in the four-node spark cluster configuration. The execution time is significantly reduced from 717.85 seconds on bare metal to just 334.00 seconds on a four-node K8s cluster, reflecting a decrease of approximately 53.5%. However, when comparing the bare metal setup to the K8s one-node cluster, there's only a small increase in execution time to 729.86 seconds, about 1.67% higher. Moreover, memory usage decreases significantly from 1190.64 MiB on bare metal to 427.56 MiB in the four-node K8s cluster, clearly demonstrating the superior memory efficiency achievable in distributed computing setups.



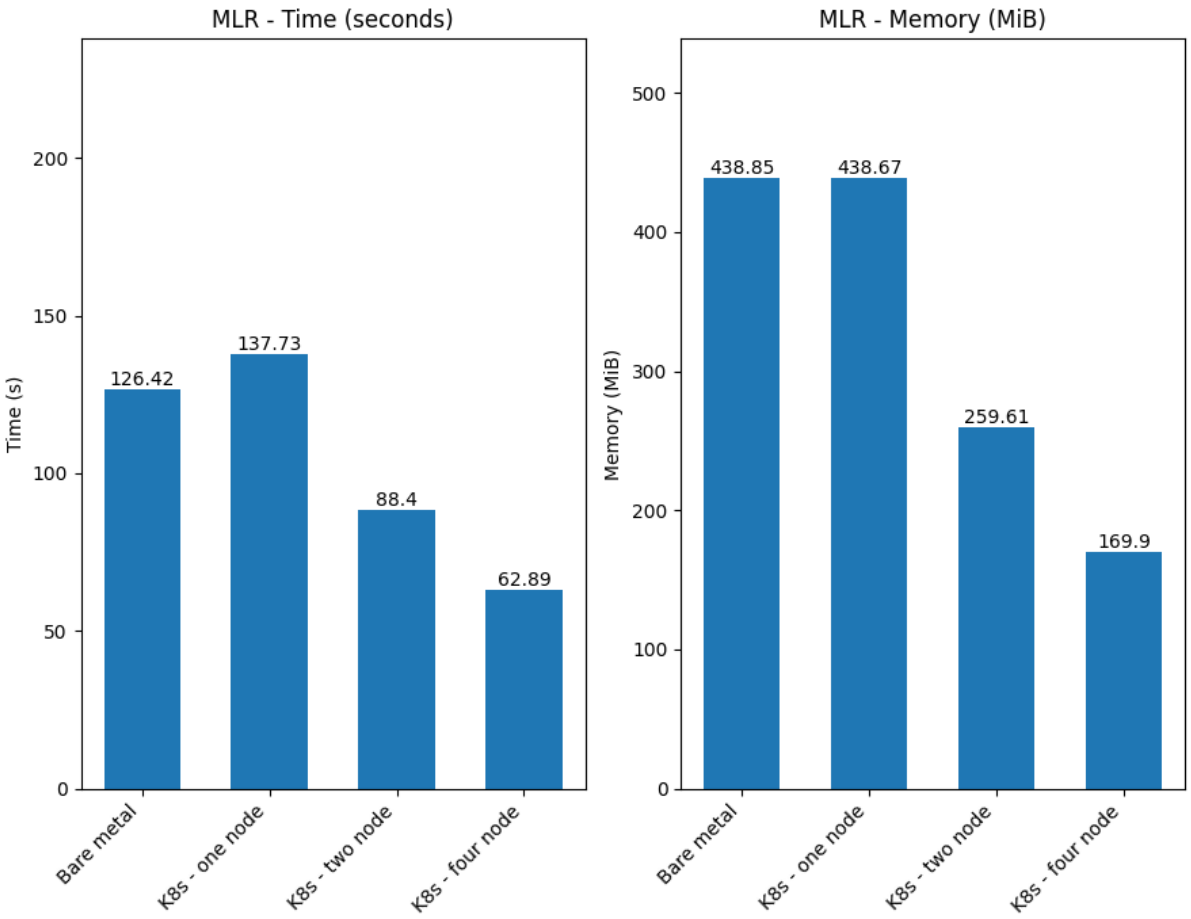
7.2-1: Random Forest bar graphs of results

The Multilayer Perceptron (MLP) algorithm shows a similar trend. The execution time decreases dramatically from 1237.64 seconds on bare metal to 574.47 seconds on the four-node K8s cluster. However, the execution time between the bare metal and the one-node K8s setup shows a negligible increase to 1309.26 seconds. Memory consumption also shows a substantial reduction, dropping from 1082.91 MiB on bare metal to 419.85 MiB on the K8s four-node cluster, which reaffirms the advantages of distributed setups in terms of memory utilization.



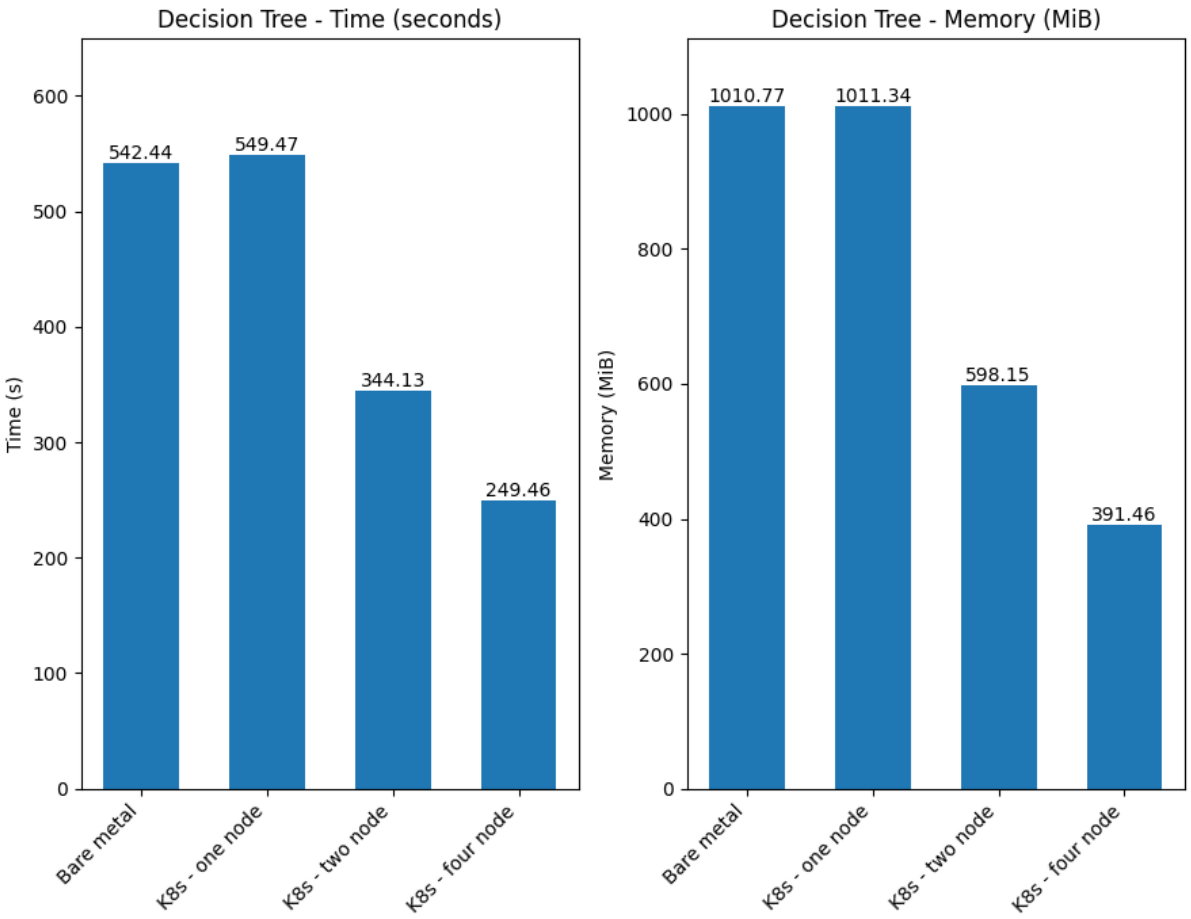
7.2-2: Multilayer perceptron bar graphs of results

For the Multiclass Logistic Regression algorithm, the performance improvement in the four-node setup is significant, with the execution time dropping by about 50.2% from 126.42 seconds on bare metal to just 62.89 seconds. Conversely, when comparing the bare metal setup to the one-node K8s setup, the execution time increases to 137.73 seconds. Similarly, memory usage sees a substantial decrease from 438.85 MiB on bare metal to 169.90 MiB in the four-node K8s cluster, underscoring the capability of distributed computing to optimize memory usage.



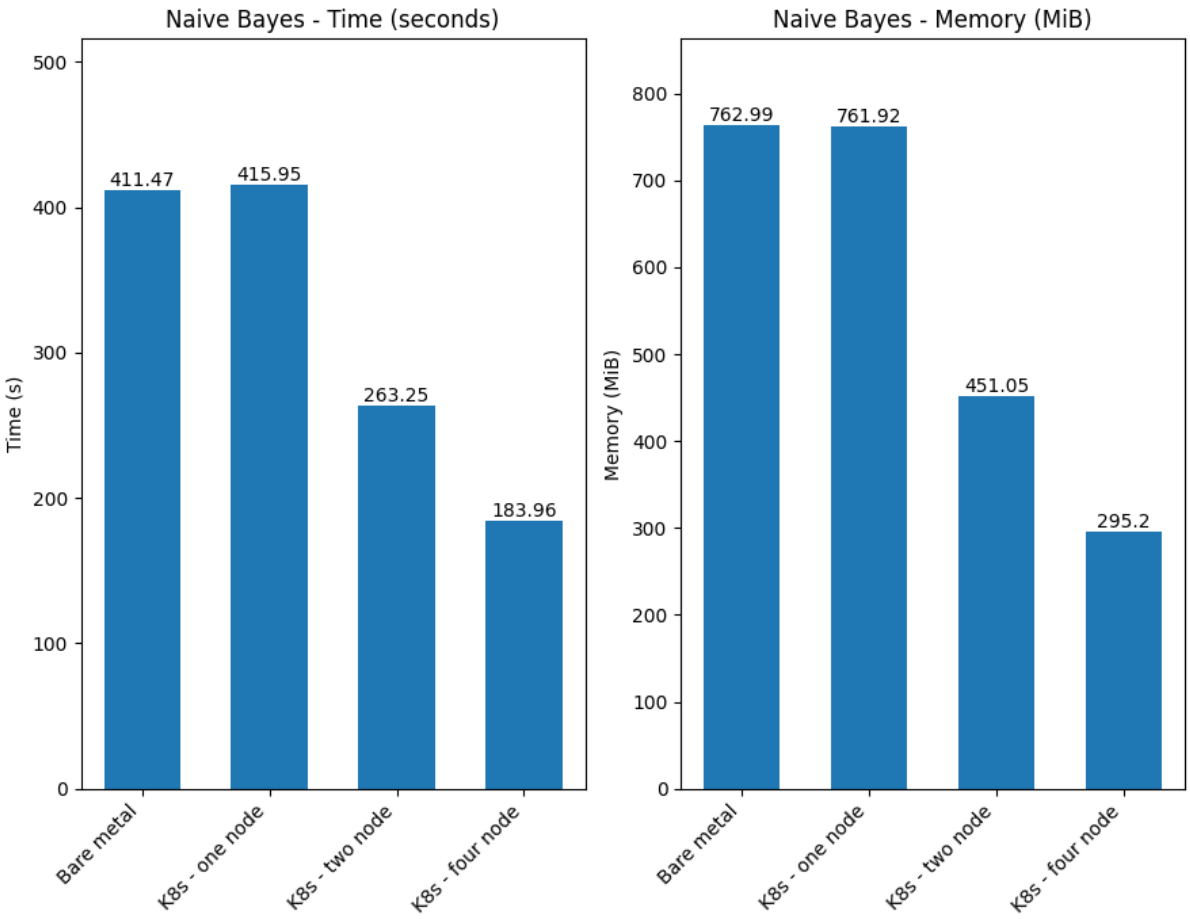
7.2-3: Multiclass logistic regression bar graphs of results

Similarly, the Decision Tree algorithm shows a decrease in execution time from 542.44 seconds to 249.46 seconds (about 54.0%) when comparing bare metal and the K8s four-node cluster. In the one-node K8s setup, the execution time is slightly higher at 549.39 seconds compared to bare metal. Memory usage also decreases significantly from 1010.77 MiB on bare metal to 391.46 MiB in the four-node K8s configuration.



7.2-4: Decision tree bar graphs of results

Lastly, the Naive Bayes algorithm under the K8s one-node setup takes marginally longer to execute at 415.95 seconds compared to 411.47 seconds on bare metal, a 1.1% increase. Memory usage shows a considerable reduction from 762.99 MiB on bare metal to 295.20 MiB in the four-node K8s cluster, demonstrating the efficiency of the multi-node environment in managing memory resources.



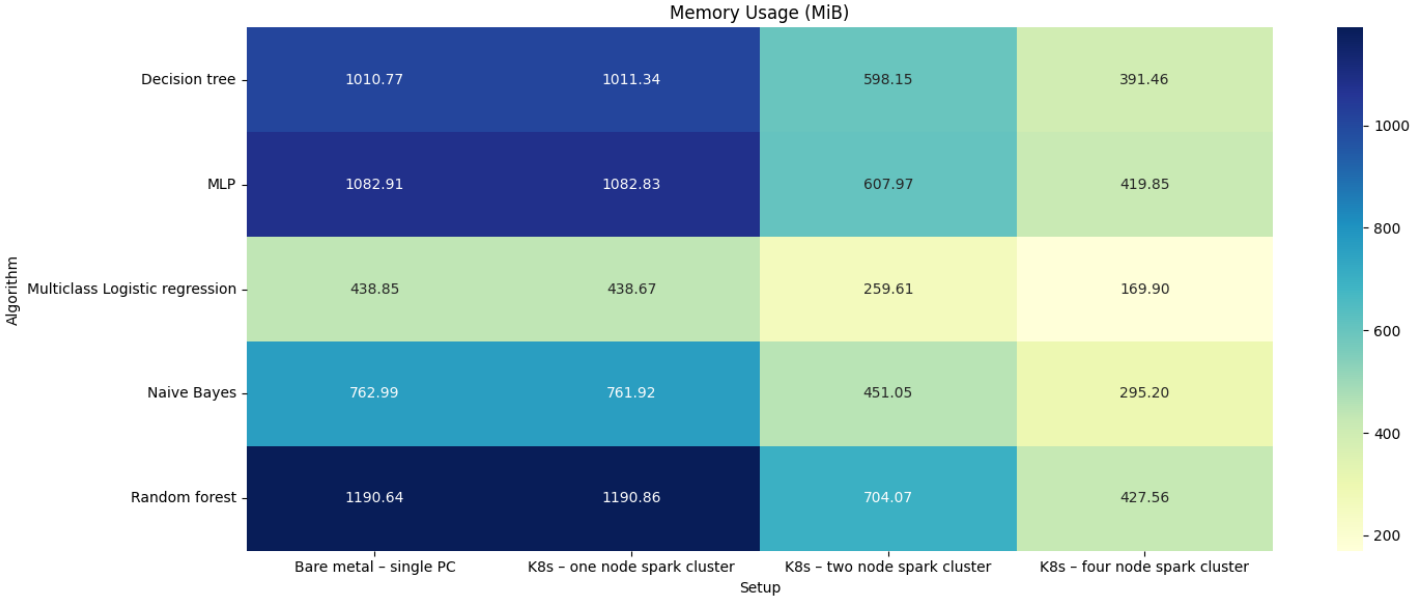
7.2-5: Naive bayes bar graphs of results

In conclusion, the transition from a traditional single PC bare metal setup to distributed computing environments, particularly those based on Kubernetes multi-node clusters, results in substantial performance improvements. These improvements are evident in reduced execution times and memory utilization across various algorithms. The most notable improvements are observed in the four-node K8s cluster configuration, which demonstrates significant enhancements in all aspects of performance. However, even in the one-node K8s setup, where there is a slight increase in execution time for most algorithms, this is offset by a considerable decrease in memory usage. This highlights the effectiveness and efficiency of distributed computing in handling a variety of computational tasks. The results illustrate the advantages of adopting distributed computing frameworks, particularly in scenarios where efficient resource utilization is a priority.

7.2.2 Interpretation and consequences

The transition from a single PC bare metal setup to Kubernetes (K8s) multi-node clusters reveals insights into the dynamics of distributed computing environments. Notably, the marked performance enhancements in multi-node configurations, particularly in the four-node K8s cluster, are evident in terms of reduced execution time and more efficient memory utilization. These improvements highlight the effectiveness and scalability of distributed systems.

An important observation from our data is the considerable decline in memory usage in the multi-node K8s configuration across all tested algorithms, relative to the bare metal setup, shown in the heatmap in figure 7.2-6. The underlying mechanism for this reduction can be associated with distributed computing principles, where tasks are split across multiple nodes, allowing for more efficient memory use. In distributed clusters, parallel processing enables simultaneous execution of tasks, which not only improves computational speed but also optimizes resource utilization. This leads to each node handling a smaller, more manageable portion of data, thereby consuming less memory than a single machine processing the entire dataset. Furthermore, advanced memory management techniques, such as intelligent data caching and load balancing across nodes, contribute to this reduction in memory usage. This distribution of tasks and efficient resource management is particularly evident in the two-node and four-node K8s Spark cluster setups, which consistently show lower memory usage compared to the single-node cluster and bare metal configurations.

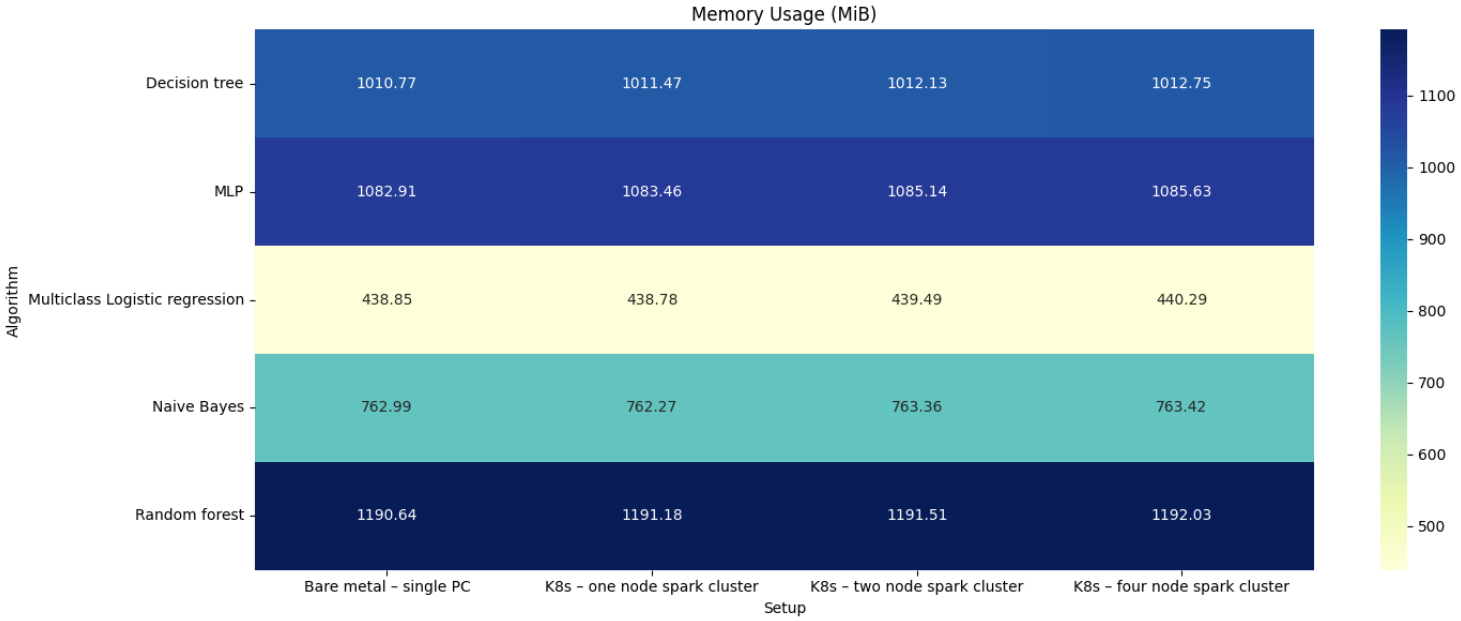


7.2-6: Memory Usage Heatmap

Adding to this, an examination of the "total memory" column in our results table reveals that the total memory utilized across all nodes remains consistent across different configurations. This stability in total memory usage, even as the number of nodes increases from one to four, suggests that K8s's approach to distributed computing does not introduce memory overhead. This indicates that horizontal scaling through K8s can be achieved without incurring additional memory costs. This feature is invaluable in processing large-scale data or machine learning workflows where efficient memory management is paramount.

The consistent total memory usage across various node configurations underscores the efficiency of K8s in resource management. It demonstrates K8s's capability to distribute workload effectively without compromising on memory efficiency. This finding is particularly reassuring for applications that demand high scalability and performance without the penalty of increased resource consumption. The efficiency of K8s in managing memory across a distributed environment, as evidenced by our data, supports its adoption for deploying complex applications that require scalable, efficient, and cost-effective memory usage. Furthermore, this observation reinforces the notion that distributed computing, especially in a K8s environment, is also about optimizing resource allocation and usage. The evidence of our experiments bolsters the argument that K8s facilitates an intelligent and efficient use of resources. This optimization is vital in big data and machine learning domains, where judicious

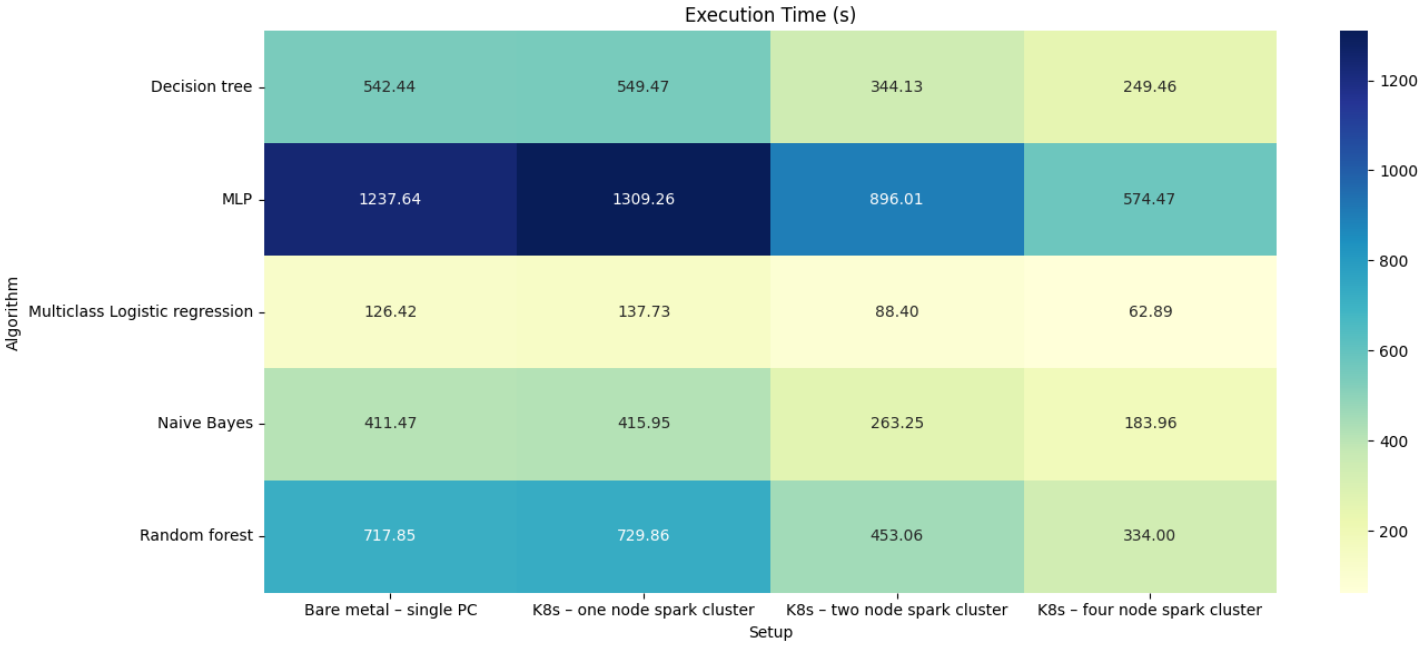
memory usage can significantly enhance performance and enable the processing of larger datasets more effectively.



7.2-1: Total memory consumption heatmap

The significant reduction in execution time observed in multi-node Kubernetes (K8s) setups, as shown in the heatmap in figure 7.2-7, has profound implications for the efficiency and scalability of computational processes in distributed computing environments. In practical terms, the reduction in execution time enables more complex computations to be completed in shorter periods. Furthermore, this efficiency gain underscores the value of Kubernetes in optimizing computational resources across multiple nodes, which is essential for applications that demand high throughput and low latency. By dramatically decreasing execution times, Kubernetes multi-node clusters not only enhance the operational efficiency of computing tasks but also enable a higher throughput of workloads. In addition, the scalability afforded by Kubernetes means that as computational demands increase, resources can be dynamically allocated to meet these needs without a corresponding increase in execution time. This scalability is a key factor in the cost-effective management of computational resources, as it allows for the precise tuning of resource allocation to match the workload, thereby avoiding underutilization or over-provisioning.

However, this improvement comes with a nuanced understanding of the initial overheads encountered during the transition from traditional bare metal setups to containerized environments. Specifically, the marginal increase in execution time in transitioning to a one-node K8s setup, when compared to the bare metal configuration, highlights the inherent complexities of container orchestration. This initial overhead can be attributed to factors such as container startup time, the abstraction layer introduced by Kubernetes, and the resource management overhead that comes with it. Despite this slight increase, the overarching trend towards significantly reduced execution times in larger multi-node configurations underscores the scalability and efficiency benefits of Kubernetes for distributed computing. The nuanced increase in execution time from bare metal to a one-node K8s setup underscores the importance of considering the trade-offs between immediate performance and long-term scalability and efficiency. While the initial setup in Kubernetes may introduce minor inefficiencies, these are quickly offset by the substantial gains in resource optimization and execution speed as the system scales.



7.2-7: Execution time Heatmap

Our analysis demonstrates that Kubernetes, especially in multi-node configurations, provides a more efficient, scalable, and resource-optimized environment for distributed computing with Apache Spark, compared to traditional Bare Metal setups. However, this efficiency requires more sophisticated setup and management, which should be considered against the performance improvements in practical applications. Understanding these trade-offs and the behavior of different algorithms in various environments is essential for effectively leveraging distributed computing with Apache Spark.

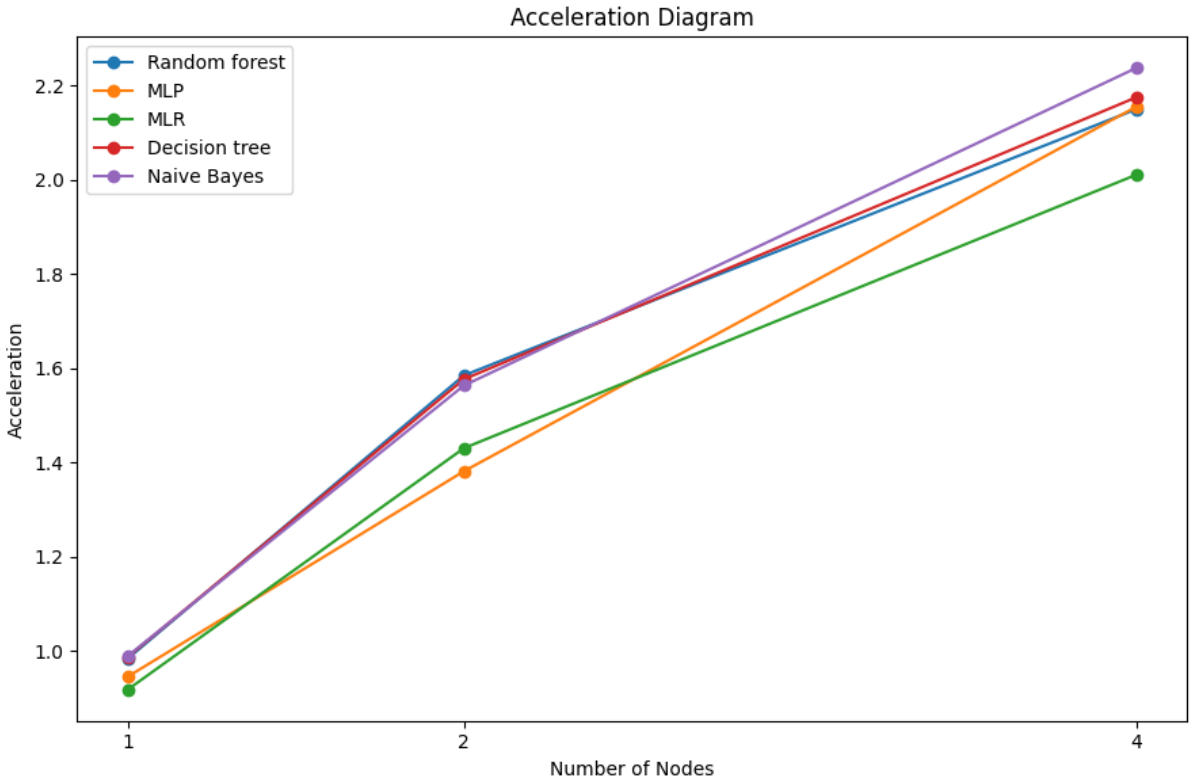
To summarize, the performance benefits of using distributed computing environments, such as Kubernetes with Apache Spark, are clear, but the optimal setup depends on factors like algorithm choice and scalability requirements. This analysis highlights the need for careful consideration and testing when selecting the appropriate infrastructure for distributed computing with Apache Spark. The shift towards distributed computing frameworks like Apache Spark on Kubernetes multi-node clusters signifies a notable advancement in efficiently and sustainably handling complex computational tasks.

7.2.3 Comparison based on the types of models

In the acceleration diagram below we can see that in the Kubernetes 4 nodes spark cluster we have an acceleration from about 2 to 2.2 depending on the algorithm. The time does not sub-quadrupled because increasing the number of nodes brings with it a cost of coordination and communication. This is known as Amdahl's law, which states that the maximum speedup of a program is limited by the execution time of the non-distributed part of the code. More specifically, even if we add more nodes, some tasks must be executed serially, and so there is a limit to the acceleration that can be completed. Beyond that, the overhead from orchestrating the containers increases further the constraint of distributed execution.

The observable variations in acceleration for the various machine learning algorithms can be ascribed to the unique characteristics and computational demands of each. The Decision Tree algorithm may be less sensitive to orchestration changes, exhibiting noticeable acceleration in execution. Similarly, the Naive Bayes, which is computationally less demanding, also shows significant

improvement. In contrast, the Multilayer Perceptron, which has higher computational requirements, displays smaller acceleration. Also, the type of computations involved in each algorithm can influence its response to containerized environments. Algorithms heavily reliant on iterative processes or intensive computations may exhibit more pronounced effects from orchestration overhead, as seen in the Multiclass Logistic Regression model. The Random Forest algorithm, with its ensemble approach, also experiences a discernible increase in acceleration, yet this is moderated by the complexity of its structure. This nuanced behavior across algorithms highlights the importance of considering both algorithmic efficiency and the computational environment when optimizing for performance in containerized systems.



7.2-8: Acceleration diagram

7.2.4 Memory fluctuations

In the graphs 7.1-1, 7.1-2, 7.1-3, 7.1-4 and 7.1-5 we see the memory usage over time with fluctuations and occasional peaks. This variability in memory usage is typically caused by data loading, garbage collection and cashing. When the model loads data into memory, there's a spike in memory usage which then drops once the data is either moved out of memory or no longer in use. Also, intermediate results or data might be cached in memory during processing. Once the cache is cleared or written to disk, memory usage may decrease. Furthermore, in programming languages like Java and Python, garbage collection (GC) is a form of automatic memory management. The garbage collector attempts to reclaim memory occupied by objects that are no longer in use by the program. When GC kicks in, memory usage can drop suddenly. This could explain the sharp drops in memory usage observed in the graphs.

8 Discussion

8.1 Performance

The empirical data from our study highlights the superior performance of containerized environments, particularly Kubernetes multi-node clusters, over traditional Bare Metal setups when executing various machine learning algorithms using Apache Spark.

- **Execution Speed:** A key finding in our study is the notable reduction in execution time across all algorithms when transitioning from Bare Metal to Kubernetes clusters, in multi-node setups. For instance, in the four-node Kubernetes configuration, the execution time for algorithms like the Multilayer Perceptron and Decision Tree saw reductions of more than 50%. This decrease is attributed to the distributed processing capabilities of Kubernetes clusters, which allows for simultaneous execution of tasks across multiple nodes, thereby optimizing computational efficiency. However, when transitioning from Bare Metal to Kubernetes clusters, in one-node setup, we observe a slightly worse execution speed, ranging from 1.6% to 8.95%.
- **Memory Usage:** Memory efficiency is another critical performance aspect where Kubernetes clusters demonstrate superiority. Our findings indicate a consistent decrease in memory usage when moving from a Bare Metal setup to a Kubernetes multi-node cluster. This suggests that Kubernetes, through its containerization technology, effectively manages and allocates memory resources across its nodes, reducing the overall memory footprint for similar workloads.

Furthermore, Kubernetes clusters, particularly those with multiple nodes, have demonstrated their capacity for scalability and resource optimization, crucial for high-performance computing tasks in distributed environments. The distributed nature of Kubernetes facilitates the handling of larger and more complex tasks by spreading the load across multiple nodes. This scalability is particularly beneficial for data-intensive applications, where handling large volumes of data efficiently is paramount.

The study also highlights the resource optimization capabilities of Kubernetes. The efficient allocation and utilization of memory resources in multi-node Kubernetes clusters imply enhancements in overall system performance, especially when compared to the more rigid and resource-intensive Bare Metal setups. These performance gains have implications for applications where time efficiency and resource optimization are critical. Kubernetes clusters emerge as a preferable choice for distributed processing tasks, given their superior performance metrics in memory.

However, it is important to note that these performance enhancements are not universally applicable. The choice of configuration, the nature of the computational task, and the specific requirements of the application should be carefully considered to harness the full potential of Kubernetes in distributed computing environments.

8.2 Performance - Portability

Kubernetes, known for its portability and scalability in container orchestration, offers benefits for cloud-native applications. However, this comes at the cost of potential performance drawbacks, especially in smaller configurations like a one-node setup.

Portability in distributed computing refers to the ease with which applications can be moved and executed across different computing environments. Containerization, epitomized by Kubernetes, has significantly advanced the portability of applications. It allows for a consistent and predictable

deployment environment, irrespective of the underlying hardware or infrastructure. This feature is particularly advantageous in cloud-native applications, where the ability to migrate workloads seamlessly across various cloud providers or between on-premises and cloud environments is essential.

While portability offers flexibility and adaptability, it can come at a cost to raw performance metrics. Containerization abstracts the application from the hardware, introducing a layer of overhead that can impact performance. In Kubernetes, this overhead is manifest in container orchestration, network configuration, and resource allocation strategies, which can slightly degrade performance compared to running applications directly on bare metal. For instance, in a one-node Kubernetes setup, the ease of scaling and deployment is counterbalanced by a marginal increase in execution time and resource utilization, as observed in our empirical data. This trade-off is critical in contexts where peak performance is paramount, such as in high-performance computing tasks that necessitate rapid data processing.

The decision to opt for a containerized environment or a bare metal setup, hinges on a thorough understanding of the application's specific requirements. If the priority is on flexibility, scalability, and ease of deployment, containerization presents a compelling case. Conversely, for applications where maximum performance is non-negotiable, bare metal setups might be more appropriate. In our study, while Kubernetes demonstrated superior resource management capabilities, especially in multi-node configurations, it is essential to note that these benefits come with increased complexity in system administration. The management overhead, including the need for specialized knowledge in container orchestration and network configurations, can be significant.

The choice between containerization and bare metal should also consider the nature of the application. For example, applications that experience fluctuating workloads and require rapid scaling will benefit more from a Kubernetes environment. In contrast, applications with consistent performance demands and less need for scalability might find bare metal setups more efficient. Looking at the broader picture, the trade-off between performance and portability also has long-term implications for organizational strategy and sustainability. Kubernetes' ability to optimize resource utilization can translate into cost savings and reduced environmental impact over time. However, this must be weighed against the immediate performance needs and the technical capacity of the organization to manage a Kubernetes environment.

Choosing the right infrastructure for an application depends on a careful evaluation of the application's specific requirements. For applications where flexibility and adaptability are paramount, Kubernetes and containerized environments are beneficial. Conversely, for applications with stringent performance requirements, Bare Metal configurations might be more suitable.

8.3 Management overhead

While Kubernetes offers advantages in scalability and portability, it introduces a certain level of complexity and management overhead. This overhead and its impact on the cost-effectiveness and practicality of Kubernetes solutions have implications. Its complexity stems from various factors, including the orchestration of containers, the management of cluster resources, network configurations, and ensuring high availability and fault tolerance. Effective management of a Kubernetes environment requires a robust understanding of these components and their interplay, which often necessitates specialized skills and knowledge (Toka et al., 2021).

One of the primary challenges in managing Kubernetes environments is the dynamic nature of resource allocation and monitoring (Kim et al., 2021). Kubernetes excels in efficiently allocating resources across multiple nodes and containers. However, this requires continuous monitoring and fine-tuning to ensure optimal performance. Administrators must oversee the distribution of memory, and storage resources, manage pod scheduling, and monitor the health of the entire cluster. This level

of oversight demands dedicated tools and a proactive management approach. Another critical aspect of management overhead is maintaining high availability and effective load balancing in a Kubernetes environment. Kubernetes provides mechanisms for load balancing and ensures that applications are always available, even in the event of node failures. However, configuring and managing these features requires careful planning and ongoing management to prevent service disruptions and maintain consistent performance levels (Jorge-Martinez et al., 2021).

Network configuration in a distributed computing environment is complex, and Kubernetes adds layers to this complexity with its own networking model. Network policies and ensuring secure communication between pods are pivotal for maintaining a robust and secure environment. Additionally, implementing and managing security measures such as role-based access control (RBAC) (Shamim et al., 2020), secrets management, and compliance with data security standards form an integral part of the management overhead. The sophisticated nature of Kubernetes necessitates a higher level of expertise in container orchestration, network management, and security (Budigiri et al., 2021). Organizations must invest in training their IT staff or hiring specialists with the requisite skills. This investment in human capital is a significant aspect of management overhead and impacts on the overall cost and feasibility of adopting Kubernetes for distributed computing. To manage the complexity of Kubernetes, organizations often rely on a suite of automation tools and platforms. These tools aid in deployment, monitoring, scaling, and managing the lifecycle of applications.

While these tools streamline management processes, selecting the right tools, integrating them into existing systems, and maintaining them adds another layer to the management overhead. Kubernetes environments are dynamic and require regular updates and upgrades to ensure security, performance, and feature enhancements. Managing these updates, while ensuring minimal disruption to services, is a challenging aspect of Kubernetes administration. It requires careful planning, testing, and execution, further adding to the management workload (He, 2020).

The management overhead associated with Kubernetes has direct cost implications. These costs include the investment in specialized personnel, training, tooling, and the time spent on administration and maintenance tasks (Nguyen et al., 2020). Organizations must consider these costs when evaluating the total cost of ownership of a Kubernetes environment.

While Kubernetes introduces substantial management overhead, it's essential to balance this with the operational benefits it provides. Kubernetes enables scalable, flexible, and efficient distributed computing environments. The decision to adopt Kubernetes should factor in the long-term operational efficiencies, potential cost savings, and the strategic benefits of having a scalable and robust distributed computing platform. Organizations considering Kubernetes must evaluate their capacity to manage these environments effectively. This includes investing in the necessary tools, expertise, and resources to handle the intricacies of Kubernetes management. The management overhead of Kubernetes can influence its overall cost-effectiveness, especially for applications that do not require its full scale and flexibility. Organizations must assess the total cost of ownership, including management and maintenance, when deciding on the suitability of Kubernetes for their specific applications.

Ultimately, the choice between Kubernetes and Bare Metal is not just about performance and portability but also about an organization's ability to manage and maintain the chosen infrastructure effectively. This decision should be grounded in a comprehensive understanding of the application's needs, the organization's management capacity, and the long-term implications of adopting a particular computing environment.

9 Conclusions

Considering the findings from our comprehensive analysis, the implementation of future dashboards using a distributed computing approach is recommended, particularly for scenarios where scalability and flexibility are key requirements. While distributed computing environments, especially those managed by Kubernetes, have demonstrated potential for enhanced scalability and improved resource utilization, their performance compared to traditional bare metal setups can be context dependent. In certain configurations, particularly in multi-node setups, distributed computing has shown promising improvements in processing efficiency. However, it's important to acknowledge that in some instances, especially in smaller-scale deployments, the performance gains might not be as significant due to the overhead introduced by containerization and orchestration. Therefore, our recommendation to adopt a distributed computing approach for future dashboards is particularly aimed at applications where the advantages of scalability, flexibility, and resource optimization align with the specific needs and constraints of the task at hand.

9.1 Summary

This thesis has undertaken a comprehensive exploration into the realms of distributed computing, focusing on the performance of Apache Spark in containerized versus bare metal environments. The thesis began with a theoretical background, laying the foundational concepts of distributed computing, Apache Spark, containerization, virtualization, and bare metal implementations. It progressed through a detailed analysis of various deployment strategies, empirical experiments, and interpretation of the results.

The empirical study, forming the core of this thesis, revealed insightful findings. Key among them is the superior performance of containerized environments, particularly Kubernetes multi-node clusters, compared to traditional bare metal setups. This superiority manifested in reduced execution times, and more efficient memory management across a range of machine learning algorithms. Notably, the enhancements in performance metrics were most pronounced in multi-node Kubernetes configurations.

However, the study also highlighted the nuances of containerized environments. While Kubernetes excelled in scalability and portability, it introduced a complexity in management and an overhead in smaller-scale setups, like the one-node cluster. This observation establishes a crucial finding: the choice between containerization and bare metal is not clear-cut but depends on specific use cases, balancing the trade-offs between performance, portability, and management overhead.

The thesis also delved into the practical aspects of deploying Apache Spark in different computing environments. It offered insights into the implications of these setups in real-world applications, particularly emphasizing the importance of understanding the specific requirements of an application and the capability of an organization to manage the chosen infrastructure.

In summary, this thesis contributes to the field of distributed computing by providing a detailed comparative analysis of containerization and bare metal environments using Apache Spark. It offers a subtle understanding of the trade-offs involved in different deployment strategies, guiding practitioners and researchers in making informed decisions for their specific computational needs.

9.2 Trade-off between performance and portability – management overhead

The thesis highlighted a trade-off in distributed computing environments: the balance between performance and portability versus the management overhead. While Kubernetes-based containerized environments offered enhanced portability and scalability, they also introduced a significant management overhead compared to bare metal setups.

9.2.1 For a specific zoom level

In exploring the trade-off between performance and portability in distributed computing environments, it becomes essential to focus on a specific zoom level, one that encapsulates the details and dynamics of these environments.

At this level, we observe that while Kubernetes excels in providing a highly portable and scalable environment, it also introduces a level of complexity and management overhead that cannot be overlooked. This is particularly evident in the one-node Kubernetes cluster setup, where the benefits of containerization become a trade-off. On the one hand, Kubernetes offers the flexibility to seamlessly deploy applications across diverse environments, a feature that is invaluable in today's rapidly evolving technological landscape. On the other hand, this flexibility comes with a cost – a slight decrease in raw performance metrics compared to traditional bare metal setups. This performance dip, though small, is critical to consider in contexts where every millisecond of computation time counts. In high-performance computing tasks, for example, where processing large volumes of data in the shortest time possible is crucial, the overhead introduced by containerization might be a significant factor. However, for applications where scalability and the ability to adapt to different environments are more important than the sheer speed of execution, Kubernetes presents a compelling option.

Furthermore, when delving into the specifics of resource utilization, Kubernetes demonstrates its strength in efficient resource management. In a one-node setup, the platform's ability to manage resources dynamically becomes a key advantage, particularly in scenarios where resource constraints are a major concern. Kubernetes' scheduling and resource allocation mechanisms allow for more efficient use of available computational resources, which can lead to cost savings and improved overall system sustainability.

However, this efficient resource management comes at the cost of increased complexity in system administration. Kubernetes requires an understanding of container orchestration, network configurations, and resource allocation strategies. This complexity necessitates a higher level of expertise and potentially more sophisticated monitoring and management tools, which could suggest challenges for organizations without the required technical capabilities.

The specific zoom level analysis also highlights the importance of application-specific considerations in choosing a deployment strategy. Depending on the nature and requirements of the application, the balance between performance, portability, and management overhead can vary. For instance, applications that require rapid scaling to handle fluctuating workloads would benefit more from a Kubernetes-based environment, despite the potential performance overhead. In contrast, applications that demand the highest level of performance and are less sensitive to scaling requirements might be better served by a bare metal setup.

In conclusion, the analysis at this specific zoom level underscores the need for a tailored approach to choosing between containerized and bare metal environments. The decision should be based on a thorough understanding of the application's requirements, the organization's technical capabilities, and the specific trade-offs involved in each deployment strategy. By carefully considering these factors,

organizations can make informed decisions that align with their strategic objectives and operational constraints, ensuring the optimal use of their computational resources.

9.3 Future suggestions

Future research could benefit from delving into the development of strategies aimed at minimizing the performance overhead observed in Kubernetes environments, especially in configurations with a smaller scale. Such strategies could potentially balance the high scalability and portability of containerized systems and the performance-centric nature of bare metal setups.

To understand the scalability limits and performance implications comprehensively, it would be instructive to expand experimental studies across more nodes, extending the current investigation from up to 24 nodes to larger clusters. Observing the behavior in larger deployments could provide insights into when and if the performance benefits plateau or decline, offering valuable data for optimizing container orchestration across varying scales.

Another promising area for future exploration is the advancement of resource management techniques. These techniques could further boost the efficiency of containerized environments, broadening their applicability across a diverse range of computational tasks and scenarios. By enhancing resource management, containerized systems could potentially resemble the performance efficiencies of bare metal setups while retaining their inherent benefits of flexibility and scalability.

The exploration of hybrid approaches also presents significant potential for future research. These approaches would aim to combine the performance benefits of bare metal environments with the scalability and portability advantages of containerization. Such a hybrid model could offer a balanced solution, tailor-made for applications requiring both high performance and the ability to scale across varied environments. Additionally, there is an opportunity in conducting application-specific studies. These studies would provide a deeper understanding of how different deployment strategies impact various domains, such as real-time data processing or high-performance computing. Insights from these studies could guide the development of specialized deployment strategies that are optimized for specific types of applications, thus ensuring optimal performance and efficiency.

Lastly, the development of improved management tools and practices for containerized environments stands as a critical need. The complexity and management overhead associated with these environments, as highlighted in this thesis, underscore the necessity for more intuitive and efficient management solutions. By simplifying the operational aspects of containerized systems, these tools and practices could significantly make it easier to adopt them, making them more accessible and practical for a wider range of users and applications.

Bibliography

Alnafessah, A., & Casale, G. (2020). Artificial neural networks based techniques for anomaly detection in Apache Spark. *Cluster Computing*, *23*(2), 1345–1360. https://doi.org/10.1007/s10586-019-02998-y

Bellavista, P., & Zanni, A. (2017, January 5). Feasibility of fog computing deployment based on docker containerization over RaspberryPi. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/3007748.3007777

Beltre, A. M., Saha, P., Govindaraju, M., Younge, A., & Grant, R. E. (2019). Enabling HPC Workloads on Cloud Infrastructure Using Kubernetes Container Orchestration Mechanisms. *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, 11–20. https://doi.org/10.1109/CANOPIE-HPC49598.2019.00007

Bhardwaj, A., & Krishna, C. R. (2021). Virtualization in Cloud Computing: Moving from Hypervisor to Containerization—A Survey. *Arabian Journal for Science and Engineering*, *46*(9), 8585–8601. https://doi.org/10.1007/s13369-021-05553-3

Bhat, S. (2018). Practical Docker with Python. In *Practical Docker with Python*. Apress. https://doi.org/10.1007/978-1-4842-3784-7

Bhimani, J., Yang, Z., Leeser, M., & Mi, N. (2017). Accelerating big data applications using lightweight virtualization framework on enterprise cloud. *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, 1–7. https://doi.org/10.1109/HPEC.2017.8091086

Budigiri, G., Baumann, C., Mühlberg, J. T., Truyen, E., & Joosen, W. (2021). Network Policies in Kubernetes: Performance Evaluation and Security Analysis. *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, 407–412. https://doi.org/10.1109/EuCNC/6GSummit51104.2021.9482526

Campbell, S., & Jeronimo, M. (n.d.). *An Introduction to Virtualization*.

Casalicchio, E., & Iannucci, S. (n.d.). *The State-of-the-Art in Container Technologies: Application, Orchestration and Security*. https://aws.amazon.com/ecs

Chaisawat, S., & Vorakulpipat, C. (2020). Fault-Tolerant Architecture Design for Blockchain-Based Electronics Voting System. *2020 17th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 116–121. https://doi.org/10.1109/JCSSE49651.2020.9268264

Chang, B. R., Tsai, H.-F., & Wang, Y.-A. (2016). Optimized Multiple Platforms for Big Data Analysis. *2016 IEEE Second International Conference on Multimedia Big Data (BigMM)*, 155–158. https://doi.org/10.1109/BigMM.2016.61

Chen, S., & Wang, W. (2009). Decision tree learning for freeway automatic incident detection. *Expert Systems with Applications*, *36*(2, Part 2), 4101–4105. https://doi.org/https://doi.org/10.1016/j.eswa.2008.03.012

Choi, J. Y., Cho, M., & Kim, J. S. (2021). Employing vertical elasticity for efficient big data processing in container-based cloud environments. *Applied Sciences (Switzerland)*, *11*(13). https://doi.org/10.3390/app11136200

Clements, A. A., Almakhdhub, N. S., Saab, K. S., Srivastava, P., Koo, J., Bagchi, S., & Payer, M. (n.d.). *Protecting Bare-metal Embedded Systems With Privilege Overlays*.

Duplyakin, D., Uta, A., Maricq, A., & Ricci, R. (2020). In Datacenter Performance, The Only Constant Is Change. *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, 370–379. https://doi.org/10.1109/CCGrid49817.2020.00-56

E X P E R T O P I N I O N 8. (2009). www.computer.org/intelligent

Eze, K. G., & Akujuobi, C. M. (2022). Design and Evaluation of a Distributed Security Framework for the Internet of Things. *Journal of Signal and Information Processing*, *13*(01), 1–23. https://doi.org/10.4236/jsip.2022.131001

Galakatos, A., Crotty, A., & Kraska, T. (2017). Distributed Machine Learning. In *Encyclopedia of Database Systems* (pp. 1–6). Springer New York. https://doi.org/10.1007/978-1-4899-7993-3_80647-1

He, Z. (2020). Novel Container Cloud Elastic Scaling Strategy based on Kubernetes. *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, 1400–1404. https://doi.org/10.1109/ITOEC49072.2020.9141552

Horchulhack, P., Viegas, E. K., & Santin, A. O. (2022). Detection of Service Provider Hardware Over-commitment in Container Orchestration Environments. *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, 6354–6359. https://doi.org/10.1109/GLOBECOM48099.2022.10001375

Hou, J., Zhu, Y., Du, S., & Song, S. (2019). Design and implementation of reconfigurable acceleration for in-memory distributed big data computing. *Future Generation Computer Systems*, *92*, 68–75. https://doi.org/https://doi.org/10.1016/j.future.2018.09.049

Jiang, H., & Deng, H. (2020). Traffic Incident Detection Method Based on Factor Analysis and Weighted Random Forest. *IEEE Access*, *8*, 168394–168404. https://doi.org/10.1109/ACCESS.2020.3023961

Jorge-Martinez, D., Butt, S. A., Onyema, E. M., Chakraborty, C., Shaheen, Q., De-La-Hoz-Franco, E., & Ariza-Colpas, P. (2021). Artificial intelligence-based Kubernetes container for scheduling nodes of energy composition. *International Journal of System Assurance Engineering and Management*. https://doi.org/10.1007/s13198-021-01195-8

K, S., & G, S. (2022). Improvement in Performance of Image Classification based on Apache Spark. *2022 2nd Asian Conference on Innovation in Technology (ASIANCON)*, 1–6. https://doi.org/10.1109/ASIANCON55314.2022.9909293

Kim, E., Lee, K., & Yoo, C. (2021). On the Resource Management of Kubernetes. *2021 International Conference on Information Networking (ICOIN)*, 154–158. https://doi.org/10.1109/ICOIN50884.2021.9333977

Kongkhaensarn, T., & Piantanakulchai, M. (2018). Comparison of probabilistic neural network with multilayer perceptron and support vector machine for detecting traffic incident on expressway based on simulation data. *2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 1–6. https://doi.org/10.1109/JCSSE.2018.8457369

Kratzke, N. (2018). About the Complexity to Transfer Cloud Applications at Runtime and How Container Platforms Can Contribute? In D. Ferguson, V. M. Muñoz, J. Cardoso, M. Helfert, & C. Pahl (Eds.), *Cloud Computing and Service Science* (pp. 19–45). Springer International Publishing.

Kshemkalyani, A. D., & Singhal, M. (n.d.). *Distributed Computing: Principles, Algorithms, and Systems*.

Kumar, M., & Kaur, G. (2022). Containerized MPI Application on InfiniBand based HPC: An Empirical Study. *2022 3rd International Conference for Emerging Technology (INCET)*, 1–6. https://doi.org/10.1109/INCET54531.2022.9824366

Lee, H., & Fox, G. (2019). Big Data Benchmarks of High-Performance Storage Systems on Commercial Bare Metal Clouds. *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, 1–8. https://doi.org/10.1109/CLOUD.2019.00014

Li, Z., Han, M., Wu, S., & Weng, C. (2021). ShadowVM: accelerating data plane for data analytics with bare metal CPUs and GPUs. *Proceedings of the 26th ACM SIGPLAN Symposium on*

Principles and Practice of Parallel Programming, 147–160. https://doi.org/10.1145/3437801.3441595

Liu, P., & Guitart, J. (2021). Performance comparison of multi-container deployment schemes for HPC workloads: an empirical study. *The Journal of Supercomputing*, *77*(6), 6273–6312. https://doi.org/10.1007/s11227-020-03518-1

Liu, Q., Lu, J., Chen, S., & Zhao, K. (2014). Multiple Naïve Bayes classifiers ensemble for traffic incident detection. *Mathematical Problems in Engineering*, *2014*. https://doi.org/10.1155/2014/383671

Lokuciejewski, R., Schüssele, D., Wilhelm, F., & Groppe, S. (2021). A Platform for Interactive Data Science with Apache Spark for On-premises Infrastructure. *International Conference on Cloud Computing and Services Science*. https://api.semanticscholar.org/CorpusID:235234356

Misevičienmisevi˘misevičien, R., Tuminauskas, R., & Pažereckas, N. (2012). Educational Infrastructure Using Virtualization Technologies: Experience at Kaunas University of Technology. In *Informatics in Education* (Vol. 11, Issue 2).

Mkandla, R., & Chikohora, E. (2021). An Evaluation of Data Consistency Models in Geo-Replicated Cloud Storage. *2021 3rd International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*, 1–5. https://doi.org/10.1109/IMITEC52926.2021.9714674

Moosavi, S., Samavatian, M. H., Parthasarathy, S., & Ramnath, R. (2019). *A Countrywide Traffic Accident Dataset*. http://arxiv.org/abs/1906.05409

Moosavi, S., Samavatian, M. H., Parthasarathy, S., Teodorescu, R., & Ramnath, R. (2019). Accident risk prediction based on heterogeneous sparse data: New dataset and insights. *GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, 33–42. https://doi.org/10.1145/3347146.3359078

Nagar, A. (2017). Developing Big Data Curriculum with Open Source Infrastructure (Abstract Only). *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 700–701. https://doi.org/10.1145/3017680.3022386

Neciu, L.-F., Pop, F., Apostol, E.-S., & Truică, C.-O. (2021). Efficient Real-time Earliest Deadline First based scheduling for Apache Spark. *2021 20th International Symposium on Parallel and Distributed Computing (ISPDC)*, 97–104. https://doi.org/10.1109/ISPDC52870.2021.9521640

Neves, P. C., & Bernardino, J. (2015). Big Data in the Cloud: A Survey. In *Big Data in the Cloud: A Survey* (Vol. 1). www.ronpub.com/ojbd

Nguyen, T. T., Yeom, Y. J., Kim, T., Park, D. H., & Kim, S. (2020). Horizontal pod autoscaling in kubernetes for elastic container orchestration. *Sensors (Switzerland)*, *20*(16), 1–18. https://doi.org/10.3390/s20164621

Nr, M., & Rezzakul Haider, M. (2016). *Deployment of TOSCA Cloud Services Archives using Kubernetes*.

Pop, D., Neagul, M., & Petcu, D. (2014). On Cloud deployment of digital preservation environments. *IEEE/ACM Joint Conference on Digital Libraries*, 443–444. https://doi.org/10.5555/2740769.2740859

Qureshi, N. M. F., Siddiqui, I. F., Abbas, A., Bashir, A. K., Choi, K., Kim, J., & Shin, D. R. (2019a). Dynamic Container-based Resource Management Framework of Spark Ecosystem. *2019 21st International Conference on Advanced Communication Technology (ICACT)*, 522–526. https://doi.org/10.23919/ICACT.2019.8701970

Qureshi, N. M. F., Siddiqui, I. F., Abbas, A., Bashir, A. K., Choi, K., Kim, J., & Shin, D. R. (2019b). Dynamic Container-based Resource Management Framework of Spark Ecosystem. *2019 21st International Conference on Advanced Communication Technology (ICACT)*, 522–526. https://doi.org/10.23919/ICACT.2019.8701970

Salehi, M., Hughes, D., & Crispo, B. (2019). MicroGuard: Securing Bare-Metal Microcontrollers against Code-Reuse Attacks. *2019 IEEE Conference on Dependable and Secure Computing (DSC)*, 1–8. https://doi.org/10.1109/DSC47296.2019.8937667

Salkenov, A., & Bagchi, S. (2019). Cloud based autonomous monitoring and administration of heterogeneous distributed systems using mobile agents. *Future Generation Computer Systems*, *99*, 527–557. https://doi.org/https://doi.org/10.1016/j.future.2019.04.047

Salloum, S., Dautov, R., Chen, X., Peng, P. X., & Huang, J. Z. (2016). Big data analytics on Apache Spark. In *International Journal of Data Science and Analytics* (Vol. 1, Issues 3–4, pp. 145–164). Springer Science and Business Media Deutschland GmbH. https://doi.org/10.1007/s41060-016-0027-9

Seznec, A., ACM Digital Library., & Sigarch. (2010). *Proceedings of the 37th annual international symposium on Computer architecture.* ACM.

Shamim, M. S. I., Bhuiyan, F. A., & Rahman, A. (2020). XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices. *2020 IEEE Secure Development (SecDev)*, 58–64. https://doi.org/10.1109/SecDev45635.2020.00025

Suneetha, V., Suresh, S., & Jhananie, V. (2020). A Novel Framework using Apache Spark for Privacy Preservation of Healthcare Big Data. *2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, 743–749. https://doi.org/10.1109/ICIMIA48430.2020.9074867

Thiruvathukal, G. K., Christensen, C., Jin, X., Tessier, F., & Vishwanath, V. (2019). *A Benchmarking Study to Evaluate Apache Spark on Large-Scale Supercomputers*. http://arxiv.org/abs/1904.11812

Toka, L., Dobreff, G., Fodor, B., & Sonkoly, B. (2021). Machine Learning-Based Scaling Management for Kubernetes Edge Clusters. *IEEE Transactions on Network and Service Management*, *18*(1), 958–972. https://doi.org/10.1109/TNSM.2021.3052837

Ugwuanyi, S., Asif, R., & Irvine, J. (2020). Network Virtualization: Proof of Concept for Remote Management of Multi-Tenant Infrastructure. *2020 IEEE 6th International Conference on Dependability in Sensor, Cloud and Big Data Systems and Application (DependSys)*, 98–105. https://doi.org/10.1109/DependSys51298.2020.00023

USENIX Association. (2003). *Proceedings of the seventeenth Large Installation Systems Administration Conference (LISA XVII) : October 26-31, 2003 San Diego, CA, USA*. USENIX Association.

Verbraeken, J., Wolting, M., Katzy, J., Kloppenburg, J., Verbelen, T., & Rellermeyer, J. S. (2020). A Survey on Distributed Machine Learning. In *ACM Computing Surveys* (Vol. 53, Issue 2). Association for Computing Machinery. https://doi.org/10.1145/3377454

Wang, Y., Coiera, E., Runciman, W., & Magrabi, F. (2017). Using multiclass classification to automate the identification of patient safety incident reports by type and severity. *BMC Medical Informatics and Decision Making*, *17*(1). https://doi.org/10.1186/s12911-017-0483-8

Watts, T., Benton, R., Bourrie, D., & Shropshire, J. (2021). *Insight from a Containerized Kubernetes Workload Introspection*. University of Hawai'i at Manoa.

Xiang, Q., Tony Wang, X., Jensen Zhang, J., Newman, H., Richard Yang, Y., & Jace Liu, Y. (2019). Unicorn: Unified resource orchestration for multi-domain, geo-distributed data analytics. *Future Generation Computer Systems*, *93*, 188–197. https://doi.org/https://doi.org/10.1016/j.future.2018.09.048

Zaharia, M. (2019). Lessons from Large-Scale Software as a Service at Databricks. *Proceedings of the ACM Symposium on Cloud Computing*, 101. https://doi.org/10.1145/3357223.3365870

Zhang, X., Zheng, X., Wang, Z., Yang, H., Shen, Y., & Long, X. (2020). High-density Multi-tenant Bare-metal Cloud. *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 483–495. https://doi.org/10.1145/3373376.3378507

Zhu, C., Han, B., & Zhao, Y. (2020a). A Comparative Study of Spark on the bare metal and Kubernetes. *Proceedings - 2020 6th International Conference on Big Data and Information Analytics, BigDIA 2020*, 117–124. https://doi.org/10.1109/BigDIA51454.2020.00027

Zhu, C., Han, B., & Zhao, Y. (2020b). A Comparative Study of Spark on the bare metal and Kubernetes. *2020 6th International Conference on Big Data and Information Analytics (BigDIA)*, 117–124. https://doi.org/10.1109/BigDIA51454.2020.00027