ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ,
ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
Msc Big Data and Analytics

‘Διατύπωση μετρικών αξιοπιστίας και βέλτιστου τρόπου εκτέλεσης κατανεμημένων αλγορίθμων μηχανικής μάθησης’

Παπαευθυμίου Γεώργιος

Επιβλέπων Καθηγητής:
Δημοσθένης Κυριαζής

Σεπτέμβριος 2019

Ευχαριστίες

Στον φίλο και συνάδελφο Ανδριτσάκο Παναγιώτη
Βασιλική Χήρα
Νικόλαο Μόρφη
Νικόλαο Πολύζο
Αλέξανδρο Παπαευθυμίου
Την οικογένειά μου
Και όσους με στήριξαν για να ολοκληρώσω τις σπουδές μου

Περιεχόμενα

ABSTRACT

Big Data is usually defined by three characteristics called the three Vs (Volume, Velocity and Variety). It refers to Data that is very large, dynamic and complex. In this context, the data is difficult to be recorded, saved, processed and analyzed using traditional data processing applications. Consequently, the new conditions imposed upon us by Big Data present serious challenges on a different level, including data clustering. In general, Big Data assembling techniques can be classified into two categories; single machine clustering techniques and multiple machine clustering techniques. This particular thesis aims to examine the behavior of classification algorithms, which are applied on the analysis of large volumes of data in different technologies – cloud architectures. Cloud computing is a powerful technology for the execution of multiple and complex calculations. It eliminates the need to sustain expensive IT material, exclusive space and software. The massive increase of the data scale or the big data that is produced through cloud computing is a time consuming task that demands a large computational infrastructure for successful data processing and analysis. The purpose of classification algorithms is the understanding and extraction of values from large arrays of structured or unstructured data. In large volumes of unstructured data, it only makes sense to try and separate the data into logical clusters, before analyzing. In that way, classification allow us to have a mass perspective and to form some logical structures before continuing to analyze. Next, we will be presenting the most popular algorithms, which are broadly used to solve classification problems. In order to ensure that final conclusions are safe and objective, we will use a common dataset to run each algorithm separately. The Cloud needs to be distributed and programmed in such a way that providers achieve their goals and users meet the requirements of their applications minimum costs. The name we call this as a cloud resource allocation problem. Resource allocation isis traditionally viewed as an optimization problem, so resource allocation is NP-Hard. limited resources are available and allocated own resources in competitive circumstances / activities so that both parties will achieve their goals. This extensive review aims to process and analyze the numerous resolve / spam issues in cloud resource allocation.

ΚΕΦΑΛΑΙΟ 1: Εισαγωγή

1.1.1 Περιγραφή προβλήματος

Τα τελευταία χρόνια παρουσιάζεται μία εκθετική αύξηση στον όγκο των δεδομένων. Τόσο η αποθήκευση όσο και η ανάλυση των τεράστιων αυτών όγκων δεδομένων αποτελεί μεγάλη πρόκληση για τον κλάδο της επιστήμης των υπολογιστών. Οι παράγοντες που αυξάνουν την πολυπλοκότητα του προβλήματος είναι:

- Ο τεράστιος όγκος δεδομένων
- Η τεράστια ταχύτητα αναπαραγωγής τους
- Η ποικιλία των δεδομένων (αδόμητα, ημιδομημένα, δομημένα)
- Η ποικιλία διαφορετικών πηγών δεδομένων (web, social media, IoT, e-government)
- Η ανάγκη για ανάλυση δεδομένων σε πραγματικό χρόνο. Αυτό απαιτεί την ελαχιστοποίηση του χρόνου μεταξύ της καταγραφής των δεδομένων και της εξαγωγής γνώσης από την ανάλυσή τους.
- Οι αλγόριθμοι ανάλυσης τους
- Η υπολογιστική ισχύ που χρειάζονται
- Η υποδομή που θα φιλοξενεί

Την τελευταία δεκαετία έχουν γίνει πολύ μεγάλα βήματα προς την κατεύθυνση της βελτίωσης των εργαλείων και των λογισμικών τα οποία παρέχουν λύσεις στα ανωτέρω προβλήματα. Οι σημαντικές τεχνολογικές εξελίξεις και η πρόοδος, κυρίως στον τομέα του distributed computing, ευνόησαν την δημιουργία κατανεμημένων συστημάτων για την αποθήκευση και επεξεργασία των δεδομένων. Ένα από αυτά, και ίσως το πιο ευρέως χρησιμοποιούμενο, είναι το Hadoop.

Το περιβάλλον του Hadoop αποτελείται από δεκάδες εργαλεία και λογισμικά. Είναι πολύ δύσκολο για ένα αρχάριο χρήστη να κατανοήσει γρήγορα τι από όλα αυτά χρειάζεται για να πραγματοποιήσει την εργασία του. Ακόμα, η διασύνδεση μεταξύ των λογισμικών, όπως για παράδειγμα η μεταφορά δεδομένων από το σύστημα αποθήκευσης αρχείων του Hadoop προς μία βάση δεδομένων, απαιτεί πολύ εξειδικευμένες εντολές που είναι δύσκολο να συνταχθούν από έναν μη εξοικειωμένο χρήστη. Επίσης, η υλοποίηση ενός Hadoop cluster είναι μία πολύ απαιτητική εργασία που δεν μπορεί να υλοποιηθεί εύκολα χωρίς εξειδικευμένες γνώσεις.

Μία από τις πιο ενδεδειγμένες τεχνικές ανάλυσης δεδομένων, η μηχανική μάθηση, έχει γνωρίσει μεγάλη εξέλιξη τα τελευταία χρόνια, με εκατοντάδες διαθέσιμους αλγορίθμους για εφαρμογές σε σχεδόν όλους τους κλάδους των επιστημών. Παρόλα αυτά, η αξιοποίηση της επεξεργαστικής ισχύς των κατανεμημένων συστημάτων, σε εφαρμογές μηχανικής μάθησης, παραμένει σε σχετικά πρώιμο στάδιο. Αυτό έχει ως αποτέλεσμα να μην είναι ευρέως διαδεδομένος ο τρόπος χρήσης του Apache Spark. Οι τεχνολογικές εξελίξεις και η τεράστια αύξηση των διαθέσιμων δεδομένων, δημιουργούν την ανάγκη αξιοποίησης των συναφών frameworks όπως το Apache Spark, με τις πήγες όμως εκμάθησης να μην είναι πολλές.

1.1.2 Αντικείμενο μελέτης

Σκοπός της παρούσας μελέτης είναι η συγκριτική ανάλυση των δυνατοτήτων των πιο σύγχρονων εργαλείων Big Data με επίκεντρο το Spark και στη χρήση τους για έρευνα στο επιστημονικό πεδίο της μηχανικής μάθησης, χρησιμοποιώντας διαφορετικές αρχιτεκτονικές δόμησης των συστημάτων. Οι αρχιτεκτονικές που χρησιμοποιήθηκαν αποτυπώνονται με τη μορφή 3ων σεναρίων. Σε κάθε σενάριο-αρχιτεκτονική γίνεται χρήση τριών διαφορετικών αλγορίθμων με σκοπό να συγκεντρωθούν διαφορετικές μετρήσεις με βάση τις απαιτήσεις του κάθε αλγορίθμου. Η ανάλυση των δυνατοτήτων των εργαλείων καθώς και η χρήση του κατάλληλου κώδικα γίνεται με αναλυτικό και επεξηγηματικό τρόπο, ώστε να βοηθηθεί ο αναγνώστης ο οποίος επιθυμεί μέσα σε μικρό χρονικό διάστημα να διδαχθεί τον τρόπο εκτέλεσης των διεργασιών, τις δυνατότητες και κάποια παραδείγματα των τεχνολογιών αυτών. Η μελέτη περιλαμβάνει μία συνοπτική επισκόπηση του θεωρητικού υποβάθρου των Big Data, τριών αλγορίθμων ανάλυσης δεδομένων για supervised learning, των κατανεμημένων συστημάτων των Hadoop και Spark, και τέλος την απόκριση του συστήματος Spark βάση των μετρήσεων απόδοσης.

Πιο συγκεκριμένα, οι επιμέρους στόχοι της διπλωματικής είναι:

1) Η κατανόηση των εννοιών των Big Data και των τεχνικών ανάλυσής τους.
2) Η μελέτη των δυνατοτήτων των κατανεμημένων συστημάτων Hadoop και Spark.
3) Η παρουσίαση διαφορετικών αρχιτεκτονικών αυτών των συστημάτων.
4) Η παρουσίαση του virtualization ως κομμάτι της βελτιωμένης λειτουργίας των Big Data συστημάτων.
5) Η χρήση Cloud τεχνολογιών για την φιλοξενία των Hadoop Cluster.
6) Η λειτουργία διαφορετικών Hadoop clusters.
7) Η εφαρμογή διερευνητικής ανάλυσης δεδομένων (exploratory data analysis) με χρήση της γλώσσας προγραμματισμού Python.
8) Η εφαρμογή τριών αλγορίθμων μηχανικής μάθησης χρησιμοποιώντας την κατανεμημένη επεξεργαστική ισχύ ενός Hadoop cluster.
9) Η παρουσίαση του τρόπου λειτουργίας του Apache Spark.
10) Η καταγραφή των πόρων που χρειαζεται το Spark βάση των διαφορετικών αρχιτεκτονικών.
11) Η ανάλυση των μετρικών Dependability (Fault Tolerance), Isolation
12) Η ανάλυση των αποτελεσμάτων των αλγορίθμων με βάση τις μετρικές Accuracy, Time.

1.1.3. Περιεχόμενα της μελέτης

Στο κεφάλαιο 2 γίνεται η ανάλυση της μεθοδολογίας που ακολουθήθηκε για την υλοποίηση της εργασίας. Περιγράφονται επίσης τα λογισμικά και τα εργαλεία που χρησιμοποιήθηκαν.

Στο κεφάλαιο 3 γίνεται η ανάλυση των Big Data για την κατανόηση των εννοιών των 5V’s, της δομής τους, των διαφορετικών τύπων τους και από ποιες πηγές συλλέγονται. Επίσης, γίνεται μια συγκεντρωτική επισκόπηση των διαφορετικών τεχνικών ανάλυσή τους και δίνεται ιδιαίτερη βαρύτητα στην μηχανική μάθηση.

ΚΕΦΑΛΑΙΟ 2: Μεθοδολογία

2.1 Εισαγωγή

Η παρούσα διπλωματική εργασία αποτελεί έναν σύνολο διαφορετικών αρχιτεκτονικών, πρακτικών εφαρμογών και χρήσης διαφορετικών λογισμικών και τεχνικών για την διαχείριση και αξιοποίηση μεγάλων συνόλων δεδομένων. Έγινε προσπάθεια ώστε τα λογισμικά και εργαλεία που χρησιμοποιήθηκαν να είναι όσο το πιο δυνατόν πιο σύγχρονα με τις πιο πρόσφατες εξελίξεις στον χώρο των Big Data, καθώς και λύσεις που χρησιμοποιούν οι μεγαλύτερες εταιρείες και οργανισμοί παγκοσμίως. Για το λόγο αυτό, επικεντρωθήκαμε στην διαδικτυακή αναζήτηση για την εύρεση πηγών οι οποίες ανταποκρίνονται σε αυτή την απαίτηση. Μέσω των πηγών αυτών, ενημερωθήκαμε από εξειδικευμένους επαγγελματίες του χώρου τόσο για τις τελευταίες εξελίξεις όσο και για τον τρόπο αξιοποίησης των εργαλείων. Πολύ σημαντική συνεισφορά υπήρξε από διάφορες πλατφόρμες διαδικτυακής εκπαίδευσης όπως το Udemy και το Kaggle, από τις οποίες αντλήθηκαν πολύτιμες γνώσεις για την χρήση των εργαλείων και για εύρεση σετ δεδομένων. Επίσης, πολύ σημαντική βοήθεια παρέχουν τα έγγραφα χρήσης (documentations) των κατάλληλων εργαλείων που χρησιμοποιήθηκαν (γλώσσες προγραμματισμού, βιβλιοθήκες, γλώσσες ανάκτησης, εγχειρίδιο χρήσης Hadoop, εφαρμογή μηχανικής μάθησης). Τέλος, σε πολλές περιπτώσεις που προέκυψαν δυσκολίες για την χρήση των εργαλείων ή για την γραφή του κατάλληλου κώδικα, η σελίδα stackoverflow.com παρείχε σημαντική βοήθεια. Η προσέγγιση που ακολουθήθηκε είναι:

1) Αναζήτηση πηγών για τις τελευταίες εξελίξεις στον χώρο των Big Data.
2) Αναζήτηση του κατάλληλου σετ δεδομένων το οποίο καλύπτει το ερευνητικό ενδιαφέρον μας.
3) Εκμάθηση του θεωρητικού υπόβαθρου των τεχνολογιών και των εργαλείων (π.χ.. μηχανική μάθηση).
4) Εκμάθηση εργαλείων μέσω διαδικτυακών πλατφορμών εκπαίδευσης και εγχειριδίων χρήσης.
5) Εφαρμογή των θεωρητικών και πρακτικών γνώσεων στο σετ δεδομένων.
6) Αξιολόγηση των αποτελεσμάτων και πιθανές διορθώσεις στον κώδικα.
7) Εξαγωγή συμπερασμάτων.

2.2.1 Hadoop cluster

Η διαδικασία υλοποίησης ενός Hadoop cluster είναι μία πολύ απαιτητική διαδικασία. Για τον λόγο αυτό προτιμήθηκε η χρήση ενός έτοιμου προ-ρυθμισμένου Hadoop cluster της Hortonworks. Η Hortonworks είναι μια εταιρεία λογισμικού που εδρεύει στην Santa Clara της Καλιφόρνια, η οποία αναπτύσσει, υποστηρίζει και παρέχει τεχνογνωσία σε ένα σύνολο λογισμικών ανοικτού κώδικα που έχουν σχεδιαστεί για τη διαχείριση και επεξεργασία δεδομένων. Στα πλαίσια αυτά, παρέχει δωρεάν για εκμάθηση το Sandbox το οποίο είναι ένα εικονικό περιβάλλον ενός Hadoop cluster το οποίο περιλαμβάνει τα περισσότερα από τα εργαλεία του Apache Hadoop. Το Sandbox μπορεί να τρέξει στο cloud ή στον προσωπικό υπολογιστή χρησιμοποιώντας ένα virtual machine monitor (VMM) [18]

Το VMM που προτιμήθηκε, είναι το VirtualBox της Oracle . Η διαδικασία είναι σχετικά απλή. Στο αρχικό μενού του VirtualBox, επιλέγοντας την επιλογή “new” εμφανίζεται ο οδηγός εγκατάστασης του Sandbox

Το Sandbox απαιτεί αρκετό αποθηκευτικό χώρο, πάνω από 30Gb, [12] καθώς και τουλάχιστον 8Gb RAM. Το Sandbox τρέχει σε Linux, πιο συγκεκριμένα στην έκδοση Ubuntu. Επίσης, για την

«απομακρυσμένη» επικοινωνία με το cluster χρειάζεται ένα πρόγραμμα SSH client. Στην παρούσα μελέτη προτιμήθηκε το Putty.

2.2.1 Python

Για την διερευνητική ανάλυση δεδομένων και για την εφαρμογή του αλγορίθμου μηχανικής μάθησης χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python. Η Python είναι μια γλώσσα προγραμματισμού υψηλού επιπέδου για γενικό προγραμματισμό. Δημιουργήθηκε από τον Guido van Rossum και κυκλοφόρησε για πρώτη φορά το 1991. Η Python έχει μια φιλοσοφία σχεδίασης που δίνει έμφαση στην αναγνωσιμότητα του κώδικα, χρησιμοποιώντας υποχρεωτικά indentation

Όταν πρόκειται για εφαρμογές στον κλάδο του data science, η Python είναι ένα πολύ ισχυρό εργαλείο. Πολύ βασικό χαρακτηριστικό της το οποίο την κάνει πολύ δημοφιλή είναι ότι πρόκειται για γλώσσα ανοιχτού κώδικα. Διαθέτει εξαιρετικές βιβλιοθήκες για την επεξεργασία δεδομένων και είναι σχετικά εύκολη στην εκμάθησή της

Οι πιο δημοφιλείς βιβλιοθήκες της για ανάλυση δεδομένων είναι οι :
- NumPy, για επιστημονικούς υπολογισμούς και κυρίως για τις πράξεις μεταξύ πινάκων
- Pandas, για επεξεργασία δεδομένων με την δημιουργία πινάκων (Dataframes)
- Matplotlib, για δημιουργία γραφημάτων
- Scikit-learn, για μηχανική μάθηση.

Τέλος, για εργασίες ανάλυσης δεδομένων συνίσταται η χρήση ως IDE (Integrated Development Environment) του IPython Notebook (γνωστό και ως Jupyter Notebook).

2.2.2 Virtualization

Με τον όρο virtualization εννοούμε την τεχνολογία με την οποία τα φυσικά συστήματα μετατρέπονται σε ιδεατά (virtual machines). Κάθε φυσικός πόρος (επεξεργαστική ισχύς, μνήμη, δίκτυο, storage κλπ.) γίνεται ένας ενιαίος πόρος και μοιράζεται ταυτόχρονα σε πολλά εικονικά συστήματα.

Για μια λύση virtualization χρειάζονται τα εξής μέρη:
- Κατάλληλος εξοπλισμός hardware
- Λογισμικό virtualization (hypervisor)
- Λογισμικό διαχείρισης

Με το virtualization το hardware διαχωρίζεται από το software (λειτουργικά συστήματα και εφαρμογές). Ο hypervisor είναι ένα νέο επίπεδο «virtualization layer» μεταξύ του hardware και του software, που ενοποιεί τους φυσικούς πόρους και τους διαμοιράζει στα ιδεατά συστήματα με τρόπο διάφανο. Τα ιδεατά συστήματα συνεχίζουν να νομίζουν ότι επικοινωνούν απευθείας με το hardware, αλλά στην πραγματικότητα επικοινωνούν με το virtualization layer. Αυτό επιτρέπει τη μετακίνηση επεξεργαστικής ισχύος, μνήμης ή και storage από ένα virtual machine σε άλλο εν ώρα λειτουργίας και

σύμφωνα με τις ανάγκες. Το αποτέλεσμα είναι η καλύτερη εκμετάλλευση των πόρων συνολικά, μεγάλη εξοικονόμηση ενέργειας και φυσικού χώρου και καλύτερη διαχείριση της υποδομής. [6]

Σημαντικό όφελος είναι επίσης η υψηλή διαθεσιμότητα. Οι λύσεις virtualization περιλαμβάνουν τεχνικές όπως clustering και multi-pathing για την αντιμετώπιση προβλημάτων σε ένα φυσικό server ή κάποια σύνδεση, χωρίς διακοπή λειτουργίας. Έτσι όλα τα ιδεατά συστήματα επωφελούνται από τις τεχνικές αυτές χωρίς να χρειάζεται κάτι πρόσθετο σε αυτά. Επιπλέον, είναι δυνατή η δημιουργία snapshots των ιδεατών μηχανών ενώ αυτές λειτουργούν και η επαναφορά τους σε οποιαδήποτε παλαιότερη στιγμή με ασφάλεια.

Εκεί όμως που υπάρχει μεγάλη διαφορά είναι η ευκολότερη διαχείριση όλου του περιβάλλοντος. Μπορούμε σε ελάχιστο χρόνο να δημιουργήσουμε ένα νέο σύστημα, να αυξήσουμε τη CPU, τη μνήμη και το χώρο σε δίσκο που χρησιμοποιεί ένα ιδεατό σύστημα, εργασίες που με φυσικά συστήματα απαιτούσαν περίπλοκους και χρονοβόρους χειρισμούς.

Τέλος, με το virtualization η δημιουργία λύσεων disaster recovery γίνεται πολύ πιο απλή, αξιόπιστη και οικονομική, καθώς στη εφεδρική τοποθεσία μεταφέρεται αυτούσιο όλο το σύστημα (λειτουργικό σύστημα και εφαρμογή) χωρίς προβλήματα consistency.

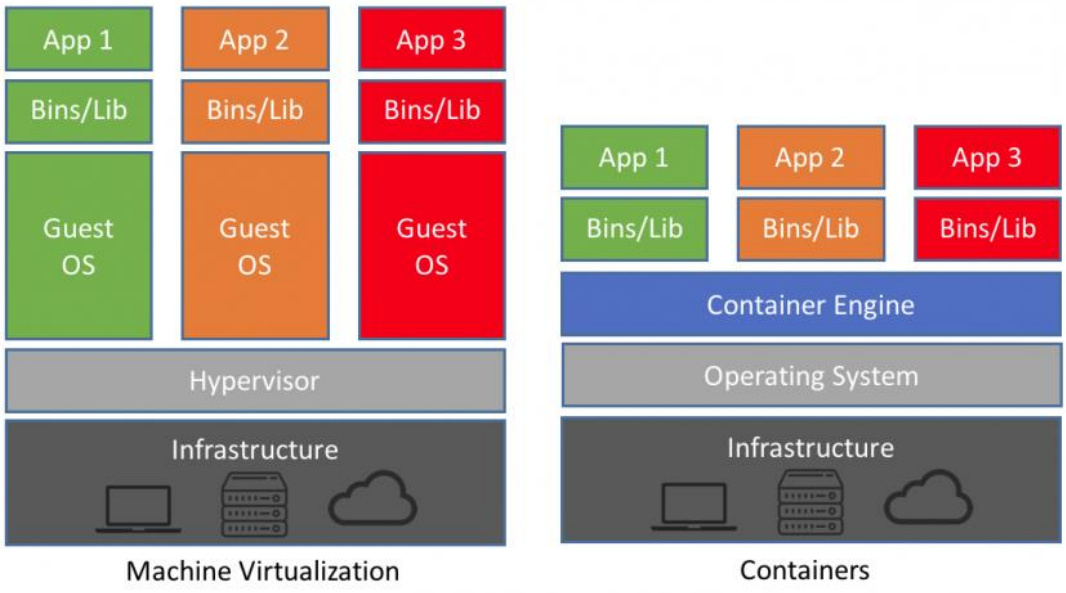
Σήμερα το virtualization έχει διεισδύσει σε πολλές επιχειρήσεις, αλλά αναμένεται ότι γρήγορα τα περισσότερα συστήματα θα γίνουν ιδεατά, καθώς οι απαιτήσεις για αποδοτική χρησιμοποίηση του hardware, μείωση της ηλεκτρικής κατανάλωσης και αναγκών ψύξης, γίνονται όλο και πιο επιτακτικές.

Υπάρχει ακόμα η επιφύλαξη ότι η απόδοση μειώνεται στα ιδεατά συστήματα σε σχέση με τα φυσικά, με τη σκέψη ότι εισάγεται ένα πρόσθετο επίπεδο μεταξύ των εφαρμογών και του hardware. Ωστόσο, πρόκειται πια για ώριμη τεχνολογία που έχει λύσει προβλήματα απόδοσης. Οι μετρήσεις επιβεβαιώνουν ότι η επίπτωση είναι πολύ μικρή, κυρίως λόγω της καλύτερης χρήσης του hardware από το hypervisor.

2.2.3 Virtual Machines

Ιστορικά, καθώς η ισχύς και η χωρητικότητα επεξεργασίας του server αυξήθηκαν, οι εφαρμογές bare metal δεν ήταν σε θέση να εκμεταλλευτούν τη νέα αφθονία πόρων. Έτσι, δημιουργήθηκαν τα VMs, σχεδιασμένα με την εκτέλεση λογισμικού πάνω από φυσικούς servers για να μιμηθούν ένα συγκεκριμένο σύστημα υλικού. Ένας hypervisor ή μια οθόνη εικονικής μηχανής είναι λογισμικό, υλικολογισμικό ή υλικό που δημιουργεί και εκτελεί το VM. Είναι αυτό που τοποθετείται μεταξύ του υλικού και της εικονικής μηχανής και είναι απαραίτητο για το virtualization του server.

Μέσα σε κάθε εικονική μηχανή λειτουργεί ένα μοναδικό guest OS. Τα VMs με διαφορετικά λειτουργικά συστήματα μπορούν να τρέξουν στον ίδιο φυσικό server - ένα UNIX VM μπορεί να καθίσει δίπλα σε Linux VM κ.ο.κ. Κάθε VM έχει τα δικά του δυαδικά αρχεία, τις βιβλιοθήκες και τις εφαρμογές που το εξυπηρετεί. [13]



Εικόνα 1 Διαφορές Αρχιτεκτονικής VMs και Containers

Το virtualization διακομιστών παρείχε μια ποικιλία πλεονεκτημάτων, μία από τις μεγαλύτερες είναι η δυνατότητα ενοποίησης εφαρμογών σε ένα ενιαίο σύστημα. Πηγαίνουν οι ημέρες μιας ενιαίας εφαρμογής που εκτελείται σε ένα μόνο server. Το virtualization οδήγησε σε εξοικονόμηση κόστους μέσω μειωμένου αποτυπώματος, ταχύτερης παροχής υπηρεσιών server και βελτιωμένης ανάκτησης καταστροφών (DR), επειδή το υλικό του χώρου DR δεν χρειάστηκε πλέον να αντικατοπτρίζει το πρωτεύον κέντρο δεδομένων.

Η ανάπτυξη επωφελήθηκε επίσης από αυτή τη φυσική εξυγίανση επειδή η μεγαλύτερη αξιοποίηση σε μεγαλύτερους και ταχύτερους servers απελευθέρωσε αργότερα τους αχρησιμοποίητους εξυπηρετητές για να επαναπροωθηθεί για QA, ανάπτυξη ή εργαστηριακά εργαλεία.

Αλλά αυτή η προσέγγιση είχε τα μειονεκτήματά της. Κάθε VM περιλαμβάνει ξεχωριστή εικόνα λειτουργικού συστήματος, η οποία προσθέτει επιβάρυνση στη μνήμη και το αποτύπωμα αποθήκευσης. Όπως αποδεικνύεται, το ζήτημα αυτό προσθέτει πολυπλοκότητα σε όλα τα στάδια ενός κύκλου ζωής ανάπτυξης λογισμικού - από την ανάπτυξη και δοκιμή μέχρι την παραγωγή και την ανάκαμψη μετά από καταστροφή. Η προσέγγιση αυτή περιορίζει επίσης σημαντικά τη δυνατότητα μεταφοράς εφαρμογών μεταξύ δημόσιων σύννεφων, ιδιωτικών σύννεφων και παραδοσιακών κέντρων δεδομένων.

2.2.4 Containers

Το virtualization του λειτουργικού συστήματος (OS) έχει εξελιχθεί σε δημοτικότητα κατά την τελευταία δεκαετία για να επιτρέψει το λογισμικό να λειτουργεί με ασφάλεια και πρόβλεψη όταν μετακινείται από ένα περιβάλλον server σε άλλο. Ωστόσο, τα containers παρέχουν έναν τρόπο για να τρέξουν αυτά τα απομονωμένα συστήματα σε ένα μόνο server ή κεντρικό λειτουργικό σύστημα.

Τα containers τοποθετούνται πάνω από ένα φυσικό server και το κεντρικό λειτουργικό του σύστημα - για παράδειγμα, το Linux ή τα Windows. Κάθε κονσόλα μοιράζεται τον πυρήνα λειτουργικού συστήματος του [3] κεντρικού υπολογιστή και, συνήθως, τα δυαδικά αρχεία και τις βιβλιοθήκες. Τα κοινόχρηστα στοιχεία είναι μόνο για ανάγνωση. Τα containers είναι επομένως εξαιρετικά "ελαφριά" - είναι μεγέθους τύπου megabytes και χρειάζονται μόλις δευτερόλεπτα για να ξεκινήσουν, έναντι των gigabytes και των λεπτών για ένα VM.

Τα κοντέινερ μειώνουν επίσης τα γενικά έξοδα διαχείρισης. Επειδή μοιράζονται ένα κοινό λειτουργικό σύστημα, μόνο ένα λειτουργικό σύστημα χρειάζεται φροντίδα και τροφοδοσία για διορθώσεις σφαλμάτων, διορθώσεις και ούτω καθεξής. Αυτή η ιδέα είναι παρόμοια με αυτή που βιώνουμε με τους κεντρικούς υπολογιστές hypervisor: λιγότερα σημεία διαχείρισης, αλλά ελαφρώς υψηλότερο τομέα σφάλματος. Εν ολίγοις, τα containers είναι ελαφρύτερα και πιο φορητά από τα VM. [13]

2.2.5 Συμπέρασμα

Οι εικονικές μηχανές και τα containers διαφέρουν με πολλούς τρόπους, αλλά η κύρια διαφορά είναι ότι τα containers παρέχουν έναν τρόπο για το virtualization ενός λειτουργικού συστήματος, έτσι ώστε να μπορούν να εκτελούνται πολλαπλά φορτία σε ένα μοναδικό στιγμιότυπο OS. Με τα VMs, το υλικό εικονικοποιείται για την [3] εκτέλεση πολλαπλών παρουσιών OS. Η ταχύτητα, η ευελιξία και η φορητότητα των containers τα καθιστούν ένα ακόμη εργαλείο που συμβάλλει στη βελτιστοποίηση της ανάπτυξης λογισμικού.

2.3 Cloud (Architecture, Computing)

Το Cloud computing είναι μια ισχυρή τεχνολογία για την εκτέλεση μεγάλου όγκου και πολύπλοκων υπολογισμών. Η παροχή κατά παραγγελία υπολογιστικής ισχύος, βάσης δεδομένων, αποθήκευσης, εφαρμογών και άλλων πόρων πληροφορικής μέσω διαδικτύου με τιμολογιακή πληρωμή. Είτε χρησιμοποιείτε για να [34]εκτέλεση εφαρμογων που μοιράζονται φωτογραφίες σε εκατομμύρια χρήστες κινητών τηλεφώνων είτε για υποστήριξη επιχειρησιακών κρίσιμων λειτουργιών, μια πλατφόρμα υπηρεσιών cloud παρέχει γρήγορη πρόσβαση σε ευέλικτους και χαμηλού κόστους πόρους πληροφορικής. Με το cloud computing, δεν χρειάζονται μεγάλες επενδύσεις εκ των προτέρων στο υλικό και να αφιερώνετε πολύ χρόνο στην βαριά άρση της διαχείρισης αυτού του υλικού. Η μαζική αύξηση της κλίμακας των δεδομένων ή μεγάλα δεδομένα που παράγει το cloud computing είναι μια χρονοβόρα εργασία που χρειάζεται μεγάλη υπολογιστική ισχύ για την επεξεργασία και ανάλυση αυτών.

2.3.1 Αρχιτεκτονική του Cloud

Η ιδέα της παροχής κεντρικών υπολογιστικών υπηρεσιών σε ένα δίκτυο δεν είναι καινούργια - η τεχνολογία χρονομεριστικής χρήσης του mainframe ήταν δημοφιλής ήδη από τη δεκαετία του 1960, αλλά αντικαταστάθηκε από προσωπικούς υπολογιστές και αρχιτεκτονική πελάτη-διακομιστή. Τα προηγούμενα χρόνια η τυπική υποδομή πληροφορικής για επιχειρήσεις αποτελούνταν από ακριβούς και μεγάλους σε υπολογιστική ισχύ servers. Η αρχιτεκτονική υποδομής ήταν μονολιθική και κάθε μία από αυτές τις ισχυρές μηχανές θα μπορούσε εύκολα να φιλοξενήσει έναν συγκεκριμένο αριθμό από επιχειρησιακές εφαρμογές. Σε αυτή την αγορά κυριαρχούν μόνο λίγοι προμηθευτές hardware, όπως η IBM, η HP και η Dec, των οποίων οι servers ήταν κοστοβόροι για αγορά και συντήρηση, χρειαζόντουσαν αρκετό χρόνο για εγκατάσταση και αναβάθμιση και σε ορισμένες περιπτώσεις ήταν ευάλωτες σε διακοπές διακομιστή οι οποίες διαρκούσαν αρκετές ώρες έως ότου ένας εκπρόσωπος του πωλητή παραδώσει αποκλειστικά ανταλλακτικά.

Το OS ήταν απευθείας εγκατεστημένο στο υλικό και οι περισσότεροι από τους servers φιλοξενούσαν πολλαπλές εφαρμογές μέσα στο ίδιο λειτουργικό σύστημα χωρίς να παρέχουν φυσικο ή εικονικο Isolation. Επειδή ήταν δύσκολο να μετακινηθούν γρήγορα και να εξισορροπηθούν οι εφαρμογές σε διάφορους servers, οι πόροι διακομιστών δεν χρησιμοποιήθηκαν αποτελεσματικότερα.

Οι κατανεμημένες εφαρμογές, οι οποίες εγκαταστάθηκαν σε πολλούς servers, επικοινωνούσαν μεταξύ τους χρησιμοποιώντας πρωτόκολλα επικοινωνίας CORBA ή DCOM μέσω RPC. Ωστόσο, ένα μεγάλο πρόβλημα με τέτοια πρωτόκολλα ήταν ότι εξαρτώνταν από τον προμηθευτή και έτσι η εφαρμογή ενός προμηθευτή ίσως δεν ήταν συμβατή με αυτή των άλλων. Αυτό επιλύθηκε στις αρχές της δεκαετίας του 2000 με την εισαγωγή υπηρεσιών διαδικτύου, οι οποίες χρησιμοποιούν ανοικτές προδιαγραφές που είναι γλώσσα, πλατφόρμα και πωλητής αγνωστικιστές. [34]

2.3.2 Ασφάλεια δεδομένων

Η ασφάλεια των δεδομένων είναι μακράν το πιο δύσκολο εμπόδιο στην υιοθέτηση του νέφους. Τα δεδομένα είναι το πιο πολύτιμο εταιρικό περιουσιακό στοιχείο και οι εταιρείες θέλουν να γνωρίζουν ότι τα δεδομένα τους είναι ασφαλή. Οι εταιρείες αισθάνονται σίγουροι όταν αποθηκεύουν δεδομένα εσωτερικά επειδή έχουν πλήρη έλεγχο αυτού. Παρόλο που δεν υπάρχει εγγύηση ότι τα δεδομένα προστατεύονται καλύτερα εσωτερικά συγκριτικά με το δημόσιο σύννεφο. Στην πραγματικότητα, υπάρχει η πιθανότητα ότι τα δεδομένα θα μπορούσαν να είναι ακόμη ασφαλέστερα στο δημόσιο σύννεφο, επειδή οι δημόσιοι προμηθευτές νέφους ενδέχεται να παρουσιάζουν υψηλότερο επίπεδο εμπειρογνωμοσύνης στον τομέα της ασφάλειας των δεδομένων σε σύγκριση με τους πελάτες τους.

Όταν αποθηκεύονται σε δημόσιο σύννεφο, τα δεδομένα μπορούν να διακυβεύονται σε πολλά διαφορετικά στάδια δεδομένων: κατά τη μεταφορά από το εσωτερικό δίκτυο εταιρειών στο δημόσιο σύννεφο, όταν τα δεδομένα αποθηκεύονται στο δημόσιο σύννεφο και κατά τη διάρκεια των διαδικασιών δημιουργίας αντιγράφων ασφαλείας και επαναφοράς δεδομένων. [6]

2.3.3 Απώλεια ελέγχου

Η απώλεια των παραγόντων ελέγχου μπορεί να υποδιαιρεθεί σε δύο τύπους: τεχνική απώλεια ελέγχου και οργανωτική απώλεια ελέγχου. Η τεχνική απώλεια ελέγχου περιλαμβάνει παράγοντες όπως τον έλεγχο πρόσβασης, τις εκδόσεις λογισμικού και τις ενημερώσεις και τον έλεγχο της χρονικής διάρκειας των τεχνικών λειτουργιών, όπως είναι η δημιουργία αντιγράφων ασφαλείας και η αποκατάσταση των δεδομένων, και σχετίζεται εν μέρει με τα ζητήματα ασφάλειας δεδομένων που αναφέρονται παραπάνω. Για να αντιμετωπιστούν οι τεχνικές απώλειες των προκλήσεων ελέγχου, ένας πάροχος σύννεφων πρέπει να προσφέρει εργαλεία της εταιρείας που παρέχουν πλήρη προβολή σε όλες τις λειτουργίες σύννεφων που σχετίζονται με τα περιουσιακά στοιχεία της εταιρείας.

Η οργανωτική απώλεια ελέγχου σχετίζεται κυρίως με τους ανθρώπινους παράγοντες που μπορούν να δημιουργήσουν εμπόδια για τη μετατροπή σε υπολογιστικά cloud. Αυτοί οι παράγοντες μπορεί να περιλαμβάνουν τους φόβους ορισμένων ατόμων που χάνουν επιρροή στην οργάνωση, ο φόβος της απώλειας θέσεων εργασίας εάν ο μετασχηματισμός σύννεφου επηρεάζει λειτουργικά συγκεκριμένες θέσεις και η απλή αδυναμία να αγκαλιάσει την αλλαγή. Ο φόβος της απώλειας ελέγχου είναι κοινός και ισχύει όχι μόνο για την αλλαγή του cloud computing αλλά και για πολλές άλλες οργανωτικές αλλαγές.

Ένα άλλο πρόβλημα είναι ότι ορισμένοι οργανισμοί δεν είναι απλώς έτοιμοι να αποδεχθούν αλλαγές ή κάνουν πολύ αργά την εφαρμογή αυτών των αλλαγών. Ως εκ τούτου, είναι σημαντικό να διασφαλιστεί η υποστήριξη της ανώτατης διοίκησης προκειμένου να εφαρμοστούν ή για να μετρηθούν, επειδή οι οργανισμοί είναι συνήθως απρόθυμοι να αποκαλύψουν αυτά τα είδη των προβλημάτων διαχείρισης. αλλαγές. για να μετρηθούν, επειδή οι οργανισμοί είναι συνήθως απρόθυμοι να αποκαλύψουν αυτά τα είδη των προβλημάτων διαχείρισης.

2.3.4 Συμφωνίες SLA

Μια συμφωνία SLA (Συμφωνία επιπέδου υπηρεσιών) είναι μια σύμβαση που περιγράφει το επίπεδο των υπηρεσιών που προσφέρει ένας πάροχος σύννεφο. Στην περίπτωση υπηρεσιών cloud, το SLA θα μπορούσε να μετρηθεί με βάση τον μέσο χρόνο μεταξύ αποτυχιών, μέσος χρόνος για την επιδιόρθωση της διακοπής και άλλες επιχειρησιακές μετρήσεις όπως ο χρόνος απόκρισης του δικτύου και η απόδοση του συστήματος. [34]

Οι εταιρείες πρέπει να επιδείξουν την απαιτούμενη επιμέλεια για να εξετάσουν προσεκτικά τις συμφωνίες SLA του παρόχου σύννεφο. Ο κάθε πάροχος σύννεφων δεν θέλει (ή μπορεί να) προσφέρει το επίπεδο επιχειρησιακής συνέχειας που απαιτείται από τους οργανισμούς. Ακόμη και οι παροχείς νέφους τόσο μεγάλες όσο το Amazon παρέχουν μόνο 99,95% εγγύηση ετήσια uptime για τους servers τους, ενώ ορισμένοι οργανισμοί απαιτούν 99,99% ετήσιο uptime. Εάν το uptime της υπηρεσίας πέσει κάτω από 99,95%, σύμφωνα με τη συμφωνία του Amazon οι πελάτες είναι επιλέξιμοι για πίστωση υπηρεσιών ίση με το 10% των λογαριασμών τους. Σημειώστε ότι το SLA της Amazon δεν περιορίζει τη διάρκεια του χρόνου αναμονής - ανεξάρτητα από το αν οι servers σας βρίσκονται εκτός λειτουργίας για δύο ώρες ή 10 ημέρες, η εταιρεία σας εξακολουθεί να λαμβάνει το ίδιο ποσό αποζημίωσης.

2.3.5 Φορητότητα / ενσωμάτωση δεδομένων

Μπορεί να είναι τεχνικά δύσκολο να ενσωματωθούν δεδομένα στο εσωτερικό κέντρο δεδομένων μιας εταιρείας με δεδομένα που βρίσκονται σε δημόσιο σύννεφο εκτός των εγκαταστάσεων. Οι οργανισμοί που εξετάζουν το ενδεχόμενο να χρησιμοποιούν ένα υβριδικό σύννεφο, όπου τα δεδομένα διαδίδονται σε ιδιωτικά και δημόσια σύννεφα, ενδέχεται να αντιμετωπίσουν προβλήματα ενοποίησης δεδομένων:

• Θέματα ασφάλειας (διαχείριση δεδομένων, σύνδεση δικτύου κ.λπ.)

• Προβλήματα με την ακεραιότητα συναλλαγών (αδυναμία υποστήριξης των συναλλαγών σε σύννεφα)

• Δυσκολίες χειρισμού μεγάλων όγκων δεδομένων

• Έλλειψη μηχανισμών για την ανίχνευση αλλαγών στα δεδομένα

• Ζητήματα ελέγχου ποιότητας δεδομένων

• Προβλήματα που καθορίζουν την προέλευση των δεδομένων

2.3.6 Συμβατότητα Λογισμικού

Οι πάροχοι σύννεφων συνήθως υποστηρίζουν ένα συγκεκριμένο σύνολο προμηθευτών και εκδόσεων λογισμικού. Ένα δημόσιο σύννεφο είναι ένα κοινό περιβάλλον, όπου το λογισμικό μοιράζεται μεταξύ εκατοντάδων ή χιλιάδων απομονωμένων πελατειακών περιβαλλόντων. Είναι κρίσιμο για τον παροχέα σύννεφων να διατηρεί καλά καθορισμένα πρότυπα λογισμικού και επομένως σε πολλές περιπτώσεις οι πάροχοι νέφους δεν μπορούν να προσφέρουν προσαρμοσμένα πακέτα λογισμικού εγκατεστημένα σε σύννεφα πελατών. Ιδιαίτερα για σύννεφα PaaS ή SaaS, το επίπεδο ελέγχου του λογισμικού είναι πολύ περιορισμένο. Οι εταιρείες πρέπει να διασφαλίζουν ότι το λογισμικό σε ένα δημόσιο σύννεφο είναι συμβατό με αυτό που χρησιμοποιούν εσωτερικά.

2.3.7 Απόδοση

Οι περισσότερες συμφωνίες SLA του παρόχου σύννεφων καλύπτουν μόνο τη διαθεσιμότητα της υποδομής και όχι την απόδοση. Εάν οι εταιρικές εφαρμογές έχουν συγκεκριμένες απαιτήσεις σχετικές με την απόδοση, η εταιρεία θα πρέπει να συζητήσει αυτές τις απαιτήσεις με τους παρόχους cloud και να επιβεβαιώσει ότι αυτές οι απαιτήσεις μπορούν να υποστηριχθούν. Είναι καλή ιδέα να συμπεριληφθούν αυτές οι απαιτήσεις σε σύμβαση SLA και αποτελεί συνήθη πρακτική η διαπραγμάτευση της επαφής SLA με τον πάροχο υπηρεσιών σύννεφο.

Είναι ευθύνη του πελάτη του cloud να παρακολουθεί την απόδοση του cloud και να διασφαλίζει ότι συμμορφώνεται με τις απαιτήσεις και τα SLA - οι μετρήσεις απόδοσης που συλλέγονται πρέπει να

αναλύονται συνεχώς. Εάν οι εφαρμογές που φιλοξενούνται σε σύννεφο χρησιμοποιούνται σε παγκόσμιο επίπεδο, είναι σημαντικό να παρακολουθούνται παράμετροι απόδοσης όπως η καθυστέρηση του δικτύου σε όλες τις μεγάλες τοποθεσίες πελατών.

2.3.8 Υπηρεσίες

Το Cloud computing ξεκίνησε τον μεγαλύτερο μετασχηματισμό της πληροφορικής στην ιστορία και αυτή η μετατροπή άνοιξε πολλές νέες επιχειρηματικές ευκαιρίες. Ήδη τα δημόσια σύννεφα παρέχουν περισσότερες δυνατότητες για τους παρόχους υπηρεσιών cloud.

Αυτή η γρήγορη υιοθέτηση του cloud μπορεί να εξηγηθεί από διάφορους παράγοντες:

• Δεν απαιτούνται προκαταβολικές επενδύσεις ή δεσμεύσεις.

• Πληρώνετε μόνο για όσα χρησιμοποιείτε.

• Οι εταιρείες μπορούν να δοκιμάσουν τις υπηρεσίες πριν από την αγορά.

• Απαιτούνται λιγότεροι ανθρώπινοι πόροι για τη συντήρηση της υποδομής.

• Οι προσφορές υπηρεσιών είναι εύκολο να συγκριθούν.

• Οι αναβαθμίσεις λογισμικού μπορούν να αυτοματοποιηθούν.

Η μετάβαση στο δημόσιο σύννεφο αναμένεται να επιταχυνθεί. Ενώ αναμένονται να αναπτυχθούν και άλλα μοντέλα ανάπτυξης - όπως η διαχειριζόμενη φιλοξενία, η υβριδική φιλοξενία και τα ιδιωτικά σύννεφα, τα δημόσια σύννεφα θα είναι αναμφισβήτητα ο τομέας της πιο ουσιαστικής ανάπτυξης. Οι εταιρείες που είναι καλύτερα σε θέση να παρέχουν δημόσιες υπηρεσίες cloud είναι πιθανό να επωφεληθούν περισσότερο. Ως εκ τούτου, οι περισσότερες νέες προσφορές υπηρεσιών cloud στοχεύουν δημόσια σύννεφα.

2.3.9 Προσαρμογή

Οι δημόσιες υπηρεσίες cloud μπορεί να έχουν χιλιάδες πελάτες και έτσι οι υπηρεσίες αυτές δεν μπορούν να ικανοποιήσουν πλήρως τις ανάγκες κάθε πελάτη. Οι πάροχοι συνήθως προσφέρουν υπηρεσίες "αρκετά καλής" και σύνολα χαρακτηριστικών που εκτιμώνται από την πλειονότητα των πελατών. Εάν απαιτούνται πρόσθετες λειτουργίες, δεν υπάρχει εγγύηση ότι ο πάροχος μπορεί να τα προσθέσει.

Πολλές λύσεις δημόσιου cloud BI προσφέρονται και κατασκευάζονται από σχετικά μικρές εταιρείες και νεοσύστατες επιχειρήσεις. Έτσι, δεν προκαλεί έκπληξη το γεγονός ότι οι λύσεις τους ενδέχεται να μην παρέχουν το ίδιο πλούσιο σύνολο χαρακτηριστικών που βρίσκονται σε λύσεις επί τόπου από εταιρείες όπως το Oracle, το Teradata ή το SAP. Οι δημόσιες υπηρεσίες cloud BI ωριμάζουν, αλλά θα χρειαστεί κάποιος χρόνος για να καλύψουν τις καθιερωμένες τεχνολογίες. Οι εταιρείες που σχεδιάζουν να μεταναστεύσουν τις τεχνολογίες BI τους στο δημόσιο σύννεφο πρέπει να επιδείξουν την επιμέλεια τους για να ελέγξουν διεξοδικά την προσφορά σύννεφων και να διασφαλίσουν ότι αυτές οι προσφορές μπορούν να υποστηρίξουν τους στόχους μιας εταιρείας. Με βάση τον ορισμό μας στην αρχή αυτού του εγγράφου, η κύρια διαφορά μεταξύ PaaS και SaaS είναι ότι η PaaS αντιπροσωπεύει κανονικά μια προσαρμόσιμη βάση πλατφόρμας για ανάπτυξη εφαρμογών, ενώ η SaaS παρέχει μια online εφαρμογή που έχει ήδη αναπτυχθεί. Στην περίπτωση των πλατφορμών cloud BI (καθώς και κάποιου άλλου λογισμικού που βασίζεται σε cloud), υπάρχει μια θολή γραμμή μεταξύ της ταξινόμησης PaaS και SaaS. Για παράδειγμα, η προσφορά BigQuery BI της Google μπορεί να θεωρηθεί και ως λύση SaaS και PaaS - είναι SaaS για χρήστες που κάνουν ερωτήματα ενάντια σε αυτήν, αλλά PaaS για προγραμματιστές που χρησιμοποιούν ένα API για να το προγραμματίσουν.

Για να αναλύσουμε την τρέχουσα αγορά τεχνολογίας cloud computing και να υποθέσουμε για τις μελλοντικές τάσεις, μπορούμε να σχεδιάσουμε ένα ιστορικό παράλληλο για να συγκρίνουμε τη διαταραχή που προκαλείται από την τεχνολογία cloud computing με τις παρεμβάσεις άλλων παρελκυστικών προϊόντων στην αγορά.

• Με μακροπρόθεσμες δεσμεύσεις πελατών, οι πάροχοι μπορούν να προγραμματίσουν καλύτερα την χωρητικότητα σύννεφων τους για τα επόμενα χρόνια. Η ικανότητα πρόβλεψης της ικανότητας βοηθά τους παρόχους να κάνουν σωστές αποφάσεις για τις κεφαλαιουχικές δαπάνες (CapEx).

• Υπάρχουν μικρά εμπόδια για την είσοδο σε ορισμένα τμήματα αγοράς νέφους (όπως το IaaS) και οι νεοεισερχόμενοι διαταράσσουν περιοδικά την αγορά προσφέροντας νέες τεχνολογίες και ανταγωνιστικές διαδικασίες. Αυτό δημιουργεί μια πρόκληση για τους καθιερωμένους παρόχους cloud και καθιστά πιο δύσκολη τη διατήρηση των πελατών. Οι προπληρωμένες υπηρεσίες διασφαλίζουν τη μακροπρόθεσμη δέσμευση των πελατών.

• Οι πάροχοι θέλουν να εμποδίσουν τους πελάτες να χρησιμοποιούν τις υπηρεσίες άλλων παρόχων.

Ως εκ τούτου, στην τρέχουσα αγορά οι πελάτες μπορούν να εξασφαλίσουν πολύ καλύτερες τιμές εάν δεσμευτούν σε έναν πάροχο. Σε πολλές περιπτώσεις, η διαφορά τιμής μεταξύ της πληρωμής και της προπληρωμένης τιμολόγησης μπορεί να είναι πολύ μεγάλη.

Το μοντέλο SaaS ακολουθεί την αρχή της οικονομίας της κλίμακας - πρόκειται για μια επιχείρηση χαμηλού κόστους και μεγάλου όγκου. Η διαδικασία απόκτησης πελάτη είναι δαπανηρή. Το Salesforce και άλλοι πάροχοι SaaS έχουν χαμηλά περιθώρια κέρδους επειδή πρέπει να επενδύσουν σε μεγάλο βαθμό στις πωλήσεις και την εμπορία των προϊόντων τους. Θα πρέπει να μειώσουν αυτή την επένδυση στις πωλήσεις και το μάρκετινγκ σε κάποιο σημείο όταν η αγορά SaaS ωριμάζει και εδραιώνεται. Για να παράσχουν βιώσιμη ανάπτυξη για την επιχείρησή τους, οι πάροχοι SaaS πρέπει να διατηρήσουν χαμηλές τις τιμές των πελατών και να επιταχύνουν συνεχώς τον ρυθμό εξαγοράς πελατών.

Πρέπει επίσης να διατηρήσουν τις τιμές τους χαμηλές για να ανταγωνιστούν τους πωλητές CRM επί τόπου, τους άλλους παίκτες του SaaS και τους νεοεισερχόμενους στην αγορά. [34]

Τα διαθέσιμα μοντέλα του Cloud Computing που σχετίζονται με τις
υπηρεσίες που παρέχονται είναι τα εξής :

Software as a Service (SaaS):

Η λογική που βασίζεται το Software as a Service δεν αντικατοπτρίζεται στην αγορά της άδειας χρήσης ενός λογισμικού, αλλά στην υπενοικίαση του από έναν πάροχο υπηρεσιών. Το συγκεκριμένο λογισμικό λειτουργεί σε κεντρικό δίκτυο server και διατίθεται από το διαδίκτυο ως υπηρεσία.

Platform as a Service (PaaS) :

Το συγκεκριμένο μοντέλο παρέχει στους χρήστες τις κατάλληλες υπηρεσίες, με σκοπό να μπορέσουν να αναπτύξουν, να διαθέσουν και να συντηρήσουν εφαρμογές και υπηρεσίες σε ένα ενιαίο περιβάλλον πλατφόρμας, το οποίο είναι ευέλικτο και διαθέσιμο, δίνοντας τη δυνατότητα αυτοδιαχείρισης, αυτό-συντήρησης και αυτό-κλιμάκωσης του λειτουργικού συστήματος, της υποδομής και της πλατφόρμας εφαρμογών. Το μοντέλο που βασίζεται το PaaS είναι το «Pay-per-use», το οποίο δίνει τη δυνατότητα να αξιοποιούνται πλήρως οι υπολογιστικοί πόροι που χρησιμοποιούνται, σε σχέση με το κόστος της χρήσης.

Infrastructure as a Service (IaaS) :

Σύμφωνα με το μοντέλο αυτό, ο χρήστης έχει τη δυνατότητα να υπενοικιάσει μόνο την υποδομή (χωρίς τη πλατφόρμα όπως αναλύσαμε παραπάνω στο PaaS), με την ίδια λογική που χρησιμοποιεί το PaaS (Pay-per-use), αντί να αγοράσει εξοπλισμό ή να συνάψει συμβόλαιο παροχής υπηρεσιών φιλοξενίας υποδομής για το συγκεκριμένο χρονικό διάστημα. Ένα μεγάλο προτέρημα του συγκεκριμένου μοντέλου είναι ότι μπορεί να μεταφέρει άμεσα εικονικές μηχανές, από την εταιρία ή τον ιδιώτη στο cloud, με διαδικασίες συνοπτικές. Η υποδομή είναι το μέσο για παροχή επεξεργασίας, αποθήκευσης, δικτύου και άλλων βασικών υπολογιστικών πόρων.

2.3.10 Εικονικοποίηση

Η εικονικοποίηση υλικού επέτρεψε την αύξηση της πυκνότητας χρήσης του υλικού και διασφαλίζει ότι οι πόροι υλικού χρησιμοποιούνται πιο αποτελεσματικά. Αυτή είναι μια από τις τεχνολογίες που επιτρέπει την ελαστικότητα και έτσι παρέχει μεγαλύτερη ευελιξία όσον αφορά την ταχύτητα ανάπτυξης, τη δυναμική αυτόματη παροχή και τη διαχείριση του cloud.Το Cloud computing παρέχει διάφορες υπηρεσίες, όπως η υποδομή ως υπηρεσία, η πλατφόρμα ως υπηρεσία, και το λογισμικό ως υπηρεσία. Αυτές οι υπηρεσίες βασίζονται σε ένα μοντέλο πληρωμής ως προς τη χρήση για το cloud των πελατών. Ευνόησε τη μετατροπή της μεγάλης βιομηχανίας πληροφορικής, καθιστώντας την υπηρεσία λογισμικού πιο ελκυστική και εύκολη. Από την πλευρά του χρήστη, το cloud computing επιτρέπει τη χρήση και την ανάπτυξη της εφαρμογής από οπουδήποτε και

οποτεδήποτε με βάση τις απαιτήσεις του. Παρέχει συνεχώς αυτές τις υπηρεσίες με βάση τη ζήτηση των πελατών, διάφορες τεχνολογίες όπως η εικονικοποίηση, η ομαδοποίηση και η εφαρμογή διακομιστή. Η εικονικοποίηση χρησιμοποιείται στο cloud computing για το virtualization του λειτουργικού συστήματος, το σύστημα αποθήκευσης και το datacenter. Αυτές οι εφαρμογές μπορεί να έχουν ξεχωριστή ρύθμιση και διαφορετική μηχανική όπως η άτυπη επικοινωνία, η διευκόλυνση του διαδικτύου, η μεταφορά περιεχομένου και η συνεχής προετοιμασία των πληροφοριών. Η καινοτομία της εικονικοποίησης είναι η καρδιά του διανεμημένου κύκλου ζωής των υπολογιστών.

2.3.11 Google Cloud

Η πλατφόρμα Google Cloud (GCP), η οποία προσφέρεται από την Google, είναι μια σουίτα υπηρεσιών cloud computing που λειτουργεί στην ίδια υποδομή που χρησιμοποιεί η Google εσωτερικά για τα προϊόντα τελικού χρήστη της, όπως η Αναζήτηση Google και το YouTube. Παράλληλα με ένα σύνολο εργαλείων διαχείρισης, παρέχει μια σειρά modular cloud services ,όπως computing, αποθήκευση δεδομένων, ανάλυση δεδομένων και μηχανική μάθηση. Η εγγραφή απαιτεί λεπτομέρειες πιστωτικής κάρτας ή τραπεζικού λογαριασμού. Η πλατφόρμα Google Cloud παρέχει υποδομή, πλατφόρμα και serverless computing environments.

Τον Απρίλιο του 2008, η Google ανακοίνωσε την App Engine, μια πλατφόρμα για την ανάπτυξη και τη φιλοξενία εφαρμογών ιστού στα κέντρα δεδομένων που διαχειρίζεται η Google, η οποία ήταν η πρώτη υπηρεσία cloud computing από την εταιρεία. Η υπηρεσία έγινε γενικά διαθέσιμη τον Νοέμβριο του 2011. Από την ανακοίνωση του App Engine, η Google πρόσθεσε πολλές υπηρεσίες cloud στην πλατφόρμα.

Η πλατφόρμα Google Cloud αποτελεί μέρος του Google Cloud, το οποίο περιλαμβάνει την υποδομή δημόσιου cloud της πλατφόρμας Google Cloud Platform καθώς και το G Suite, επιχειρησιακές εκδόσεις Android και Chrome OS και διεπαφές προγραμματισμού εφαρμογών (API) για μηχανική μάθηση και χαρτογράφηση επιχειρήσεων Υπηρεσίες. [8]

2.3.12 Cloud Dataproc

Το Cloud Dataproc είναι μια διαχειριζόμενη υπηρεσία Apache Spark και Apache Hadoop που επιτρέπει τη χρήση εργαλείων δεδομένων ανοιχτού κώδικα για batch processing, querying, streaming, και machine learning. Το Cloud Automation της Dataproc βοηθά στη γρήγορη δημιουργία clusters, την εύκολη διαχείριση και την εξοικονόμηση χρημάτων, απενεργοποιώντας τα clusters όταν δεν τα χρειάζεται ο χρήστης. [8]

ΚΕΦΑΛΑΙΟ 3: Big Data

3.1 Εισαγωγή

Ο όρος Big Data χρησιμοποιείται κατά κόρον τα τελευταία χρόνια, χωρίς να υπάρχει κάποιος σαφής ορισμός του. Ο ορισμός που έχει επικρατήσει είναι αυτός της Gartner από το 2001: Τα μεγάλα δεδομένα είναι δεδομένα που περιέχουν μεγάλη ποικιλία (Variety), πολύ αυξανόμενο όγκο (Volume) και μεγάλη ταχύτητα παραγωγής (Velocity). Αυτό είναι γνωστά ως τα 3V's. Με απλά λόγια, τα μεγάλα δεδομένα είναι μεγαλύτερα, πιο σύνθετα σύνολα και από πολλές νέες πηγές δεδομένων. Αυτά τα σύνολα δεδομένων είναι τόσο ογκώδη που τα παραδοσιακά λογισμικά επεξεργασίας δεδομένων δεν μπορούν να τα διαχειριστούν. [3]

3.1.1 Τα 5 V's

Τα 3 Vs καλύπτουν τα βασικά χαρακτηριστικά των μεγάλων δεδομένων. Τα τελευταία χρόνια όμως εμφανίζονται και άλλες δύο διαστάσεις: εγκυρότητα (Veracity) και αξία (Value). Πιο αναλυτικά τα 5 V's είναι:

- Volume, ο όγκος αναφέρεται στις τεράστιες ποσότητες δεδομένων που οι οργανισμοί προσπαθούν να αξιοποιήσουν για να βελτιώσουν την λήψη αποφάσεων τους. Οι όγκοι δεδομένων συνεχίζουν να αυξάνονται με πρωτοφανή ρυθμό. Ωστόσο, τι αποτελεί πραγματικά "τεράστιο" όγκο ποικίλλει ανάλογα με τον κλάδο και την εταιρεία. Κάποιες εταιρίες ή οργανισμοί διαχειρίζονται petabytes δεδομένων, ενώ άλλες zetabytes. Είναι σίγουρο ότι οτιδήποτε θεωρείται "μεγάλος όγκος" σήμερα θα είναι ακόμη υψηλότερο αύριο .
- Variety, αναφέρεται στους διαφορετικούς τύπους και πηγές δεδομένων. Η ποικιλία αφορά τη διαχείριση της πολυπλοκότητας των πολλών τύπων δεδομένων, συμπεριλαμβανομένων των δομημένων, ημιδομημένων και αδόμητων δεδομένων. Οι οργανισμοί πρέπει να ενσωματώνουν και να αναλύουν δεδομένα από διαφορετικές πηγές δεδομένων, τόσο εντός όσο και εκτός του περιβάλλοντός τους. Με την έκρηξη των αισθητήρων και του IoT, τα δεδομένα παράγονται σε αμέτρητες μορφές, όπως: κείμενο, δεδομένα ιστού, tweets, δεδομένα από αισθητήρες, ήχο, βίντεο και πολλά άλλα

- Velocity, αναφέρεται στην ταχύτητα με την οποία τα δεδομένα δημιουργούνται και διακινούνται, η οποία και συνεχώς αυξάνεται. Η ανάγκη για μεγαλύτερη ταχύτητα πηγάζει από την δημιουργία δεδομένων σε πραγματικό χρόνο και την ανάγκη ενσωμάτωσης της ροής δεδομένων (data streaming) σε επιχειρηματικές διαδικασίες και στη λήψη αποφάσεων. Η ταχύτητα προκαλεί επιπτώσεις στον
«νεκρό χρόνο» (lag time) ο οποίος μεσολαβεί μεταξύ της λήψης των δεδομένων και της αξιοποίησης τους. Τα δεδομένα παράγονται συνεχώς με ρυθμούς που είναι αδύνατο για τα παραδοσιακά συστήματα να λαμβάνουν, να αποθηκεύουν και να αναλύουν σε πραγματικό χρόνο. Για διαδικασίες που είναι ευαίσθητες στις χρονικές καθυστερήσεις, τα δεδομένα πρέπει να αναλύονται σε πραγματικό χρόνο ώστε να έχουν αξία για την επιχείρηση . [3]
- Value, αναφέρεται στην αξία, δηλαδή στην ανάγκη μετατροπής των δεδομένων σε αξία για τους οργανισμούς και τις επιχειρήσεις. Η μετατροπή αυτή γίνεται μέσα από διαδικασίες επεξεργασίας και ανάλυσης. Τα δεδομένα αποτελούν πλέον την ειδοποιό διαφορά για την επιτυχία των επιχειρήσεων . Ένα από τα κορυφαία σκίτσα που περιγράφουν ακριβώς αυτή την έννοια είναι το εξώφυλλο του "The

Economist" τον Μάιο του 2017. Στο εξώφυλλο αυτό, παρομοιάζονται οι μεγάλες εταιρείες δεδομένων όπως το Facebook, η Google, η Amazon και άλλες ότι «εξορύσσουν» δεδομένα, δηλαδή τα δεδομένα πλέον αποτελούν τους «υδρογονάνθρακες» της σύγχρονης οικονομίας (Data is the new oil).

• Veracity, αναφέρεται στην εγκυρότητα και αξιοπιστία των δεδομένων. Η προσπάθεια για υψηλή ποιότητα δεδομένων είναι μια σημαντική απαίτηση και πρόκληση, αλλά ακόμη και οι καλύτερες μέθοδοι «καθαρισμού» δεδομένων (data cleansing) δεν μπορούν να αφαιρέσουν την εγγενή μη προβλεψιμότητα ορισμένων δεδομένων .

Τα μεγάλα δεδομένα είναι ένας συνδυασμός αυτών των χαρακτηριστικών και δημιουργούν ευκαιρίες και ανταγωνιστικό πλεονέκτημα στη σημερινή ψηφιοποιημένη αγορά.

3.1.2 Δομή Big Data

Είναι σημαντικό να εξετάσουμε ενδελεχώς και να αναφέρουμε όλες τις πηγές δεδομένων. Ο γενικός κανόνας είναι ότι όσο περισσότερα δεδομένα, τόσο το καλύτερο. Ωστόσο, τα δεδομένα μπορεί να είναι αναξιόπιστα εξαιτίας ύπαρξης ανακολουθιών (incons
istencies), ελλείψεων (incompleteness), διπλοεγγραφών (duplication) και συγχωνευτικών προβλημάτων (merging). Σε όλα τα στάδια ανάλυσης, εφαρμόζονται διάφοροι μηχανισμοί φιλτραρίσματος ώστε τα δεδομένα να «καθαριστούν» και να μειωθούν σε ένα διαχειρίσιμο μέγεθος. Αξίζει να αναφερθεί το αξίωμα (Garbage in Garbage out - GIGO), το οποίο με απλά λόγια σημαίνει ότι αναξιόπιστα δεδομένα παράγουν αναξιόπιστα μοντέλα και προβλέψεις. Είναι εξαιρετικά σημαντικό όλα τα δεδομένα να περάσουν από το στάδιο προεπεξεργασίας πριν προχωρήσουν σε περαιτέρω

ανάλυση. Ακόμη και το παραμικρό σφάλμα μπορεί να καταστήσει τα δεδομένα εντελώς άχρηστα για περαιτέρω ανάλυση . Για τον λόγο αυτό είναι πολύ σημαντικό να γνωρίζουμε την μορφή, τις πηγές και τους τύπους των δεδομένων μας.

Τα δεδομένα κατηγοριοποιούνται σε δύο βασικές κατηγορίες:

• Δομημένα (structured). Τα περισσότερα δομημένα δεδομένα προέρχονται από συναλλαγές (transactions). Αποτελούν την πρώτη σημαντική πηγή δεδομένων. Οι συναλλαγές αποτελούνται από δεδομένα σε δομημένη μορφή, τα οποία περιγράφουν λεπτομερώς τα βασικά χαρακτηριστικά μιας συναλλαγής ενός πελάτη (π.χ. αγορά, μεταφορά χρημάτων, πληρωμή με πιστωτική κάρτα). Αυτός ο τύπος δεδομένων αποθηκεύεται συνήθως σε σχεσιακές βάσεις δεδομένων επεξεργασίας συναλλαγών (OLTP). Οι συναλλαγές μπορούν επίσης να συνοψιστούν σε μεγαλύτερα χρονικά διαστήματα, και να υπολογιστούν στατιστικά μεγέθη, όπως μέσες τιμές, απόλυτες ή σχετικές τάσεις, μέγιστες ή ελάχιστες τιμές, και ούτω καθεξής.

• Μη δομημένα (unstructured) δεδομένα. Είναι ενσωματωμένα σε έγγραφα κειμένου (π.χ. μηνύματα ηλεκτρονικού ταχυδρομείου, ιστοσελίδες, έγγραφα, κτλ.) ή περιεχόμενο πολυμέσων (βίντεο, εικόνα, ήχο). Τεράστια πηγή τέτοιων δεδομένων είναι και τα social media (Facebook, Twitter, κτλ.). Αυτές οι πηγές απαιτούν εκτεταμένη προεπεξεργασία πριν μπορέσουν να αναλυθούν επιτυχώς

3.1.3 Τύποι Big Data

Είναι σημαντικό να εξεταστούν κατάλληλα οι διαφορετικοί τύποι δεδομένων κατά την έναρξη της ανάλυσης. Τα δεδομένα είναι:

- Συνεχή. Τα συνεχή δεδομένα καθορίζονται σε ένα διάστημα που μπορεί να είναι περιορισμένο ή απεριόριστο. Για παράδειγμα, συνεχή δεδομένα είναι τα ποσά των τραπεζικών συναλλαγών.
- Κατηγορικά. Τα κατηγορικά δεδομένα διακρίνονται σε:

1. Ονομαστικά (nominal): Τα δεδομένα αυτά μπορούν να λάβουν ένα περιορισμένο σύνολο τιμών και η σειρά τους δεν παίζει κανένα ρόλο. Για παράδειγμα, η οικογενειακή κατάσταση και το επάγγελμα είναι τέτοιου είδους δεδομένα.
2. Σειριακά (ordinal): Τα δεδομένα αυτά μπορούν να λάβουν ένα περιορισμένο σύνολο τιμών, η σειρά των οποίων παίζει ρόλο. Για παράδειγμα, η αξιολόγηση πιστοληπτικής ικανότητας, η ηλικία κωδικοποιημένη ως νεαρή, μεσαία και ηλικιωμένη, κτλ.
3. Δυαδικό (binary): Τα δεδομένα αυτά μπορούν να λάβουν μόνο δύο τιμές. Για παράδειγμα, το φύλο, το καθεστώς απασχόλησης, κτλ.

Η κατάλληλη διάκριση μεταξύ αυτών των διαφορετικών τύπων δεδομένων έχει καίρια σημασία κατά την διαδικασία της εισαγωγή των δεδομένων σε ένα λογισμικό για ανάλυση δεδομένων. Για παράδειγμα, εάν η οικογενειακή κατάσταση εισαχθεί εσφαλμένα ως συνεχούς τύπου δεδομένων, τότε το λογισμικό θα θεωρούσε ότι είναι δυνατό να υπολογίσει τη μέση τυπική απόκλιση και ούτω καθεξής, πράγμα που είναι προφανώς χωρίς νόημα.

3.1.4 Πηγές Big Data

Όπως αναφέρθηκε παραπάνω, οι οργανισμοί έχουν μακρά παράδοση να αποθηκεύουν δεδομένα από τις συναλλαγές τους. Εκτός από αυτά, υπάρχουν πάρα πολλές πηγές δεδομένων όπως:

- Web δεδομένα. Τα δεδομένα αυτά περιγράφουν την καταναλωτική συμπεριφορά των πελατών, όπως προβολές σελίδας, αναζητήσεις, κριτικές, αγορές. Η αξιοποίησή τους μπορεί να βελτιώσει τις επιδόσεις σε τομείς όπως οι πωλήσεις, η αξιοπιστία αποπληρωμής από τον πελάτη, η στοχευμένη διαφήμιση, κτλ.
- Δεδομένα από «έξυπνα δίκτυα» (smart grid) και αισθητήρες (IoT). Τα δεδομένα αυτά συλλέγονται από αγωγούς πετρελαίου, ανεμογεννήτριες, περιβαλλοντικούς σταθμούς και πολλές ακόμα πηγές σε εξαιρετικά υψηλή συχνότητα. Τα δεδομένα από αισθητήρες παρέχουν σημαντικές πληροφορίες σχετικά με την απόδοση των μηχανημάτων. Έτσι επιτρέπεται ταχύτερα και πιο εύκολα η διάγνωση των προβλημάτων και επιτυγχάνεται αποτελεσματικότερη συντήρηση. Επίσης, τα GPS και τα κινητά τηλέφωνα αποτελούν μία συνεχώς αναπτυσσόμενη πηγή δεδομένων για την τοποθεσία του χρήστη. Πολλές εταιρείες επιδιώκουν να αξιοποιήσουν τις πολύ σημαντικές αυτές πληροφορίες, δηλαδή ποια στιγμή οι πελάτες βρέθηκαν σε συγκεκριμένες τοποθεσίες.

- Δεδομένα κοινωνικών δικτύων (Social Media). Η ανάλυση των κοινωνικών δικτύων (Facebook, LinkedIn, Instagram) ενός χρήστη μπορεί να δώσει πληροφορίες για τις καταναλωτικές του προτιμήσεις. Αυτό γίνεται λαμβάνοντας υπόψη όχι μόνο τα προσωπικά του ενδιαφέροντα, αλλά και τα

ενδιαφέροντα του κύκλου φίλων ή συναδέλφων του. Τα δεδομένα που καταγράφουν αυτές οι πηγές είναι εικόνες, βίντεο, ηχητικά μηνύματα και podcasts και παρέχουν ποσοτικές και ποιοτικές πληροφορίες για τους χρήστες.

• Ανοιχτά δεδομένα. Είναι δεδομένα που προέρχονται κυρίως από διεθνείς οργανισμούς (Παγκόσμια Τράπεζα, Παγκόσμιος Οργανισμός Υγείας, ΟΗΕ), διεθνείς θεσμούς (Ευρωπαϊκή Ένωση, Eurostat), πανεπιστήμια (UC Irvine, Stanford, MIT), υπουργεία κρατών και στατιστικές υπηρεσίες, επιστημονικούς οργανισμούς (United States Geological Survey), ιδιωτικές πλατφόρμες (Kaggle), από δήμους και κοινότητες (e-government) και πολλές ακόμα πηγές.

3.2 Τεχνικές ανάλυσης Big Data

Η τεράστια ανάπτυξη στον όγκο των δεδομένων έχει δημιουργήσει πολύ μεγάλες ανάγκες, όχι μόνο για την αποθήκευσή τους, αλλά και για την επεξεργασία τους. Στόχος της ανάλυσης των δεδομένων είναι η μετατροπή τους σε πολύτιμες πληροφορίες ή αλλιώς γνώση για τις επιχειρήσεις και τους οργανισμούς. Η ανάγκη αυτή οδήγησε στην δημιουργίας μία τεράστιας ποικιλίας από τεχνικές και εργαλεία ανάλυσης δεδομένων. Τα περισσότερα από αυτά έχουν τις βάσεις τους σε επιστήμες όπως τα μαθηματικά, η στατιστική, η οικονομία και η πληροφορική .

- Βελτιστοποίηση (optimization)
- Στατιστική
- Εξόρυξη δεδομένων (data mining)
- Τεχνικές οπτικοποίησης
- Ανάλυση δικτύων (network analysis)
- Σημασιολογική ανάλυση (semantic analysis)
- Πληθοπορισμός (crowdsourcing)
- Μηχανική μάθηση (machine learning)

3.2.1 Machine Learning

Η μηχανική μάθηση είναι το πεδίο της τεχνητής νοημοσύνης το οποίο χρησιμοποιεί τεχνικές στατιστικής ώστε να δώσε σε έναν υπολογιστή την ικανότητα να μάθει από τα δεδομένα, χωρίς να την χρήση προγραμματισμού. Το 1959, ο Άρθουρ Σάμουελ ορίζει τη μηχανική μάθηση ως «Πεδίο μελέτης που δίνει στους υπολογιστές την ικανότητα να μαθαίνουν, χωρίς να έχουν ρητά προγραμματιστεί» . Η μηχανική μάθηση βρίσκεται στη διασταύρωση της επιστήμης των υπολογιστών, του προγραμματισμού και της στατιστικής και διερευνά τη μελέτη και την κατασκευή αλγορίθμων που μπορούν να μαθαίνουν από τα δεδομένα και να κάνουν προβλέψεις σχετικά με αυτά. [17]

Η μηχανική μάθηση ταξινομείται κυρίως σε τρεις κατηγορίες, αλλά παρόλα αυτά, αναλόγως της περίπτωσης, οι κατηγορίες αυτές μπορούν να συνδυαστούν για να επιτύχουν τα επιθυμητά αποτελέσματα :

- Supervised learning (επιτηρούμενη μάθηση)

Είναι η τεχνική στην οποία το υπολογιστικό πρόγραμμα εκπαιδεύεται ώστε να μάθει τη σχέση μεταξύ διαφόρων μεταβλητών και μιας μεταβλητής στόχου. Το υπολογιστικό πρόγραμμα δέχεται τις παραδειγματικές εισόδους καθώς και τα επιθυμητά αποτελέσματα και ο στόχος είναι να μάθει έναν γενικό κανόνα προκειμένου να αντιστοιχίσει τις εισόδους με τα αποτελέσματα . Οι κύριοι αλγόριθμοι της είναι:

1. Regression (παλινδρόμηση)
2. Classification (ταξινόμηση)
3. Unsupervised learning (μη επιτηρούμενη μάθηση)

Είναι η τεχνική στην οποία ο αλγόριθμος μαθαίνει από μόνος τους χωρίς καμία εποπτεία και χωρίς να υπάρχει κάποια μεταβλητή στόχος. Ο στόχος είναι ο αλγόριθμος να μπορέσει να ανακαλύψει κάποιο κρυφό μοτίβο σχέσεων στα δεδομένα. Οι κύριοι αλγόριθμοι της είναι:

1. Dimensionality reduction (μείωση διαστάσεων)
2. Clustering (ομαδοποίηση)
3. Reinforcement learning (ενισχυτική μάθηση)

Είναι η τεχνική η οποία επιτρέπει στο υπολογιστικό πρόγραμμα να μάθει τη συμπεριφορά του με βάση την ανατροφοδότηση (feedback) από το περιβάλλον. Στην ενισχυτική μάθηση, ο πράκτορας παίρνει μια σειρά αποφάσεων χωρίς επίβλεψη και του δίνεται μια ανταμοιβή +1 ή -1. Με βάση την τελική ανταμοιβή, ο πράκτορας επανεκτιμά τις διαδρομές του. Τα προβλήματα ενισχυτικής μάθησης είναι πιο κοντά στη μεθοδολογία της τεχνητής νοημοσύνης παρά στους συχνά χρησιμοποιούμενους αλγόριθμους μηχανικής μάθησης.

Σε ορισμένες περιπτώσεις, εκτελούμε αρχικά μη επιτηρούμενη μάθηση για να μειώσουμε τις διαστάσεις και ύστερα ακολουθείται επιτηρούμενη μάθηση όταν ο αριθμός των μεταβλητών είναι πολύ υψηλός. Ομοίως, σε ορισμένες εφαρμογές τεχνητής νοημοσύνης, η επιτηρούμενη μάθηση σε συνδυασμό με την ενισχυτική μάθηση χρησιμοποιούνται για την επίλυση ενός προβλήματος, όπως για παράδειγμα, το πρόβλημα της αυτόματης οδήγησης (self-driving) αυτοκινήτων . [17]

3.2.2 Βήματα ανάπτυξης μοντέλου

Η ανάπτυξη μοντέλων μηχανικής μάθησης περιλαμβάνει μια σειρά από βήματα προκειμένου να αναπτυχθούν, να επικυρωθούν (validate) και να εφαρμοστούν. Τα βήματα είναι τα εξής :

- Συλλογή δεδομένων: Τα δεδομένα για τη μηχανική μάθηση συλλέγονται απευθείας από πηγές δομημένων δεδομένων, από δεδομένα internet (web scraping), από API, κ.λπ., καθώς η μηχανική μάθηση μπορεί να λειτουργήσει τόσο με δομημένα όσο και σε μη δομημένα δεδομένα (φωνή, εικόνα και κείμενο).
- Προετοιμασία δεδομένων και επιμέλεια ακραίων τιμών (outliers): Τα δεδομένα πρέπει να μορφοποιηθούν σύμφωνα με τις απαιτήσεις του επιλεγμένου αλγορίθμου. Επίσης, οι ακραίες

τιμές και τα ελλιπή δεδομένα (missing values) πρέπει να αντικατασταθούν με τον μέσο όρο ή την διάμεσο κτλ.

- Ανάλυση δεδομένων και μηχανική χαρακτηριστικών (feature engineering): Τα δεδομένα πρέπει να αναλύονται προκειμένου να εντοπιστούν τυχόν κρυμμένα μοτίβα και σχέσεις μεταξύ των μεταβλητών. Το σωστό feature engineering σε συνδυασμό με κατάλληλες γνώσεις του υπό μελέτη κλάδου (domain knowledge) θα λύσει το 70% των προβλημάτων. Επίσης, στην πράξη, το 70% του χρόνου των αναλυτών δαπανάται για καθήκοντα feature engineering.
- Εκπαίδευση αλγορίθμου στα δεδομένα εκπαίδευσης. Μετά το feature engineering, τα δεδομένα θα χωριστούν σε δύο μέρη (εκπαίδευσης και επαλήθευσης).
- Δοκιμή του αλγορίθμου στα δεδομένα επαλήθευσης. Εάν η ακρίβεια των προβλέψεων είναι αρκετά καλή, μπορούμε να προχωρήσουμε στο επόμενο και τελικό βήμα.
- Ανάπτυξη του αλγορίθμου: Οι εκπαιδευμένοι αλγόριθμοι μηχανικής μάθησης αναπτύσσονται σε πραγματικά δεδομένα. Ένα παράδειγμα θα μπορούσε να είναι τα συστήματα προτάσεων (recommender systems) που εφαρμόζονται από ιστότοπους ηλεκτρονικού εμπορίου.

Επιγραμματικά οι αλγόριθμοι μηχανικής μάθησης είναι:

- Supervised learning
- Classification problems (ταξινόμηση)
- Logistic regression (αλγοριθμική παλινδρόμηση)
- Lasso and ridge regression
- Decision trees (δέντρα αποφάσεων)
- Bagging
- Random forest (τυχαία δέντρα αποφάσεων)
- Boosting (adaboost, gradient boost, and xgboost)
- SVM (Support Vector Machines)
- Recommendation engine (μηχανή προτάσεων)
- Linear regression (γραμμική παλινδρόμηση)
- Unsupervised learning
- Principal component analysis (PCA)
- K-means clustering (συσταδοποίηση)
- Reinforcement learning:
- Markov decision process (αλυσίδα Μαρκόφ)
- Monte Carlo methods (Μέθοδοι Μόντε Κάρλο)
- Temporal difference learning

ΚΕΦΑΛΑΙΟ 4: Hadoop και Spark

4.1 Εισαγωγή

Το Hadoop και το Spark είναι τα πιο γνωστά κατανεμημένα συστήματα ροής δεδομένων. Η τεράστια αποδοχή και χρήση τους βασίζεται στο ότι έχουν προσφέρει πολλαπλές λύσεις σε επίπεδο αποθήκευσης και επεξεργασίας μεγάλου όγκου δεδομένων. Παρακάτω αναλύονται τα χαρακτηριστικά και ο τρόπς λειτουργίας των συστημάτων αυτών.

4.2 Hadoop

Η ανάπτυξη του Hadoop ξεκίνησε το 2005. Το Hadoop είναι ένα πλαίσιο ανοιχτού κώδικα (open source framework) για αξιόπιστη και επεκτάσιμη κατανεμημένη υπολογιστική (distributed computing). Το Hadoop δίνει την δυνατότητα ώστε μεγάλα σύνολα δεδομένων να μπορούν να αποθηκευτούν και επεξεργαστούν αποδοτικά και οικονομικά χρησιμοποιώντας commodity hardware.

Η αποδοτικότητα προέρχεται από την εκτέλεση διεργασιών παράλληλα. Τα δεδομένα δεν χρειάζεται να μετακινούνται μέσω του δικτύου σε έναν κεντρικό κόμβο επεξεργασίας. Αντ 'αυτού, οι διεργασίες επιλύονται με το να διαχωριστούν σε μικρότερες οι οποίες επιλύονται αυτόνομα και στη συνέχεια να συνδυάζουν τα αποτελέσματα τους ώστε να δώσουν ένα τελικό αποτέλεσμα ή απάντηση. Η αποδοτικότητα του κόστους προέρχεται από τη χρήση υλικού. Τα μεγάλα σύνολα δεδομένων απλά διασπώνται και αποθηκεύονται σε σκληρούς δίσκους μέτριας αποθηκευτικής ικανότητας. Οι αποτυχίες αντιμετωπίζονται μέσω του λογισμικού, αντί για servers υψηλού κόστους με χαρακτηριστικά υψηλής διαθεσιμότητας .

4.2.1 Εργαλεία Hadoop

Ο Παγκόσμιος Ιστός δημιουργεί πολλά δεδομένα. Η ανάγκη για αποδοτικότερες αναζητήσεις οδήγησε την Google στην δημιουργία και διατήρηση ευρετηρίου (indexing). Η συνεχόμενη όμως διατήρηση και ενημέρωσή του είναι εξαιρετικά δύσκολη και απαιτεί τεράστιους υπολογιστικούς και αποθηκευτικούς πόρους. Για να καταστεί αυτό δυνατό, η Google δημιούργησε το λογισμικό επεξεργασίας MapReduce. Ο προγραμματισμός MapReduce χρησιμοποιεί δύο λειτουργίες, μια εργασία που μετατρέπει ένα σύνολο δεδομένων σε ζεύγη κλειδιών / τιμών (map) και μια εργασία μείωσης (reduce) που συνδυάζει τα αποτελέσματα εξόδου της εργασίας map σε ένα μόνο αποτέλεσμα. Αυτή η προσέγγιση στην επίλυση προβλημάτων υιοθετήθηκε από προγραμματιστές που εργάζονταν στο “Nutch” πρόγραμμα ανίχνευσης ιστού (web- crawler) του Apache και με βάση αυτή την τεχνολογία ξεκίνησε η ανάπτυξη Hadoop .

Το Hadoop αποτελείται από τέσσερα modules :

1. **MapReduce**: Ένα πρόγραμμα MapReduce αποτελείται από την μέθοδο map, η οποία εκτελεί φιλτράρισμα και ταξινόμηση των δεδομένων και την μέθοδο reduce, η οποία εκτελεί μια συνοπτική λειτουργία υπολογίζοντας το συνολικό αποτέλεσμα της διεργασίας. Το MapReduce είναι ένα μοντέλο προγραμματισμού για την επεξεργασία και παραγωγή μεγάλων συνόλων δεδομένων πάνω σε ένα δίκτυο υπολογιστών (cluster) που αρχικά προτάθηκε από την Google το 2004. Μια εργασία που πρέπει να εκτελεστεί χρησιμοποιώντας το πλαίσιο MapReduce πρέπει να οριστεί σε δύο φάσεις: η φάση του χάρτη όπως ορίζεται από μια συνάρτηση χάρτη (που ονομάζεται mapper) παίρνει ζεύγη κλειδιών / τιμών ως είσοδο, πιθανώς εκτελεί κάποιον υπολογισμό σε αυτήν την είσοδο και παράγει ενδιάμεσα αποτελέσματα με τη μορφή κλειδιού / τιμής ζεύγη, και τη φάση μείωσης που επεξεργάζεται αυτά τα αποτελέσματα όπως καθορίζεται από μια λειτουργία Μείωσης (ονομάζεται μειωτής). Τα δεδομένα από τη φάση χάρτη ανακατεύονται, δηλαδή, ανταλλάσσονται και ταξινομούνται με συγχώνευση, στις μηχανές που εκτελούν τη φάση μείωσης. Πρέπει να σημειωθεί ότι η φάση ανακατέματος μπορεί να είναι πιο χρονοβόρα από την δεύτερη, ανάλογα με τη διαθεσιμότητα εύρους ζώνης δικτύου και άλλους πόρους.

2. **Hadoop Distributed File System (HDFS)**: Αυτό το module υποστηρίζει την κατανεμημένη αποθήκευση μεγάλων αρχείων δεδομένων. Το HDFS χωρίζει τα δεδομένα και τα διανέμει στους κόμβους της συστάδας υπολογιστών (cluster). Δημιουργεί πολλαπλά αντίγραφα των δεδομένων για σκοπούς πλεονασμού και αξιοπιστίας. Εάν ένας κόμβος αποτύχει, το HDFS θα έχει αυτόματα πρόσβαση στα δεδομένα από ένα από τα αντίγραφα που υπάρχουν σε κάποιον άλλο κόμβο. Τα δεδομένα που διαχειρίζεται το HDFS μπορούν να είναι δομημένα ή μη δομημένα και μπορεί να υποστηρίξει σχεδόν οποιαδήποτε μορφή. Το Hadoop δεν απαιτεί τη χρήση του HDFS και μπορούν να χρησιμοποιηθούν και άλλα συστήματα αποθήκευσης όπως το S3 της Amazon Cloud.

3. **Yet Another Resource Negotiator (YARN)** ή αλλιώς σε ελεύθερη μετάφραση "ακόμα ένας διαπραγματευτής πόρων": Το YARN εισήχθη στο Hadoop 2.0 και παρέχει υπηρεσίες προγραμματισμού των διεργασιών και διαχείρισης των υπολογιστικών πόρων του cluster του Hadoop. Μέσα από τα χαρακτηριστικά του YARN, το Hadoop μπορεί να τρέξει και άλλα frameworks πέρα από το MapReduce. Αυτό έχει επεκτείνει τη λειτουργικότητα του Hadoop έτσι ώστε να μπορεί να υποστηρίζει σε πραγματικό χρόνο, διαδραστικούς υπολογισμούς σε ροή (streaming) δεδομένων εκτός από την επεξεργασία εργασιών batch.

4. **Hadoop Common**: Η βιβλιοθήκη και τα εργαλεία που υποστηρίζουν τα 3 παραπάνω modules. Όταν τα δεδομένα διαχειρίζονται από το HDFS, υπάρχει ένα master NameNode που περιέχει το ευρετήριο αρχείων. Τα δεδομένα αποθηκεύονται σε slave DataNodes. Το Hadoop μπορεί να τρέξει σε ένα μόνο μηχάνημα, το οποίο είναι χρήσιμο για πειραματισμό με αυτό, αλλά η δύναμη του είναι να τρέχει σε ένα cluster υπολογιστών. Τα clusters μπορούν να κυμαίνονται από μερικούς μόνο σε χιλιάδες κόμβους. [18]

4.2.2 Το οικοσύστημα του Hadoop

Το Hadoop χρησιμοποιείται συνήθως σε συνδυασμό με αρκετά άλλα εργαλεία της Apache τα οποία αποτελούν ένα πλήρες περιβάλλον Big Data Analytics. Τα εργαλεία αυτά περιλαμβάνουν:

- Pig: Το pig είναι μια scripting γλώσσα για την διαχείριση δεδομένων με λειτουργίες ETL (extract, transform, load). Τα scripts που γράφονται σε "Pig Latin" μετατρέπονται σε εργασίες MapReduce.
- Hive: Το Hive παρέχει μια γλώσσα ερωτημάτων, παρόμοια με την SQL, η οποία μπορεί να συνδεθεί σε μια αποθήκη δεδομένων (data warehouse) που βρίσκεται αποθηκευμένη στο Hadoop.
- Oozie: Το Oozie είναι ένας χρονοπρογραμματιστής (scheduler) εργασιών που χρησιμοποιείται για τη διαχείριση των εργασιών του Hadoop.
- Sqoop: Το Sqoop παρέχει εργαλεία για τη μεταφορά δεδομένων μεταξύ Hadoop και σχεσιακών βάσεων δεδομένων.

Επίσης, υπάρχουν αρκετοί προμηθευτές που παρέχουν έτοιμες πλατφόρμες με προεγκατεστημένα όλα τα λογισμικά του Hadoop. Ορισμένοι τέτοιοι προμηθευτές είναι :

- Υπηρεσίες Web της Amazon: Η AWS μπορεί να παρέχει γρήγορα ένα cluster Hadoop, να προσθέτει πόρους σε αυτό και παρέχει υποστήριξη
- Η πλατφόρμα Google Cloud: Παρόμοια με το AWS, η Google παρέχει γρήγορα ένα cluster Hadoop και συναφείς πόρους.
- Cloudera: Ένας από τους συνδημιουργούς του Hadoop. Η Cloudera προσφέρει ένα πλήρως υποστηριζόμενο περιβάλλον για επιχειρήσεις που θέλουν να εκμεταλλευτούν τις δυνατότητες του Hadoop, συμπεριλαμβανομένων και των επαγγελματικών υπηρεσιών. [18]
- HortonWorks: Η HortonWorks προσφέρει μία έτοιμη πλατφόρμα δεδομένων, βασισμένη στο Hadoop, για την υποστήριξη επιχειρηματικών data lakes και αναλύσεων δεδομένων, συν το Hortonworks DataFlow για συλλογή και ανάλυση δεδομένων σε πραγματικό χρόνο.

4.2.3 Οφέλη και περιορισμοί με τη χρήση του Hadoop

Το Hadoop προσφέρει πολλά πλεονεκτήματα για την επίλυση εφαρμογών Big Data :

- Είναι οικονομικά αποδοτικό: Χρησιμοποιεί commodity hardware
- Επιλύει τα προβλήματα αποδοτικά: Η αποδοτικότητα οφείλεται στη χρήση των πολλαπλών κόμβων για την παράλληλη επεξεργασία των τμημάτων του προβλήματος και στην εκτέλεση υπολογισμών στους κόμβους αποθήκευσης, εξαλείφοντας τις καθυστερήσεις λόγω της μεταφοράς δεδομένων από τον κόμβο αποθήκευσης στον κόμβο υπολογισμού. Επειδή τα δεδομένα δεν μετακινούνται μεταξύ των κόμβων, δεν υπερφορτώνει το δίκτυο.
- Είναι επεκτάσιμο: Οι servers μπορούν να προστεθούν δυναμικά, και κάθε μηχανή που προστίθεται παρέχει μια αύξηση τόσο στην αποθηκευτική όσο και στην υπολογιστική ικανότητα.
- Είναι ευέλικτο: Παρόλο που συνήθως χρησιμοποιείται το MapReduce, είναι δυνατόν να χρησιμοποιηθεί και άλλο framework, όπως το Spark. Τέλος, μπορεί να διαχειρίζεται οποιοδήποτε τύπο δεδομένων, δομημένο ή αδόμητο.

Αυτά τα πλεονεκτήματα και η ευελιξία δεν σημαίνει ότι το Hadoop είναι κατάλληλο για κάθε πρόβλημα. Τα προβλήματα με μικρότερα σύνολα δεδομένων είναι πιθανότερο να λυθούν πιο εύκολα με παραδοσιακές μεθόδους. Το HDFS προορίζεται να υποστηρίζει λειτουργίες "write-once read-many" και δεν προτείνεται για εφαρμογές που χρειάζονται συνεχή ενημέρωση δεδομένων [18] .

Το Hadoop ενδέχεται επίσης να μην είναι η κατάλληλη επιλογή για την αποθήκευση δεδομένων που είναι ιδιαίτερα ευαίσθητα. Παρόλο που διαθέτει δικλείδες ασφαλείας, η προεπιλεγμένη ρύθμιση παραμέτρων τις απενεργοποιεί. Οι διαχειριστές πρέπει να κάνουν τις κατάλληλες επιλογές για να εξασφαλίσουν ότι τα δεδομένα κρυπτογραφούνται και προστατεύονται όπως απαιτείται.
.

4.2.4 Τυπικές περιπτώσεις χρήσης

Το Hadoop χρησιμοποιείται ευρέως από οργανισμούς σε πολλούς διαφορετικούς επιχειρηματικούς τομείς. Το Cloudera παραθέτει 10 κοινά προβλήματα που αρμόζουν στην ανάλυση Hadoop:

1. Risk modeling (μοντελοποίηση κινδύνου)
2. Customer churn analysis (ανάλυση φερεγγυότητας πελάτη)
3. Recommendation engine (μηχανή συστάσεων)
4. Ad targeting (στόχευση διαφημίσεων)
5. Transaction analysis (ανάλυση συναλλαγών)
6. Analyzing network data to predict failure (ανάλυση δεδομένων δικτύου για την πρόβλεψη αποτυχίας)
7. Threat analysis (ανάλυση απειλών)

Οι κλάδοι που τα τελευταία χρόνια εφάρμοσαν το Hadoop στα προβλήματα μεγάλων δεδομένων τους είναι το λιανικό εμπόριο, ο τραπεζικός, η υγειονομική περίθαλψη και πολλοί άλλοι. Ο ιστότοπος Hadoop απαριθμεί πολλές γνωστές εταιρείες με clusters που περιέχουν έως 4500 κόμβους, συμπεριλαμβανομένων των Amazon, EBay, Facebook, Hulu, LinkedIn, Twitter και Yahoo .

4.3 Apache Spark

Το Apache Spark είναι ένα πλαίσιο λογισμικών (framework) για υποστήριξη υπολογισμών σε clusters υπολογιστικών συστημάτων. Το Spark είναι ένα έργο ανοιχτού κώδικα και αναπτύσσεται από μια πολύ εξειδικευμένη κοινότητα προγραμματιστών. [19] Η ανάπτυξή του ξεκίνησε το 2009 ως ερευνητικό έργο στο εργαστήριο RAD Lab του πανεπιστημίου UC Berkeley, το οποίο αργότερα μετονομάστηκε σε AMPLab. Οι ερευνητές στο εργαστήριο είχαν εργαστεί στο παρελθόν για το Hadoop MapReduce και παρατήρησαν ότι το [19]MapReduce ήταν αναποτελεσματικό για διεργασίες που απαιτούσαν πολλές επαναλήψεις (iterations). Έτσι από την αρχή, το Spark σχεδιάστηκε για να είναι γρήγορο για ερωτήματα επαναληπτικών αλγορίθμων και η βασική ιδέα πίσω από την μεγαλύτερη ταχύτητα είναι η αποθήκευση των δεδομένων στην μνήμη RAM κι όχι στον σκληρό δίσκο. Το Spark έχει σχεδιαστεί για να είναι ιδιαίτερα προσβάσιμο, προσφέροντας απλά API σε Python, Java, Scala και SQL και πλούσιες ενσωματωμένες βιβλιοθήκες. Επίσης, συνδυάζεται εξαιρετικά με άλλα εργαλεία Big Data. Ειδικότερα, [2]το Spark μπορεί να τρέξει σε clusters Hadoop και να έχει πρόσβαση σε

οποιαδήποτε πηγή δεδομένων Hadoop, συμπεριλαμβανομένης της NoSQL βάσης Cassandra. Εκτός από το UC Berkeley, κύριοι συντελεστές του Spark είναι το Databricks, η Yahoo και η Intel. [21]

4.3.1 Spark Core

Το Spark Core είναι η υποκείμενη γενική μηχανή εκτέλεσης της πλατφόρμας του Spark στην οποία βασίζονται όλες οι άλλες λειτουργίες. Παρέχει κατανεμημένη αποστολή εργασιών, χρονοδρομολόγηση και βασικές λειτουργίες I/O.

4.3.2 Αρχιτεκτονική

Το Spark χρησιμοποιεί αρχιτεκτονική master/worker. Υπάρχει ένας driver που μιλάει με έναν συντονιστή που ονομάζεται master και διαχειρίζεται τους workers στους οποίους τρέχουν οι executors. Ένας master είναι ένα Spark instance που συνδέεται με έναν Cluster Manager για πόρους και αποκτά κόμβους της συστάδας για να εκτελέσει [19] executors. Από την άλλη πλευρά, οι workers (γνωστοί και ως slaves) [2]αποτελούν Spark instances στα οποία οι executors ζουν για να εκτελούν εργασίες. Αυτοί είναι οι υπολογιστικοί κόμβοι του Spark που επίσης επικοινωνούν μεταξύ τους χρησιμοποιώντας τα Block Manager instances που διαθέτουν.

Ο driver και οι executors τώρα, τρέχουν στις δικές τους Java διαδικασίες. Μπορούν να τρέξουν όλοι στην ίδια (οριζόντια συστάδα) ή σε ξεχωριστές μηχανές (κάθετη συστάδα) ή σε μικτή διαμόρφωση μηχανής. Όταν δημιουργείται ένα SparkContext, κάθε worker εκκινεί έναν executor. Οι executors συνδέονται πίσω στο πρόγραμμα του driver. Τώρα ο driver μπορεί να τους στείλει εντολές, όπως για παράδειγμα flatMap, map και reduceByKey. Όταν ο driver κλείσει, οι executors κλείνουν επίσης.

Συνοπτικά, μια εφαρμογή στο Spark εκτελείται σε τρία στάδια:

1. Δημιουργείται γράφος RDD, δηλαδή ένας DAG (Directed Acyclic Graph) των RDDs ο οποίος αντιπροσωπεύει ολόκληρο τον υπολογισμό.
2. Δημιουργείται ένας γράφος σταδίων, δηλαδή DAG των σταδίων ο οποίος είναι ένα λογικό σχέδιο εκτέλεσης που βασίζεται στον γράφο RDD. Τα στάδια δημιουργούνται με το σπάσιμο του γράφου RDD σε τυχαία όρια.
3. Με βάση το σχέδιο, χρονοδρομολογούνται και εκτελούνται τα καθήκοντα στους workers.

4.3.3 Spark SQL

Το Spark SQL είναι το module του Spark για την εργασία με δομημένα δεδομένα. Επιτρέπει την ανάκτηση δεδομένων μέσω SQL και υποστηρίζει πολλές πηγές δεδομένων, όπως πίνακες Hive, αρχεία Parquet και JSON. Παρέχει δομή δεδομένων που ονομάζεται DataFrames και επιτρέπει στα ερωτήματα του Hive να τρέχουν μέχρι και 100 φορές ταχύτερα. Παρέχει επίσης ισχυρή ενσωμάτωση με το υπόλοιπο οικοσύστημα Spark (π.χ. ενοποίηση της επεξεργασίας ερωτημάτων SQL με μηχανική μάθηση) .

4.3.4 Spark ML Library

Το Spark παρέχει και μια βιβλιοθήκη για μηχανική μάθηση που ονομάζεται MLlib. Η MLlib παρέχει πολλούς τύπους αλγορίθμων μηχανικής μάθησης, συμπεριλαμβανομένων classification, regression, clustering, και collaborative filtering, καθώς και την υποστήριξη για αξιολόγηση του μοντέλου και εισαγωγή δεδομένων. Όλοι αυτοί οι αλγόριθμοι έχουν σχεδιαστεί για να κλιμακώνονται (scale) στο cluster [26] .

4.3.5 Χρήσεις Spark

Επειδή το Spark είναι ένα framework γενικού σκοπού για κατανεμημένα υπολογιστικά συστήματα, χρησιμοποιείται για ένα ποικίλο φάσμα εφαρμογών. Οι κυριότερες είναι :

- Streaming data (δεδομένα ροής): Η βασική περίπτωση χρήσης του Apache Spark είναι η ικανότητά του να επεξεργάζεται δεδομένα ροής. Η ανάγκη για καθημερινή επεξεργασία δεδομένων σε καθημερινή βάση έχει καταστεί σημαντική απαίτηση για τις επιχειρήσεις ώστε να είναι σε θέση να αναλύσουν τα δεδομένα αυτά σε πραγματικό χρόνο.
- Machine Learning: Μια άλλη από τις πολλές περιπτώσεις χρήσης του Apache Spark είναι οι δυνατότητες μηχανικής μάθησης που παρέχει. Το Spark διαθέτει ένα ολοκληρωμένο πλαίσιο για την εκτέλεση αλγορίθμων μηχανικής μάθησης. Παραπάνω αναφερθήκαμε στην αντίστοιχη βιβλιοθήκη η οποία είναι η MLlib.
- Interactive Analysis (διαδραστική ανάλυση): Μεταξύ των πιο αξιοσημείωτων δυνατοτήτων του Spark είναι η ικανότητά του για διαδραστική ανάλυση. Το MapReduce δημιουργήθηκε για την επεξεργασία batch διεργασιών και οι μηχανές SQL-on-Hadoop, όπως το Hive ή το Pig, είναι [2]συχνά πολύ αργές για διαδραστική ανάλυση. Ωστόσο, το Apache Spark, είναι αρκετά γρήγορο για να διεξάγει εξερευνητικά ερωτήματα χωρίς δειγματοληψία. Το Spark συνδέεται επίσης με πολλές γλώσσες ανάπτυξης, όπως με τις SQL, R και Python. Συνδυάζοντας το Spark με εργαλεία οπτικοποίησης, σύνθετα σύνολα δεδομένων μπορούν να επεξεργαστούν και να απεικονιστούν διαδραστικά.
- Fog computing: Ενώ το Spark είναι ίσως πιο γνωστό για τις αναλύσεις δεδομένων, ο επόμενος μεγάλος στόχος της κοινότητας είναι το Ίντερνετ των πραγμάτων (IoT). Το IoT ενσωματώνει αντικείμενα και συσκευές με μικροσκοπικούς αισθητήρες που επικοινωνούν μεταξύ τους και με τον χρήστη, δημιουργώντας ένα πλήρως διασυνδεδεμένο κόσμο. Αυτός ο κόσμος συγκεντρώνει τεράστια ποσά δεδομένων, τα επεξεργάζεται και προσφέρει επαναστατικά νέα χαρακτηριστικά και εφαρμογές που μπορούν να χρησιμοποιήσουν οι άνθρωποι στην καθημερινότητά τους. Ωστόσο, καθώς το IoT επεκτείνεται τόσο πολύ, υπάρχει ανάγκη για κατανεμημένη μαζική παράλληλη επεξεργασία τεράστιων ποσοτήτων και ποικίλων δεδομένων

τα οποία παράγονται από μηχανές και αισθητήρες. Ωστόσο, ο όγκος αυτός υπολογισμών είναι δύσκολο να διαχειριστεί με τις τρέχουσες δυνατότητες ανάλυσης του cloud. Την δυσκολία αυτή έρχεται να λύσει το fog computing, το οποίο αποκεντροποιεί (decentralizes) την επεξεργασία και αποθήκευση δεδομένων, αντί να εκτελεί αυτές τις λειτουργίες στην άκρη του δικτύου (edge of network)

4.3.6 Resilient Distributed Dataset

Όπως αναφέρθηκε προηγουμένως, το Resilient Distributed Dataset (γνωστό και ως RDD) είναι η πρωταρχική οντότητα δεδομένων του Apache Spark. Ένα RDD είναι μια συλλογή στοιχείων που έχουν ανοχή σε σφάλματα [2]και τα οποία μπορούν να επεξεργάζονται παράλληλα. Η έννοια του RDD καθώς και το όνομά τους εμφανίστηκαν για πρώτη φορά στο δημοσίευση με τίτλο Resilient Distributed Datasets: A Tolerant Fault Abstraction for Computing Cluster In Memory [22].

Τα χαρακτηριστικά των RDDs (αποσυνθέτοντας το όνομα):

- Resilient" δηλ. ανεκτικά σε σφάλματα, με τη βοήθεια ενός γράφου «ζωής» κάθε RDD. Είναι καθ' αυτόν τον τρόπο ικανά να υπολογίσουν εκ νέου κομμάτια των δεδομένων που λείπουν ή έχουν υποστεί ζημιά εξαιτίας αποτυχιών ενός κόμβου.
- Distributed: τα δεδομένα βρίσκονται διαμοιρασμένα σε πολλούς κόμβους ενός cluster.
- Dataset: μια συλλογή δεδομένων που είναι χωρισμένη σε κομμάτια (partitioned collection).

Τα RDDs μπορούν να δημιουργηθούν με δύο τρόπους: είτε με παραλληλισμό μιας υπάρχουσας συλλογής δεδομένων στο πρόγραμμα οδήγησης (driver program) ή με αναφορά ενός συνόλου δεδομένων από οποιαδήποτε πηγή αποθήκευσης υποστηριζόμενη από τον Hadoop (συμπεριλαμβανομένων του τοπικού συστήματος αρχείων, το HDFS, κ.λπ.). Το Spark, υποστηρίζει αρχεία κειμένου, SequenceFiles και οποιαδήποτε άλλο τύπο αρχείου εισόδου του Hadoop [18]. Μια σημαντική παράμετρος για τις παράλληλες συλλογές, είναι ο αριθμός των κατατμήσεων για την αποκοπή του συνόλου δεδομένων. Το Spark εκτελεί μία εργασία για κάθε κομμάτι της συλλογής δεδομένων. Επιπλέον, τα RDDs επιδέχονται δύο τύπους ενεργειών: τους μετασχηματισμούς (transformations), οι οποίοι δημιουργούν ένα νέο σύνολο δεδομένων από ένα υπάρχον και τις δράσεις (actions), οι οποίες επιστρέφουν μια τιμή στο πρόγραμμα οδήγησης μετά την εκτέλεση ενός

υπολογισμού στο σύνολο των δεδομένων. Για παράδειγμα, η συνάρτηση map() είναι ένας μετασχηματισμός που μεταβιβάζει κάθε στοιχείο ενός συνόλου δεδομένων μέσω μιας συνάρτησης και επιστρέφει ένα νέο RDD που περιέχει τα αποτελέσματα. Από την άλλη πλευρά, η reduce() αποτελεί μια ενέργεια που συγκεντρώνει όλα τα στοιχεία ενός RDD, χρησιμοποιώντας κάποια συνάρτηση, και επιστρέφει το τελικό αποτέλεσμα στο [14]πρόγραμμα οδήγησης. Προκειμένου το Spark να είναι πιο αποδοτικό, όλοι οι μετασχηματισμοί σε ένα RDD είναι lazy, με την έννοια ότι δεν υπολογίζουν τα αποτελέσματά τους μέχρι μία «δράση» να απαιτήσει την επιστροφή ενός αποτελέσματος στο πρόγραμμα οδήγησης.

Μία από τις πιο σημαντικές δυνατότητες του Spark είναι ότι τα δεδομένα των RDD μπορούν να αποθηκευτούν προσωρινά στη μνήμη ή το δίσκο. Με αυτόν τον τρόπο, τα στοιχεία ενός RDD που είχε προηγουμένως υποστεί έναν μετασχηματισμό μπορούν να προσπελαστούν πολύ ταχύτερα, δεδομένου ότι οι νέες ενέργειες πάνω σε αυτό δεν απαιτούν εκ νέου υπολογισμούς. Η προσωρινή αποθήκευση είναι ένα σημαντικότατο [30] εργαλείο όσον αφορά τους επαναληπτικούς αλγόριθμους, όπου η επαναχρησιμοποίηση των δεδομένων είναι μεγάλη και συχνή.

4.3.7 RDD Lineage

Όταν δημιουργείται ένα νέο RDD από ένα υπάρχον, το νέο RDD φέρει ένα ID που το συνδέει με το parent RDD του Spark. Αυτές οι εξαρτήσεις μεταξύ αυτών αποθηκεύονται προσωρινά σε ενα Γράφο που ονομάζετε Lineage Graph. Το RDD lineage είναι η γραφική παράσταση όλων των parent RDDs. Το ονομάζουμε επίσης RDD operator graph ή RDD dependency graph και αποτελεί το αποτέλεσμα της εφαρμογής του Transformation του Spark. Μετά την δημιουργία του Logical Execution Plan, ξεκινάει το process με τα παλαιότερα (αυτά που δεν έχουν εξαρτήσεις από άλλα RDD ή δεδομένα προσωρινής αποθήκευσης-cached) και τελειώνει με το αυτό που παράγει το αποτέλεσμα της ενέργειας που έχει κληθεί να εκτελέσει.

4.3.8 RDD Persistence και Caching στο Spark

Το Persistence του Spark RDD είναι μια τεχνική βελτιστοποίησης στην οποία αποθηκεύεται το αποτέλεσμα της αξιολόγησης RDD. Με αυτόν τον τρόπο δίνεται η δυνατότητα αποθήκευσης του

ενδιάμεσου αποτελέσματος, ώστε να χρησιμοποιείται περαιτέρω, αν χρειαστεί και μειώνει τα γενικά έξοδα υπολογισμού.

Τα RDD δύναται να γίνουν persist, μέσω των μεθόδων cache () και persist (). Με τη μέθοδο cache () μπορεί να αποθηκευτεί ένα ολόκληρο RDD μέσα στη μνήμη έτσι ώστε διατηρώντας στη μνήμη να χρησιμοποιηθεί αποδοτικά σε παράλληλες λειτουργίες.

Η διαφορά μεταξύ cache () και persist () είναι ότι χρησιμοποιώντας την προσωρινή μνήμη () το προεπιλεγμένο επίπεδο αποθήκευσης είναι 'MEMORY_ONLY' ενώ το persist () δίνει τη δυνατότητα να χρησιμοποιηθούν διάφορα επίπεδα αποθήκευσης (περιγράφονται παρακάτω). Είναι ένα βασικό εργαλείο για έναν διαδραστικό αλγόριθμο, επειδή κάθε κόμβος αποθηκεύει οποιοδήποτε partition που μπορεί να υπολογιστεί στη μνήμη και το καθιστά επαναχρησιμοποιήσιμο για μελλοντική χρήση. Αυτή η διαδικασία επιταχύνει τον περαιτέρω υπολογισμό x10 φορές.
Όταν υπολογίζεται το RDD για πρώτη φορά, διατηρείται στη μνήμη του κόμβου. Η μνήμη cache του Spark είναι ανεκτική σε σφάλματα (Fault Tolerant), οπότε κάθε φορά που χάνονται κάποια διαμερίσματα του RDD, μπορεί να ανακτηθεί με τη λειτουργία μετασχηματισμού που το δημιούργησε αρχικά.

Storage levels του RDDs persistence

Storage level	**Characteristics**
memory_only	Αποθήκευση δεδομένων στη μνήμη, εάν είναι δυνατή. Όταν το μέγεθος RDD είναι υψηλότερο από το μέγεθος της μνήμης, δεν θα αποθηκεύει προσωρινά τα partitions που δεν έχουν αρκετό χώρο, συνεπώς, δεν θα επαναυπολογίζει αυτά τα partitions όποτε απαιτείται. Αυτό το επίπεδο παρέχει πολύ μεγάλο χώρο αποθήκευσης και μειώνει τον χρόνο υπολογισμού της CPU
memory_and_disk	Σε αυτό το επίπεδο, είναι δυνατή η αποθήκευση των partitions στο δίσκο αν δεν υπάρχει αρκετός χώρος στη μνήμη. Κατά συνέπεια, θα ανακτήσει αυτά τα partitions όποτε απαιτείται. Αυτό το επίπεδο παρέχει πολύ μεγάλο χώρο αποθήκευσης και μειώνει τον χρόνο υπολογισμού της CPU
disk_only	Αποθηκεύοντας όλα τα partitions μόνο στο δίσκο, παρέχει περισσότερο χώρο αποτελεσματικό. Για αυτό το επίπεδο, ο χώρος αποθήκευσης γίνεται μικρός και ο χρόνος υπολογισμού καθίσταται υψηλός
memory_only_ser	Σε αυτό το επίπεδο αποθηκεύει το RDD ως σειριακό αντικείμενο Java και μόνο στη μνήμη. Παρέχει περισσότερο αποδοτικό χώρο σε σύγκριση με τα επίπεδα που δεν έχουν επιλεγεί. Ωστόσο, αυξάνει το γενικό κόστος της CPU. Για αυτό το επίπεδο, ο χώρος αποθήκευσης γίνεται μικρός και ο χρόνος υπολογισμού καθίσταται υψηλός
memory_and_disk_ser	Σε αυτό το επίπεδο αποθηκεύει το RDD ως σειριακό αντικείμενο Java στη μνήμη και στο δίσκο. Παρέχει περισσότερο αποδοτικό χώρο σε σύγκριση με το επίπεδο που είναι αποστειρωμένο. Ωστόσο, αυξάνει το γενικό κόστος της CPU. Για αυτό το επίπεδο, ο χώρος αποθήκευσης γίνεται μικρός και ο χρόνος υπολογισμού καθίσταται υψηλός

4.3.9 Inside Spark Application

Ένα Spark application δημιουργείται στο interface του Hadoop cluster τη στιγμή που εκινείται ένας αλγόριθμος. [30]

Τα βασικά μέρη που αποτελούν το application είναι τα εξής:

Job: - Ένα κομμάτι κώδικα που διαβάζει το dataset από το HDFS ή τον τοπικό δίσκο, εκτελεί υπολογισμούς στα δεδομένα και γράφει κάποια δεδομένα εξόδου.

Stages: - Τα Jobs χωρίζονται σε stages. Τα stages ταξινομούνται σε Map και Reduce stages. Χωρίζονται με βάση υπολογιστικά όρια, όλοι οι υπολογισμοί (operators) δεν μπορούν να ενημερωθούν σε ένα μόνο στάδιο. Αυτό συμβαίνει σε πολλά στάδια. [15]

Tasks: - Κάθε stage έχει tasks. Ένα task σε κάθε partition. Ένα task εκτελείται σε ένα partition δεδομένων σε έναν executor (machine).

DAG - Το DAG σημαίνει Directed Acyclic Graph, στο παρόν πλαίσιο είναι ένα DAG των operators. Μετά την εισαγωγή του DAG στο Spark, το σχέδιο [15]εκτέλεσης βελτιστοποιείται, π.χ. για να ελαχιστοποιούνται τα δεδομένα ανακατεύθυνσης, δεδομένου ότι σχηματίζεται ένα DAG (Κατευθυνόμενο Ακυκλικό Γράφημα) διαδοχικών σταδίων υπολογισμού

DAGScheduler - Ο DAGScheduler υπολογίζει ένα κατευθυνόμενο ακυκλικό γράφημα (DAG) των σταδίων για κάθε εργασία, παρακολουθεί ποια είναι τα RDD και οι εξόδους του σταδίου και υλοποιεί ένα ελάχιστο χρονοδιάγραμμα για την εκτέλεση εργασιών. Στη συνέχεια υποβάλλει στάδια στο TaskScheduler.

Executor - Η διαδικασία που είναι υπεύθυνη για την εκτέλεση των tasks.

Driver - Το πρόγραμμα / διαδικασία που είναι υπεύθυνο για την εκτέλεση ενός Job μέσω του Spark Engine

Master - The machine on which the Driver program runs

Slave - The machine on which the Executor program runs

Όλα τα jobs στο spark περιλαμβάνουν μια σειρά από operators και τρέχουν σε ένα σύνολο δεδομένων. Όλοι οι operators σε ένα job χρησιμοποιούνται για να κατασκευάσουν ένα DAG (Directed Acyclic Graph). Το DAG βελτιστοποιείται με αναδιατάξεις και συνδυασμούς χειριστών όπου είναι δυνατόν.
Για παράδειγμα, ας υποθέσουμε ότι πρέπει να υποβάλετε ένα job στο Spark η οποία περιέχει ένα Map operation που ακολουθείται από μια λειτουργία φίλτρου. Ο μηχανισμός βελτιστοποίησης DAG του Spark θα αναδιατάξει τη σειρά αυτών των χειριστών, καθώς το φιλτράρισμα θα μειώσει τον αριθμό των records που θα υποβληθούν στον Mapper.

Το Spark έχει μια μικρή βάση κώδικα και το σύστημα χωρίζεται σε διάφορα στρώματα. Κάθε στρώμα έχει κάποιες ευθύνες. Τα στρώματα είναι ανεξάρτητα το ένα από το άλλο

1. Το πρώτο στρώμα είναι ο interpreter (διερμηνέας). Το Spark χρησιμοποιεί έναν Scala interpreter, με μερικές τροποποιήσεις.
2. Καθώς εισάγετε τον κωδικό σας στην κονσόλα του Spark (δημιουργώντας RDD και εφαρμόζοντας operators), το Spark δημιουργεί ένα graph operator.
3. Όταν ο χρήστης εκτελεί ένα action (όπως το collect), ο Γράφος υποβάλλεται σε έναν DAG Scheduler. Ο DAG Scheduler διαιρεί τον operator του γράφου σε (Map και Reduce) stages.
4. Ένα stage αποτελείται από εργασίες που βασίζονται στα partitions των δεδομένων εισόδου. Ο DAG scheduler ταξινομεί σε σειρά όλους τους operators (pipelines operators together) για να βελτιστοποιήσει το γράφο. π.χ. Πολλοί Map operators μπορούν να προγραμματιστούν σε ένα μόνο stage. Αυτή η βελτιστοποίηση είναι το κλειδί για την απόδοση του Spark. Το αποτέλεσμα ενός DAG scheduler είναι ένα σύνολο σταδίων.
5. Τα στάδια μεταβιβάζονται στον Task Scheduler. Ο task scheduler εκκινεί διεργασίες μέσω του cluster manager. (Spark Standalone / Yarn / Mesos). Ο task scheduler δεν γνωρίζει τα dependencies μεταξύ των stages.
6. Ο Worker εκτελεί τα tasks στον Slave. Ένα νέο JVM ξεκινάει ανά JOB. Ο Worker γνωρίζει μόνο τον κώδικα που έχει περάσει σε αυτόν.

Το Spark αποθηκεύει τα δεδομένα που πρέπει να επεξεργαστούν, επιτρέποντάς του να τρέχει 100 φορές ταχύτερα από το hadoop. Το Spark χρησιμοποιεί το Akka για το Multithreading, διαχειρίζεται την κατάσταση του executor, προγραμματίζοντας χρονικά τα tasks.

Χρησιμοποιεί το Jetty για να μοιράζεται αρχεία (π.χ. Jars), Http Broadcast και τρέχει το Spark Web UI. Το Spark είναι ιδιαίτερα διαμορφωμένο και μπορεί να αξιοποιήσει τα υπάρχοντα συστατικά(components) που υπάρχουν ήδη στο Hadoop EcoSystem. Αυτό επέτρεψε στο Spark να αναπτυχθεί εκθετικά, και σε λίγο χρόνο αυξήθηκε κατά πολύ η εμπορευματική του αξία, καθώς πολλές εταιρείες το χρησιμοποιούν ήδη στην παραγωγή.

4.3.10 Serialization in Spark

Serialization είναι μια διαδικασία μετατροπής ενός αντικειμένου σε μια ακολουθία bytes που μπορεί να επιμείνει σε δίσκο ή βάση δεδομένων ή μπορεί να αποσταλεί μέσω ροών. Η αντίστροφη διαδικασία δημιουργίας αντικειμένου από ακολουθία bytes ονομάζεται deserialization.

Στο Spark τα objects που δημιουργούνται χρειάζονται serialization ώστε να μπορούν να σταλούν στα Worker Nodes. Ενώ οι κανόνες για το serialization φαίνονται αρκετά απλοί, η ερμηνεία τους όμως σε μια σύνθετη βάση κώδικα μπορεί να καταλήξει σε πολυσύνθετη διαδικασία! Σε περίπτωση που ο κώδικας να περιλαμβάνει περιττά κομμάτια, υπάρχει περίπτωση αύξηση του χρόνου εκτέλεσης όπου κάποια objects μπορεί να μην γίνουν serialized.

Κανόνες του Serialization

Πότε τα αντικείμενα πρέπει να σειριοποιηθούν?

Όταν εκτελείτε μια λειτουργία σε ένα RDD ή σε οτιδήποτε είναι μια αφαίρεση πάνω από αυτό (π.χ. Dataframes, Datasets), είναι συνηθισμένο ότι αυτή η λειτουργία θα πρέπει να είναι σειριοποιημένη έτσι ώστε να μπορεί να αποσταλεί σε κάθε κόμβο του Worker Node και αυτός να εκτελέσει τους υπολογισμούς στο τμήμα των δεδομένων του.

Τι γίνεται Serialized?

Οι κανόνες για το τι μπορεί να γίνει Serialized είναι ίδιοι με τους κανόνες της Java - μόνο τα αντικείμενα μπορούν να είναι σειριοποιημένα.
Η συνάρτηση που μεταβιβάζεται στο map (ή παρόμοια λειτουργία των Spark RDD) θα πρέπει να είναι Serialized. Αν οι παραπομπές σε άλλα αντικείμενα γίνονται μέσα σε αυτή τη λειτουργία τότε αυτά τα αντικείμενα θα πρέπει επίσης να είναι σειριοποιημένα. Το σύνολο αυτών των αντικειμένων θα είναι σειριοποιημένο, ακόμη και όταν προσπελάσει μόνο ένα από τα πεδία τους.

Η σειριοποίηση χρησιμοποιείται συνήθως για communication (κοινή χρήση αντικειμένων μεταξύ πολλών υπολογιστών) και persistence (αποθήκευση της κατάστασης του αντικειμένου σε ένα αρχείο ή μια βάση δεδομένων). Πρόκειται για ένα αναπόσπαστο μέρος δημοφιλών πρωτοκόλλων όπως το Remote Invocation (RMI), το Java Management Extension (JMX), το Java Messaging System (JMS), το Μορφή μηνύματος δράσης (AMF), το Java Server Faces (JSF) ViewState κ.λ.

Deserialization των μη αξιόπιστων δεδομένων (CWE-502) είναι όταν η εφαρμογή διαγράφει τα μη αξιόπιστα δεδομένα χωρίς να διασφαλίζει ότι τα δεδομένα που προκύπτουν θα είναι έγκυρα, επιτρέποντας στον attacker να ελέγξει την κατάσταση ή τη ροή της εκτέλεσης.

Τα προβλήματα του deserialization της Java είναι γνωστά εδώ και χρόνια. Ωστόσο, το ενδιαφέρον για το ζήτημα εντάθηκε σε μεγάλο βαθμό το 2015, όταν σε μια δημοφιλή βιβλιοθήκη (Apache Commons Collection) βρέθηκαν κλάσεις που θα μπορούσαν να χρησιμοποιηθούν καταχρηστικά για την απομακρυσμένη εκτέλεση κώδικα. Αυτές οι κλάσεις χρησιμοποιήθηκαν στις zero-days επηρεάζοντας το IBM WebSphere, το Oracle WebLogic και πολλά άλλα προϊόντα.

Ένας attacker πρέπει απλώς να εντοπίσει ένα κομμάτι λογισμικού που έχει ταυτόχρονα μια ευάλωτη κλάση στην πορεία του και εκτελεί deserialization σε μη αξιόπιστα δεδομένα. Τότε το μόνο που χρειάζεται να κάνουν είναι να στείλουν το ωφέλιμο φορτίο στο deserializer, εκτελώντας την εντολή.

ΚΕΦΑΛΑΙΟ 5: ΑΛΓΌΡΙΘΜΟΙ ΚΑΤΗΓΟΡΙΟΠΟΊΗΣΗΣ

5.1 Λογιστική Παλινδρόμηση

Στα στατιστικά στοιχεία, το μοντέλο logistics (ή το μοντέλο logit) χρησιμοποιείται για να μοντελοποιήσει την πιθανότητα μιας συγκεκριμένης κατηγορίας ή συμβάντος όπως pass / fail, win / lose, ζωντανός / νεκρός ή υγιής / άρρωστος. Η λογική παλινδρόμηση είναι ένα στατιστικό μοντέλο που στη βασική της μορφή χρησιμοποιεί μια λογική λειτουργία για να μοντελοποιήσει μια δυαδική εξαρτώμενη μεταβλητη, αν και υπάρχουν [27]πολύ πιο πολύπλοκες επεκτάσεις. Στην ανάλυση παλινδρόμησης, η λογική παλινδρόμηση εκτιμά τις παραμέτρους ενός λογοτεχνικού μοντέλου (μορφή δυαδικής παλινδρόμησης). Μαθηματικά, ένα δυαδικό λογικό μοντέλο έχει μια εξαρτημένη μεταβλητή με δύο πιθανές τιμές, όπως το pass / fail που αντιπροσωπεύεται από μια μεταβλητή δείκτη, όπου οι δύο τιμές είναι επισημασμένες με "0" και "1". Στο logistic μοντέλο, η log-odds (ο λογάριθμος των αποδόσεων) για την τιμή με την ένδειξη "1" είναι ένας γραμμικός συνδυασμός μιας ή περισσοτέρων ανεξάρτητων μεταβλητών ("predictors"). οι ανεξάρτητες μεταβλητές μπορούν να είναι μια δυαδική μεταβλητή (δύο κλάσεις, κωδικοποιημένες από μια μεταβλητή δείκτη) ή μια συνεχής μεταβλητή (οποιαδήποτε πραγματική τιμή). [1] Η αντίστοιχη πιθανότητα της τιμής "1" μπορεί να κυμαίνεται μεταξύ 0 (βεβαίως η τιμή "0") και 1 (βεβαίως η τιμή "1"), εξ ου και η επισήμανση. η συνάρτηση που μετατρέπει τα log-odds στην πιθανότητα είναι η λογική λειτουργία, εξ ου και το όνομα. Η μονάδα μέτρησης για την κλίμακα log-odds ονομάζεται logit, από την εφοδιαστική μονάδα. Μπορούν επίσης να χρησιμοποιηθούν ανάλογα μοντέλα με διαφορετική λειτουργία sigmoid αντί της λογικής λειτουργίας, όπως το μοντέλο probit. το καθοριστικό χαρακτηριστικό του μοντέλου είναι ότι η αύξηση μιας από τις ανεξάρτητες [26]μεταβλητές πολλα[1]πλασιάζει πολλαπλασιαστικά τις αποδόσεις του δεδομένου αποτελέσματος με σταθερό ρυθμό, με κάθε ανεξάρτητη μεταβλητή να έχει τη δική της παράμετρο. [44]

5.2 Naive Bayes ταξινομητής

Στη μηχανική μάθηση, οι αφελείς ταξινομητές Bayes είναι μια οικογένεια απλών πιθανοτικών ταξινομητών» που βασίζονται στην εφαρμογή του θεώρημα του Bayes με ισχυρές (αφηρημένες) υποθέσεις ανεξαρτησίας μεταξύ των χαρακτηριστικών. Το Naive Bayes έχει μελετηθεί εκτενώς από τη δεκαετία του 1960. Εισήχθη (αν και όχι κάτω από αυτό το όνομα) στην κοινότητα ανάκτησης κειμένου στις αρχές της δεκαετίας του 1960 και παραμένει μια δημοφιλής (βασική) μέθοδος για την κατηγοριοποίηση κειμένων, το πρόβλημα της κρίσης εγγράφων ως ανήκουσας σε μία ή την άλλη κατηγορία ως ανεπιθύμητο ή νόμιμο, αθλητικό ή πολιτικό, κλπ.) [42] με συχνότητες λέξεων ως χαρακτηριστικά. Με την κατάλληλη προεπεξεργασία, είναι ανταγωνιστική σε αυτόν τον τομέα με πιο προηγμένες μεθόδους, συμπεριλαμβανομένων μηχανισμών φορέα υποστήριξης. Επίσης βρίσκει εφαρμογή στην αυτόματη ιατρική διάγνωση.

Οι ταξινομητές Naive Bayes είναι εξαιρετικά κλιμακωτοί, απαιτώντας έναν αριθμό γραμμικών παραμέτρων στον αριθμό των μεταβλητών [1] (χαρακτηριστικά / πρόβλεψη) σε ένα μαθησιακό πρόβλημα. Η εκπαίδευση μέγιστης πιθανότητας μπορεί να γίνει με την εκτίμηση μιας έκφρασης κλειστής μορφής που παίρνει γραμμικό χρόνο, παρά με ακριβή επαναληπτική προσέγγιση, όπως χρησιμοποιείται για πολλούς άλλους τύπους ταξινομητών. Στη βιβλιογραφία των στατιστικών και της

επιστήμης των υπολογιστών, τα αφελής μοντέλα Bayes είναι γνωστά κάτω από μια ποικιλία ονομάτων, συμπεριλαμβανομένων των απλών Bayes και Bayes ανεξαρτησία. Όλα αυτά τα ονόματα αναφέρονται στη χρήση του θεωρήματος του Bayes στον κανόνα απόφασης του ταξινομητή, αλλά η το Naive Bayes δεν είναι (απαραιτήτως) Bayesian μέθοδος.

Το Naive Bayes [26] είναι μια απλή τεχνική για την κατασκευή ταξινομητών: μοντέλα που αποδίδουν ετικέτες κλάσης σε στιγμιότυπα προβλημάτων, που αντιπροσωπεύονται ως φορείς των τιμών χαρακτηριστικών, όπου οι ετικέτες κλάσης προέρχονται από κάποιο πεπερασμένο σύνολο. Δεν υπάρχει ένας και μόνος αλγόριθμος για την κατάρτιση τέτοιων ταξινομητών, αλλά μια οικογένεια αλγορίθμων με βάση μια κοινή αρχή: όλοι οι απλοί ταξινομητές Bayes υποθέτουν ότι η αξία ενός συγκεκριμένου στοιχείου είναι ανεξάρτητη από την αξία οποιουδήποτε άλλου χαρακτηριστικού, δεδομένης της μεταβλητής τάξης. Για παράδειγμα, ένας καρπός μπορεί να θεωρηθεί μήλο αν είναι κόκκινο και στρογγυλό. Ένας αφελής ταξινομητής Bayes θεωρεί ότι κάθε ένα από αυτά τα χαρακτηριστικά συμβάλλει ανεξάρτητα στην πιθανότητα ότι αυτός ο καρπός είναι ένα μήλο, ανεξάρτητα από τις πιθανές συσχετίσεις μεταξύ των χρωμάτων, στρογγυλότητας και διαμέτρου.

Για ορισμένους τύπους μοντέλων πιθανοτήτων, οι αφελείς ταξινομητές Bayes μπορούν να εκπαιδευτούν πολύ αποτελεσματικά σε μια εποπτευόμενη μαθησιακή ρύθμιση. Σε πολλές πρακτικές εφαρμογές, η εκτίμηση παραμέτρων για τα αφημένα μοντέλα Bayes χρησιμοποιεί τη μέθοδο της μέγιστης πιθανότητας. με άλλα λόγια, μπορεί κανείς να εργαστεί με το αφελές μοντέλο Bayes χωρίς να δεχτεί Bayesian πιθανότητα ή χρησιμοποιώντας οποιαδήποτε Bayesian μεθόδους.

Παρά το αφελές σχεδιασμό τους και τις φαινομενικά υπεραπλουστευμένες υποθέσεις, οι αφελείς ταξινομητές Bayes εργάστηκαν αρκετά καλά σε πολλές περίπλοκες πραγματικές καταστάσεις. Το 2004, μια ανάλυση του Bayesian ταξινομικού προβλήματος έδειξε ότι υπάρχουν σοβαροί θεωρητικοί λόγοι για την φαινομενικά απίθανη αποτελεσματικότητα των αφελών ταξινομητών Bayes. Ακόμη, μια συνολική σύγκριση με άλλους αλγορίθμους ταξινόμησης το 2006 έδειξε ότι η ταξινόμηση Bayes υπερέχει από άλλες προσεγγίσεις, όπως τα ενισχυμένα δένδρα ή τυχαία δάση. Ένα πλεονέκτημα του αφελούς Bayes είναι ότι απαιτεί μόνο ένα μικρό αριθμό δεδομένων εκπαίδευσης για την εκτίμηση των παραμέτρων που είναι απαραίτητες για την ταξινόμηση.

5.3 Linear SVM

Στη μηχανική μάθηση, τα μηχανήματα υποστήριξης-φορέα (SVM, επίσης δίκτυα υποστήριξης-φορέα) είναι υπό εποπτεία μαθησιακά μοντέλα με συναφείς αλγόριθμους εκμάθησης που αναλύουν δεδομένα που χρησιμοποιούνται για ανάλυση ταξινόμησης και παλινδρόμησης. Δεδομένου ότι ένα σύνολο εκπαιδευτικών παραδειγμάτων, κάθε ένα από τα οποία χαρακτηρίζεται ότι ανήκουν σε μία ή την άλλη από τις δύο κατηγορίες, ένας αλγόριθμος κατάρτισης SVM δημιουργεί ένα μοντέλο που εκχωρεί νέα παραδείγματα σε μία ή την άλλη κατηγορία, καθιστώντας τον έναν μη πιθανοτικό δυαδικό γραμμικό ταξινομητή όπως η Platt κλίμακα [1]υπάρχουν για να χρησιμοποιήσετε SVM σε μια πιθανοτική ρύθμιση ταξινόμησης). Ένα μοντέλο SVM είναι μια αναπαράσταση των παραδειγμάτων ως σημεία στο διάστημα, χαρτογραφημένα έτσι ώστε τα παραδείγματα των ξεχωριστών κατηγοριών χωρίζονται από ένα σαφές κενό που είναι όσο το δυνατόν ευρύτερο. Στη συνέχεια, νέα παραδείγματα χαρτογραφούνται στον ίδιο χώρο και προβλέπεται να ανήκουν σε μια κατηγορία που βασίζεται στην πλευρά του χάσματος επί του οποίου πέφτουν. [42]

Εκτός από την εκτέλεση γραμμικής ταξινόμησης, τα SVM μπορούν να εκτελούν αποτελεσματικά μια μη γραμμική ταξινόμηση χρησιμοποιώντας αυτό που ονομάζεται κόλπο πυρήνα, χαρτογραφώντας σιωπηρά τις εισόδους τους σε χώρους μεγάλης διαστάσεως.

Όταν τα δεδομένα δεν έχουν επισημανθεί, η εποπτευόμενη μάθηση δεν είναι δυνατή και απαιτείται μια μη εποπτευόμενη μαθησιακή προσέγγιση, η οποία προσπαθεί να βρει φυσική ομαδοποίηση των δεδομένων σε ομάδες και στη συνέχεια να χαρτογραφήσει νέα δεδομένα σε αυτές τις σχηματιζόμενες ομάδες. Ο αλγόριθμος συμπλέγματος φορέα υποστήριξης [2], που δημιουργήθηκε από τον Hava Siegelmann και τον Vladimir Vapnik, εφαρμόζει τα στατιστικά στοιχεία των φορέων υποστήριξης που αναπτύσσονται στον αλγόριθμο μηχανισμών φορέα υποστήριξης για την κατηγοριοποίηση μη επισημασμένων δεδομένων και είναι ένας από τους πιο ευρέως χρησιμοποιούμενους αλγόριθμους ομαδοποίησης σε βιομηχανικούς εφαρμογών.

Τα SVM μπορούν να χρησιμοποιηθούν για την επίλυση διαφόρων πραγματικών προβλημάτων:

Τα SVM βοηθούν στην κατηγοριοποίηση κειμένων και υπερκειμένων, καθώς η εφαρμογή τους μπορεί να μειώσει σημαντικά την ανάγκη για επισημάνσεις κατάρτισης σε επισημάνσεις και στις δύο τυπικές επαγωγικές και [27] μεταγωγικές ρυθμίσεις. [26]Μερικές μέθοδοι για ρηχή σημασιολογική ανάλυση βασίζονται σε μηχανές φορέα υποστήριξης.

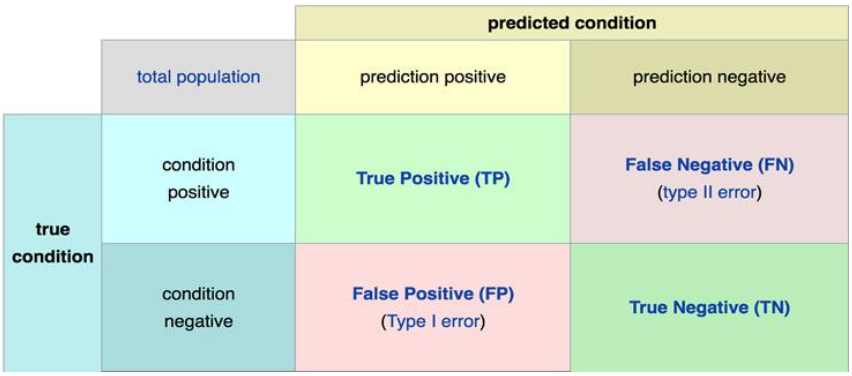
Η ταξινόμηση των εικόνων μπορεί επίσης να γίνει χρησιμοποιώντας SVM. Τα πειραματικά αποτελέσματα δείχνουν ότι τα SVM επιτυγχάνουν σημαντικά υψηλότερη ακρίβεια αναζήτησης από τα παραδοσιακά συστήματα βελτίωσης επερωτήσεων μετά από μόλις τρεις έως τέσσερις κύκλους ανατροφοδότησης. Αυτό ισχύει και για τα συστήματα κατακερματισμού εικόνας, συμπεριλαμβανομένων εκείνων που χρησιμοποιούν μια τροποποιημένη έκδοση SVM που χρησιμοποιεί την προνομιακή προσέγγιση όπως προτείνεται από τον Vapnik
Οι χειρόγραφες χαρακτήρες μπορούν να αναγνωριστούν χρησιμοποιώντας το SVM. Ο αλγόριθμος SVM έχει εφαρμοστεί ευρέως στις βιολογικές και άλλες επιστήμες. Έχουν χρησιμοποιηθεί για την ταξινόμηση πρωτεϊνών με έως και 90% των ενώσεων που έχουν ταξινομηθεί σωστά. Οι δοκιμασίες μετασχηματισμού με βάση τα βάρη SVM έχουν προταθεί ως ένας μηχανισμός για την ερμηνεία των μοντέλων SVM . Τα βάρη μηχανής υποστήριξης-φορέα έχουν επίσης χρησιμοποιηθεί για την ερμηνεία των μοντέλων SVM στο παρελθόν. Η ερμηνεία του μεταφορέα των μοντέλων μηχανημάτων υποστήριξης-φορέα προκειμένου να προσδιοριστούν τα χαρακτηριστικά που χρησιμοποιεί το μοντέλο για την πρόβλεψη είναι ένας σχετικά νέος τομέας έρευνας με ιδιαίτερη σημασία στις βιολογικές επιστήμες. [40]

ΚΕΦΑΛΑΙΟ 6: Μέθοδοι Εκτίμησης Κατηγοριοποιητών

Σε περιπτώσεις όπου οι κλάσεις δεν είναι ισομερώς κατανεμημένες ή όπου οι εσφαλμένες κατηγοριοποιήσεις διαφορετικών κλάσεων έχουν διαφορετικό κόστος, είναι σημαντική η εκτίμηση της ικανότητας πρόβλεψης του κατηγοριοποιητή για την κάθε κλάση. Για να εκτιμήσουμε τις ανά κλάση επιδόσεις ενός κατηγοριοποιητή, εισάγουμε την αναγκαία ορολογία. Για την περίπτωση δυαδικής κλάσης ισχύουν οι ακόλουθοι όροι: Θετικές παρατηρήσεις (positive) ονομάζονται οι παρατηρήσεις, οι οποίες ανήκουν σε μια τιμή της κλάσης (π.χ. χρεοκοπία)

1. Αρνητικές παρατηρήσεις (negative) ονομάζονται οι παρατηρήσεις, οι οποίες ανήκουν στην άλλη τιμή της κλάσης (π.χ. μη χρεοκοπία)
2. Αληθινές Θετικές Προβλέψεις (True Positive – tp) είναι το πλήθος των επιτυχών προβλέψεων των θετικών παρατηρήσεων (π.χ. η επιχείρηση είναι χρεοκοπημένη και ο κατηγοριοποιητής προβλέπει σωστά την κλάση).
3. Αληθινές Αρνητικές Προβλέψεις (True Negative – tn) είναι το πλήθος των επιτυχημένων προβλέψεων των αρνητικών παρατηρήσεων (π.χ. η επιχείρηση δεν είναι χρεοκοπημένη και ο κατηγοριοποιητής προβλέπει σωστά την κλάση).
4. Ψευδείς Θετικές Προβλέψεις (False Positive – fp) είναι το πλήθος των προβλέψεων των αρνητικών παρατηρήσεων που κατηγοριοποιήθηκαν λανθασμένα στις θετικές παρατηρήσεις (η επιχείρηση δεν είναι χρεοκοπημένη, ο κατηγοριοποιητής όμως την προβλέπει ως χρεοκοπημένη).
5. Ψευδείς Αρνητικές Προβλέψεις (False Negative – fn) είναι το πλήθος προβλέψεων των θετικών παρατηρήσεων που κατηγοριοποιήθηκαν λανθασμένα στις αρνητικές προβλέψεις (η επιχείρηση είναι χρεοκοπημένη, ο κατηγοριοποιητής όμως την προβλέπει ως μη χρεοκοπημένη).

Ένας τρόπος παρουσίασης των επιδόσεων ανά κλάση ενός κατηγοριοποιητή είναι με τη χρήση του πίνακα σύγχυσης (Confusion Matrix). Ο Πίνακας Σύγχυσης είναι ένας δισδιάστατος πίνακας, όπου οι στήλες αντιστοιχούν στις προβλέψεις και οι γραμμές στις πραγματικές τιμές κλάσης. Στα κελιά του πίνακα αναγράφονται οι αληθινές θετικές, οι αληθινές αρνητικές, οι ψευδείς θετικές και οι ψευδείς αρνητικές προβλέψεις. Στο Σχήμα απεικονίζεται ένας Πίνακας Σύγχυσης.



Εικόνα 2 Πίνακας Σύγχυσης

Ορισμένα πρόσθετα μέτρα για τις επιδόσεις ενός κατηγοριοποιητή είναι τα ακόλουθα:

$$sensitivity = \frac{tp}{pos}$$

$$specificity = \frac{tn}{negat}$$

$$precision = \frac{tp}{tp + fp}$$

$$accuracy = sensitivity * \frac{pos}{pos + negat} + specificity * \frac{negat}{pos + negat} = \frac{tp + tn}{pos + negat}$$

όπου *pos* είναι το πλήθος των θετικών παρατηρήσεων και *negat* είναι το πλήθος των αρνητικών παρατηρήσεων. Σύμφωνα με τα παραπάνω, η ακρίβεια (accuracy) ορίζεται ως το ποσοστό των ορθών θετικών προβλέψεων επί το ποσοστό των θετικών παρατηρήσεων συν το ποσοστό των ορθών αρνητικών προβλέψεων επί το ποσοστό των αρνητικών παρατηρήσεων ή ισοδύναμα ως το πλήθος των ορθών προβλέψεων προς το πλήθος των παρατηρήσεων.

6.1 Καμπύλες ROC

Ένα ισχυρό μέτρο για την εκτίμηση της ανά κλάση ακρίβειας του κατηγοριοποιητή είναι οι λεγόμενες καμπύλες ROC (Receiver Operating Characteristics). Οι καμπύλες ROC σχεδιάζονται σε έναν δισδιάστατο επίπεδο χώρο. Ο οριζόντιος άξονας εκφράζει το μέγεθος $1 - specificity$, το οποίο ονομάζεται και False Positive Rate.

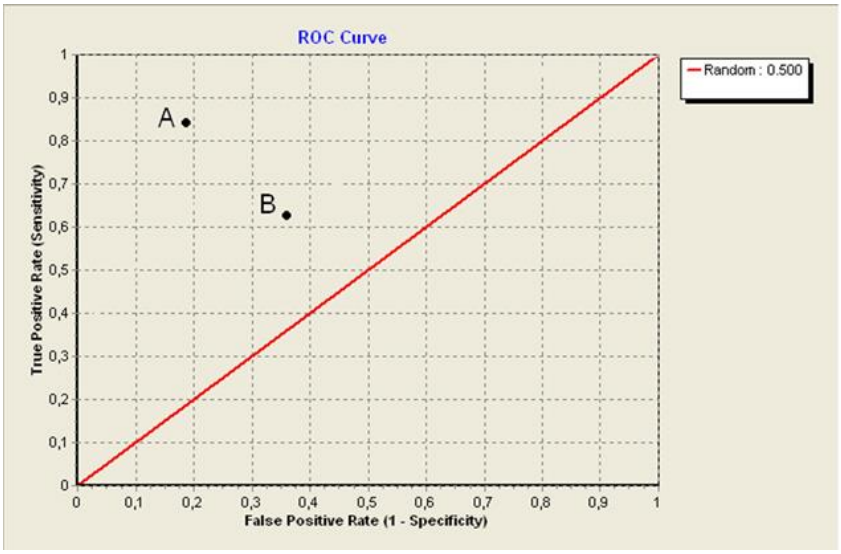
$$False\ Positive\ Rate = 1 - specificity = \frac{fp}{negat}$$

Ο κατακόρυφος άξονας εκφράζει το μέγεθος sensitivity, το οποίο ονομάζεται και True Positive Rate

$$True\ Positive\ Rate = sensitivity = \frac{tp}{pos}$$

Ουσιαστικά, ο οριζόντιος άξονας εκφράζει το ποσοστό των αρνητικών παρατηρήσεων, οι οποίες κατηγοριοποιήθηκαν λάθος, και ο κατακόρυφος άξονας εκφράζει το ποσοστό των θετικών παρατηρήσεων, οι οποίες κατηγοριοποιήθηκαν σωστά. Το Σχήμα 10.8 απεικονίζει τον δισδιάστατο χώρο καμπύλων ROC. Κάθε σημείο του χώρου αυτού εκφράζει ένα ισοζύγιο ανάμεσα στο ποσοστό ορθών θετικών προβλέψεων και εσφαλμένων θετικών προβλέψεων. Το σημείο 0,0 είναι ένας κατηγοριοποιητής, που δεν προβλέπει ποτέ θετική παρατήρηση. Το σημείο 1,1 είναι ένας κατηγοριοποιητής, που προβλέπει πάντα θετική παρατήρηση. Η διαγώνια γραμμή, από το σημείο 0,0 στο σημείο 1,1 είναι ένας κατηγοριοποιητής που προβλέπει τυχαία την κλάση. Οι κατηγοριοποιητές που βρίσκονται κάτω από τη διαγώνια γραμμή είναι χειρότεροι από την τυχαία πρόβλεψη. Οι

κατηγοριοποιητές που βρίσκονται πάνω από τη διαγώνια γραμμή είναι καλύτεροι από την τυχαία πρόβλεψη. Το σημείο 0,1 είναι ο άριστος κατηγοριοποιητής, οι οποίος προβλέπει σωστά όλες τις θετικές και αρνητικές παρατηρήσεις. Γενικώς, όσο πιο μετατοπισμένο είναι προς τα πάνω και προς τα αριστερά ένα σημείο, τόσο καλύτερη θεωρείται η επίδοση. Στο Σχήμα το σημείο Α είναι καλύτερο από το σημείο Β.



Εικόνα 3 Καμπύλη ROC

6.2 Σημεία στον χώρο καμπύλων ROC

Η επίδοση των κατηγοριοποιητών στον χώρο ROC συμβολίζεται με μία καμπύλη. Για να συγκρίνουμε κατηγοριοποιητές χρειαζόμαστε ένα μέτρο σύγκρισης. Τέτοιο μέτρο σύγκρισης είναι η Περιοχή Κάτω από την Καμπύλη ROC (Area Under ROC Curve (AUC)). Η AUC εκφράζει το ποσοστό του χώρου που βρίσκεται κάτω από την καμπύλη, και παίρνει τιμές από 0 έως 1. Η διαγώνια γραμμή τυχαίας πρόβλεψης έχει AUC = 0,5. Συνεπώς, κάθε κατηγοριοποιητής καλύτερος της τυχαίας πρόβλεψης έχει AUC > 0,5. Όσο μεγαλύτερη περιοχή AUC έχει ένας κατηγοριοποιητής τόσο καλύτερος είναι. Στο Σχήμα 10.9, παρουσιάζονται οι καμπύλες ROC ενός Δέντρου Αποφάσεων και ενός Νευρωνικού Δικτύου, τα οποία προβλέπουν περιπτώσεις, όπου οι εξωτερικοί ελεγκτές εκδίδουν δυσμενή σχόλια. Όπως φαίνεται στο σχήμα, η τιμή AUC του Νευρωνικού Δικτύου είναι μεγαλύτερη από την αντίστοιχη του Δέντρου Αποφάσεων, γεγονός που σημαίνει ότι το μοντέλο του Νευρωνικού Δικτύου προβλέπει πιο αποτελεσματικά τις περιπτώσεις έκδοσης δυσμενών σχολίων.

6.3 Metrics του Spark

6.3.1 Dependability

Στην αρχιτεκτονική συστημάτων, το dependability αποτελεί μέτρο της διαθεσιμότητας του συστήματος (availability), της αξιοπιστίας του (reliability) και της διατηρησιμοτητας του (maintainability). [17] Στην αρχιτεκτονική συστημάτων, dependability είναι η δυνατότητα παροχής services που μπορούν να υποστηριχθούν αξιόπιστα χωρίς να πέφτουν σε μια ορισμένη χρονική περίοδο. Αυτό μπορεί επίσης να περιλαμβάνει μηχανισμούς σχεδιασμένους να αυξάνουν και να διατηρούν το χρόνο υποστήριξης ενός συστήματος ή λογισμικού. [11]

6.3.2 Availability

Στη θεωρία του reliability και στο reliability engineering, ο όρος availability έχει τις ακόλουθες έννοιες:Ο βαθμός στον οποίο ένα σύστημα, υποσύστημα ή εξοπλισμός βρίσκεται σε μια συγκεκριμένη λειτουργική και δεσμευτική [4] κατάσταση κατά την έναρξη μιας αποστολής, όταν η αποστολή καλείται σε έναν άγνωστο, δηλαδή έναν τυχαίο χρόνο. Με απλά λόγια, η διαθεσιμότητα είναι το ποσοστό του χρόνου που ένα σύστημα βρίσκεται σε λειτουργική κατάσταση. Αυτό συχνά περιγράφεται ως ποσοστό ικανότητας αποστολής. Μαθηματικά, αυτό εκφράζεται ως 100% μείον μη διαθεσιμότητα.

6.3.3 Reliability

Reliability ενός συστήματος είναι η πιθανότητα ότι το σύστημα εκτελεί μια καθορισμένη λειτουργία κάτω από τις προδιαγραφές των λειτουργικών συνθηκών καθώς και των συνθηκών περιβάλλοντος [17]κατά τη διάρκεια και σε ολόκληρη την καθορισμένη χρονική περίοδο.
Ποσοτικά, η αξιοπιστία ορίζεται ως η πιθανότητα επιτυχίας (Reliability=1-Probability of Failure). Συνήθως εκφράζεται ως το Mean Time Between Failures (MTBF). [4]

6.3.4 Maintainability

Η ικανότητα ενός συστήματος, υπό καθορισμένες συνθήκες χρήσης, να διατηρηθεί ή να αποκατασταθεί σε μια κατάσταση στην οποία μπορεί εκτελεί τις απαιτούμενες λειτουργίες, όταν εκτελείται η συντήρηση [18] υπό καθορισμένους όρους και με τη χρήση καθορισμένων διαδικασιών και πόρων. Εκφράζεται ως Mean Time To Repair (MTTR). [4]

6.3.5 Fault Tolerance

Fault tolerance (ανοχή σφάλματος) είναι η ιδιότητα που επιτρέπει σε ένα σύστημα να συνεχίσει να λειτουργεί σωστά σε περίπτωση βλάβης (ή ενός ή περισσότερων σφαλμάτων εντός) ορισμένων από τα εξαρτήματά του. Εάν η ποιότητα λειτουργίας του μειωθεί, η μείωση είναι ανάλογη με τη σοβαρότητα της βλάβης, σε σύγκριση με ένα αφηρημένα σχεδιασμένο σύστημα, στο οποίο ακόμη και μια μικρή βλάβη μπορεί να προκαλέσει συνολική κατάρρευση. Η ανοχή σφάλματος επιδιώκεται ιδιαιτέρως από τα συστήματα υψηλής αξιοπιστίας ή life-critical συστήματα. Η ικανότητα διατήρησης της λειτουργικότητας όταν τμήματα ενός συστήματος σταματούν να λειτουργούν, αναφέρονται ως graceful degradation. [9]

Το Spark έχει [18]σχεδιαστεί για να υποστηρίζει την απώλεια οποιουδήποτε συνόλου Worker nodes. Θα επαναλάβει όλα τα tasks που εκτελέστηκαν από αυτούς τους κόμβους και θα ανασυνθέσει τα δεδομένα που είχαν αποθηκευτεί σε αυτά. Επιπλέον το Spark έχει σχεδιαστεί για να λειτουργεί ακόμα και στις περιπτώσεις που τα δεδομένα δεν χωράνε στη μνήμη, οπότε μπορεί να συνεχίσει την επεξεργασία στο σκληρό δίσκο.

Ένα πλεονέκτημα του Spark, είναι ότι είναι ένα software βασισμένο σε μικρό σύνολο πηγαίου κώδικα (περίπου 17K γραμμές κώδικα), το οποίο βοηθά στο να διατηρείται σταθερό και επίσης διευκολύνει το debug. [9]

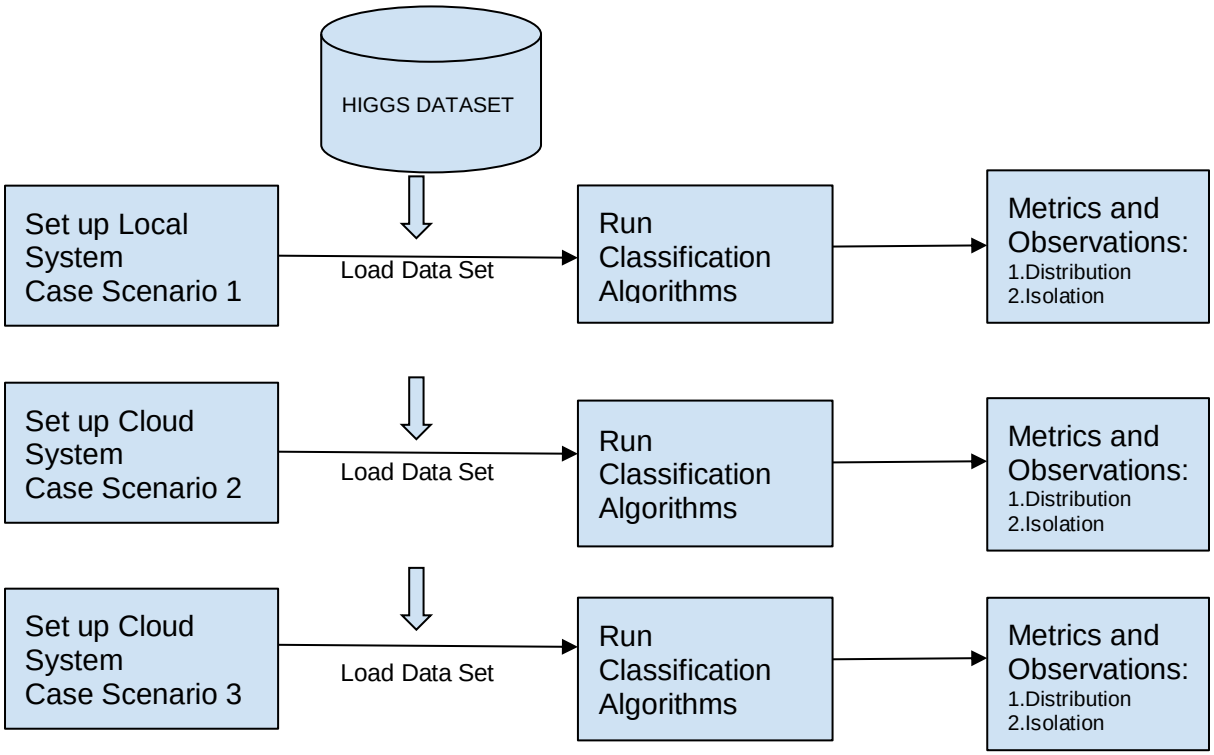
6.3.6 Distribution

Το distribution metric, χρησιμοποιείται για να δείξει τον τρόπο κατανομής των εργασιών του Spark αναμεσα στα VM, στα Container και στους Executors που αναλαμβάνουν να τρέξουν τα συγκεκριμένα tasks. Το Spark λειτουργεί με τρόπο που να γίνεται όσο το δυνατόν καλύτερη κατανομή πόρων και υπολογιστικής ισχύς μεταξύ των component που το απαρτίζουν [25].

ΚΕΦΑΛΑΙΟ 7: Case Scenarios

7.1 Εισαγωγή

Αφού αναλύθηκε σε βάθος η θεωρία για τον τρόπο και τις απαιτήσεις που έχει ένα distributed environment για την εφαρμογή machine learning τεχνικών, εφαρμόστηκαν 3 case scenarios. Σε κάθε ένα από αυτά τα σενάρια χρησιμοποιήθηκε διαφορετική αρχιτεκτονική για το Hadoop και το Spark, καθώς και διαφορετικό infrastructure. Το dataset που χρησιμοποιήθηκε είναι του μποζονίου Higgs στο οποίο εφαρμόστηκαν οι συγκεκριμένοι αλγοριθμοι, δημιουργώντας ένα μοντέλο κατηγοριοποίησης με στόχο να προβλέψει με βάση τα features των δεδομένων το αν κάποιο σωματίδιο μπορεί να κατηγοριοποιηθεί στα συγκεκριμένο μποζόνιο.



WORK FLOW CASE SCENARIOS

Στο από πάνω διάγραμμα παρουσιάζεται ο τρόπος υλοποίησης των τριών σεναρίων. Η αναλύτικότερη περιγραφή των σταδίων υλοποίησης των κατηγοριοποιήσεων παρουσιάζεται αναλύτικότερα στις επόμενες υποενότητες.

7.2 DataSet

Το dataset του μποζονίου Higgs, είναι αναρτημένο στην ιστοσελίδα «UCI Machine Learning Repository». Είναι αναρτημένο από το 2014 και ο όγκος των δεδομένων που περιέχει για ανάλυση κρίθηκε κατάλληλος για επεξεργασία με Big Data συστήματα. Τα δεδομένα έχουν παραχθεί χρησιμοποιώντας προσομοιώσεις Monte Carlo. Τα πρώτα 21 χαρακτηριστικά (στήλες 2-22) φανερώνουν κινηματικές ιδιότητες των σωματιδίων, που μετριούνται από τους ανιχνευτές σωματιδίων στον επιταχυντή. Τα τελευταία 7 feature, είναι βασισμένα στις λειτουργίες των 21 πρώτων χαρακτηριστικών. Τα συγκεκριμένα, είναι χαρακτηριστικά υψηλού επιπέδου που παράγονται από ερευνητές του κλάδου της φυσικής επιστήμης, για να βοηθήσουν να γίνουν διακρίσεις μεταξύ των δύο κλάσεων. Υπάρχει ενδιαφέρον για τη χρήση μεθόδων deep learning για να αποφευχθεί η ανάγκη για να αναπτυχθούν χειροκίνητα τέτοια χαρακτηριστικά.

Αφού έγινε download του αρχείο του μποζονίου Higgs τοπικά, για να επιτευχθεί η παράλληλη επεξεργασία έπρεπε να γίνει upload στο οικοσύστημα του Hadoop. Παρακάτω περιγράφονται τα βήματα μεταφόρτωσης του αρχείου.

Βήματα του Uploading στο Hadoop HDFS:

1. **Set Up Hadoop-Spark**

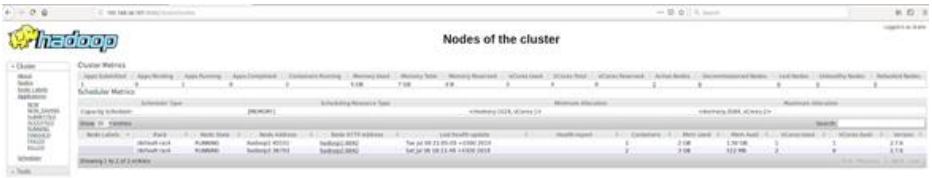
Αρχικά υλοποιήθηκε το σετάρισμα της κάθε αρχιτεκτονικής (locally και cloud). Το όνομα του χρήστη που δόθηκε και στα δύο VM έτσι ώστε να επιτευχθεί η επικοινωνία τους είναι ο “hduser”.

2. **Start Hadoop Services**

Αφού εισέλθουμε στο VM με το χρήστη hduser, επόμενο βήμα είναι να εκκινήσουμε τα services του Hadoop.

```
$ start-dfs.sh
$ start-yarn.sh
```

Με την πρώτη εντολή εκινούνται τα HDFS daemons του Hadoop. Έτσι μπορεί να γίνει πλοήγηση μέσα από την πόρτα του localhost: 8088.



Με τη δεύτερη εντολή εκινείται το yarn που είναι ο resource manager που θα χρησιμοποιήσουμε.

3. Upload Dataset

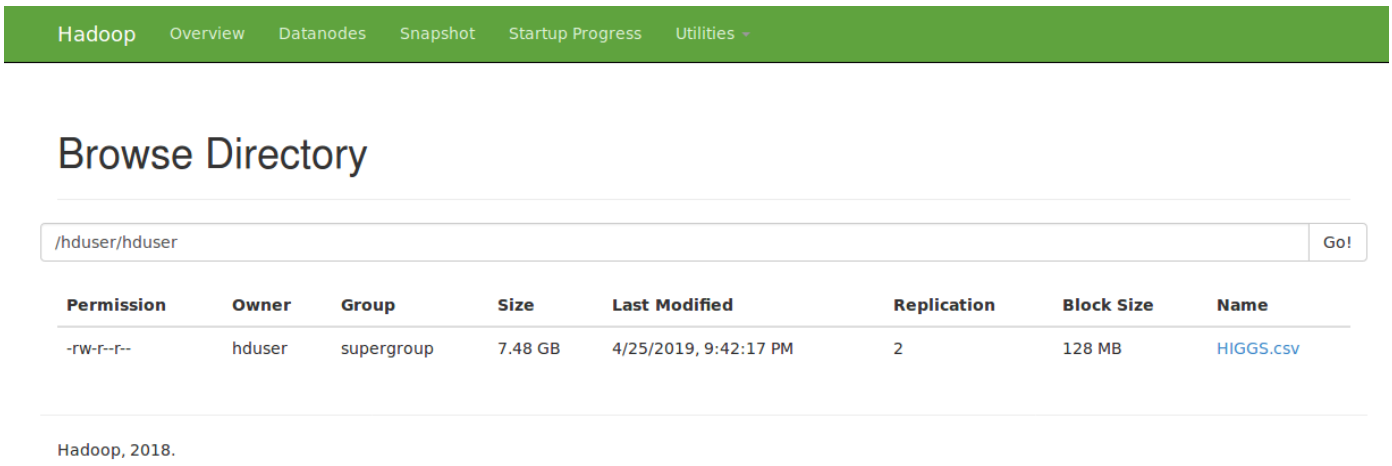
Στη συνέχεια το dataset έγινε upload στο Hadoop-HDFS με τις εντολές που παρατίθενται από κάτω.

```
$ hadoop fs -mkdir /hduser/hduser

$ hadoop fs -put /home/usr/HIGGS.csv /hduser/hduser/HIGGS.csv
```

Upload Data Set στο Hadoop

Αρχικά δημιουργήθηκε ένας φάκελος "hduser" μέσα στον χρήστη με το ίδιο όνομα έτσι ώστε να αποθηκεύσουμε εκεί το αρχείο. Στη συνέχεια, με την δεύτερη εντολή έγινε upload το dataset από τον τοπικό δίσκο στο file system του Hadoop. Για να εξασφαληστεί ότι όλα πήγαν καλά το Hadoop δίνει τη δυνατότητα οπτικοποίησης των περιεχομένων του HDFS μέσα από την πόρτα του localhost: 50070.



Store Dataset του HDFS του Hadoop

Αφού πλοηγηθούμε στο HDFS δίνεται η δυνατότητα για περιήγηση μέσα στους φακέλους του. Επιλέγοντας το χρήστη και το μετά το φάκελο που δημιουργήθηκε με το ίδιο όνομα παρατειρείται ότι το dataset έχει εισαχθεί στο file system. Τα αρχεία μέσα στο HDFS είναι read only και δεν μπορούν να γίνουν modified.

Οι πληροφορίες που γίνονται διαθέσιμες είναι οι εξής:

- Owner: Ο ιδιοκτήτης του αρχείου είναι ο χρήστης
- Size: Το μέγεθος του αρχείου
- Last Modified: Είναι το πότε εισήχθη στο σύστημα το αρχείο
- Replication: Σε πόσα VM έχει αντιγραφεί το αρχείο
- Block Size: Το μέγεθος των partition στο οποίο έχει χωριστεί το dataset

7.3 Αλγόριθμοι Κατηγοριοποίησης :

Για να γίνουν διακριτές οι διαφορές στον τρόπο που συμπεριφέρονται τα big data συστήματα, χρησιμοποιήθηκαν 3εις αλγόριθμοι κατηγοριοποίησης (όπως περιγράφονται στο Κεφάλαιο 5) που ο καθένας τους έχει διαφορετικές προδιαγραφές, καθώς και τρόπο επεξεργασίας.

Τα βήματα των αλγορίθμων για την επεξεργασία του dataset είναι τα παρακάτω:

I. Cast features as DoubleTypes
II. Διαχωρισμός των label from features
III. Vectorize features using VectorAssembler
IV. Διαχωρισμός του dataset σε train και test
V. Εκτέλεση αλγορίθμου
VI. Εκτύπωση αποτελεσμάτων
VII. Εκτέλεση καμπύλης ROC
VIII. Εκτύπωση αποτελέσματος καμπύλης ROC

7.4 Μετρησεις του Spark:

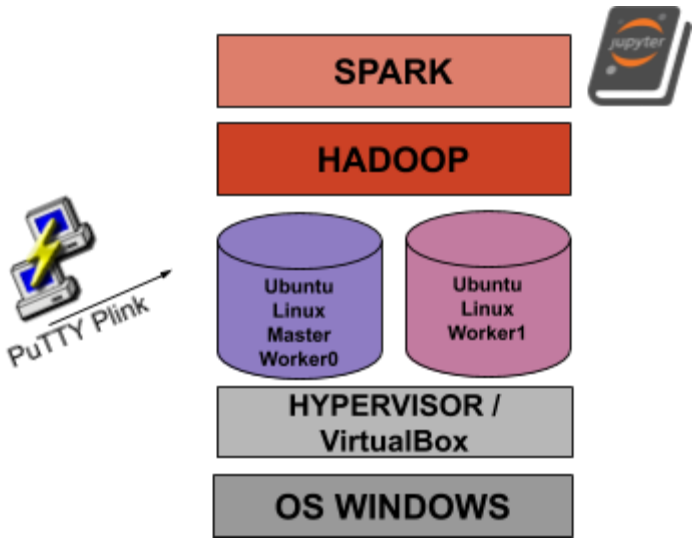
Οι μετρικές που ελήφθησαν υπ όψιν ως μέτρα σύγκρισης μεταξύ των αρχιτεκτονικών είναι οι εξής:

- ο χρόνος εκτέλεσης κάθε αλγορίθμου,
- η ακρίβεια των αποτελεσμάτων του,
- η αξιοπιστία των συστημάτων όταν εκτελούνταν ο κάθε αλγόριθμος,
- η κατανομή των πόρων στους επεξεργαστές για την επεξεργασία των δεδομένων
- καθώς και ο τρόπος που επιτεύχθηκε αυτή η επεξεργασία

Παρακάτω παρατίθενται όλες οι πληροφορίες για την αρχιτεκτονική, για τα χαρακτηριστικά των συστημάτων, καθώς και για τα αποτελέσματα των μετρικών.

7.5.1 Case Scenario 1ο:

Η αρχιτεκτονική που χρησιμοποιήθηκε είναι η εξής:



Αρχιτεκτονική Case Scenario 1

Το infrastructure οικοδομήθηκε πάνω σε ένα τοπικό μηχάνημα. Για να δοθεί η δυνατότητα δημιουργίας Virtual Machines, εγκαταστάθηκε ένας Hypervisor type 2. Στη συγκεκριμένη περίπτωση ο hypervisor ήταν το VirtualBox της Oracle. Πάνω σε αυτόν εγκαταστάθηκαν 2 VMs με λειτουγικό Linux, τα οποία χωρίστηκαν σε Master-Worker στο ένα και Worker στο άλλο. Στη συνέχεια εγκαταστάθηκαν κάποιες βιβλιοθήκες βελτιώσης της λειτουργίας των VM.

Αφού στήθηκε ολόκληρη η υποδομή του virtualization, επόμενο βήμα ήταν η εγκατάσταση του Hadoop ως file system και του Spark, το οποίο έγινε χειροκίνητα. Στη συνέχεια εγκατασταθηκε η python ως γλώσσα προγραμματισμού για την ανάπτυξη των αλγορίθμων. Τέλος εγκαταστάθηκε το IDE περιβάλλον: Jupyter Notebook IDE το οποίο δίνει τη δυνατότητα λόγω του ότι έχει oline interface ανοίγοντας σε πόρτα του localhost, να μπορείς να το χρησιμοποιήσεις εκτός των vm. Χρησιμοποιόντας τον kernel της python3 αναπτύχθηκαν οι αλγόριθμοι που αναφέρθηκαν στην αρχή του κεφαλαίου και στη συνέχεια συλλέχθηκαν τα αποτελέσματα που εξυπηρετούν το σκοπό της εργασίας.

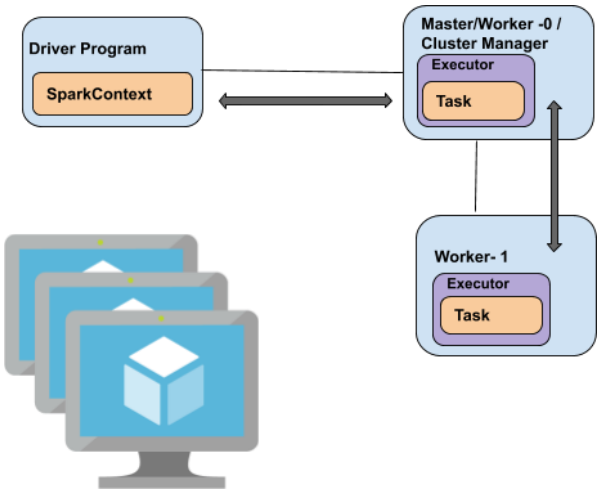
Παρακάτω αναφέρονται στον πίνακα τα χαρακτηριστικά των VM.

Τα χαρακτηριστικά του συστήματος είναι τα εξής:

Scenario 1	**Master node / Worker node -0**	**Worker node -1**
Machine type	Oracle's VirtualBox VM	Oracle's VirtualBox VM
CPU	2 vCPU	2 vCPU
RAM	4 GB memory	4 GB memory
Primary disk type	pd-ssd	pd-ssd
Primary disk size	40 GB	40 GB
OS	Ubuntu 16.04	Ubuntu 16.04
Optional components	JUPYTER, ANACONDA	JUPYTER, ANACONDA

Πίνακας Υπολογιστικών Πόρων των Vms

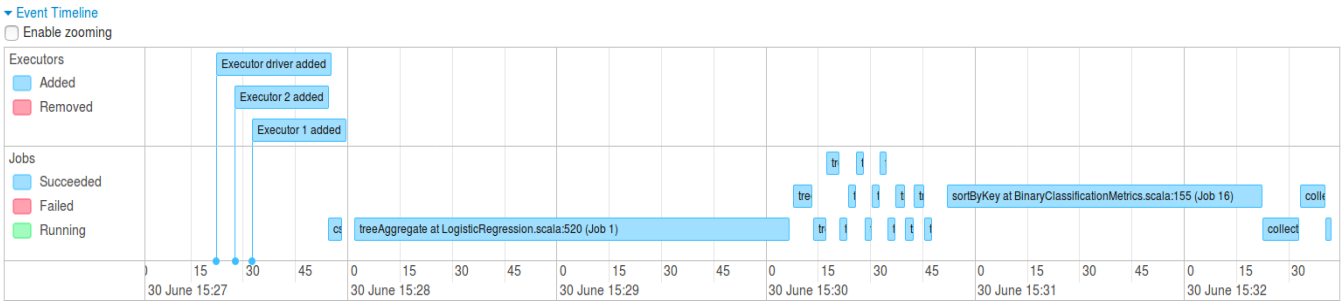
Σε αυτό το σενάριο, το cluster των υπολογιστών αποτελείται από 2 εικονικές μηχανές. Στην πρώτη περιλαμβάνεται ο Master Node μαζί με τον Worker Node-0 και στη δεύτερη ο Worker Node-1. Σε αυτή την αρχιτεκτονική, συνυπάρχουν σε ένα VM ο Resource Manager, που ορίζει τις διεργασίες, με τον Worker, στον οποίο γίνεται κομμάτι της επεξεργασίας. Τέλος, και στους δύο εικονικούς υπολογιστές παρέχονται τα ίδια resources για την εκτέλεση των αλγορίθμων.



Internal Spark architecture CASE SCENARIO 1

Δυστυχώς η εγκατάσταση του Master node μαζί με τον Worker node δεν είναι η βέλτιστη αρχιτεκτονική, αλλά επιλέχθηκε με αυτόν τον τρόπο λόγο των περιορισμένων resources που μπορούσαν να διατεθούν.

1st Case Study Algorithm: Αποτελέσματα Λογιστικής Παλινδρόμησης



Spark UI, Timeline Process: Logistic Regression CS1

Στο παραπάνω παράρτημα απεικονίζεται το Timeline της εκτέλεσης της λογιστικής παλινδρόμησης. Στο πάνω μέρος της εικόνας απεικονίζονται οι Executors. Αρχικά, προστίθεται ο Executor driver που ορίζει τον τρόπο που θα κατανεμηθούν οι δουλειές στους υπόλοιπους executors. Στη συνέχεια προστίθεται ο Executor 2 και τέλος ο Executor 1 για να εκτελέσουν τις διεργασίες. Στο κάτω μέρος που απεικονίζονται τα βήματα που ακολουθεί ο αλγόριθμος ως Jobs και η διαδρομή τους μέχρι να τυπωθεί το τελικό αποτέλεσμα.

Ο συνολικός αριθμός των jobs που λανβάνουν χώρα για την εκτέλεση του αλγορίμθου της λογιστική παλινδρόμησης είναι 20. Παρατηρείται ότι τα μεγαλύτερα σε έκταση άρα και σε χρόνο jobs είναι τα “treeAggregate at Logistic Regression.scala” και “sortByKey at BinaryClassificationMetric.scala”.

Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
Active(3)	0	1.4 MB / 1.2 GB	905.3 KB	4	0	0	1369	1369	18 min (1.8 min)	26.5 GB	157 MB	78.6 MB
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
Total(3)	0	1.4 MB / 1.2 GB	905.3 KB	4	0	0	1369	1369	18 min (1.8 min)	26.5 GB	157 MB	78.6 MB

Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs
driver	hadoop1:44119	Active	0	868.6 KB / 384.1 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	
1	hadoop2:46083	Active	0	166 KB / 384.1 MB	522.4 KB	2	0	0	729	729	8.9 min (55 s)	13.7 GB	81.2 MB	39.3 MB	stdout stderr
2	hadoop1:43475	Active	0	335.2 KB / 384.1 MB	382.9 KB	2	0	0	640	640	9.0 min (53 s)	12.9 GB	75.7 MB	39.3 MB	stdout stderr

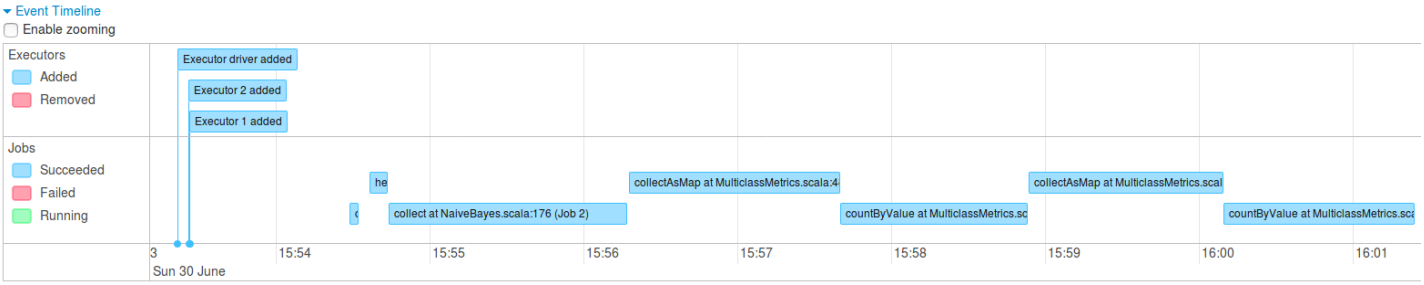
SPARK UI, EXECUTORS: Logistic Regression CS1

Στη σελίδα των executors που περιέχεται στο UI του Spark εμφανίζεται όπως δείχνει και η από πάνω εικόνα το "Summary" και τα στοιχεία των "Executors". Στο summary εμφανίζονται τα συνολικά αποτελέσματα της επαιξεργασίας των executors, ενώ στο executors εμφανίζεται το πώς έγινε η κατανομή των πόρων, των tasks και της επαιξεργασίας σε κάθε έναν executor.

Παρατηρήσεις:

1. Event Timeline: Η επεξεργασία του κάθε job ορίζεται από τον executor driver να γίνεται παράλληλα στους executors
2. Ο maximum αριθμός των task για κάθε job της λογιστικής παλινδρόμησης είναι 67, ενώ για την παραγωγή των μετρικών φτάνει τα 120
3. Παρατηρούμε ότι καθ' όλη τη διάρκεια του process και οι τρεις executors είναι active.
4. Executors: Το σύνολο των ολοκληρωμένων Tasks (1369) μοιράζονται 53% στον πρώτο Executor και 47% στον δεύτερο, μια επιπλέον παρατήρηση είναι ότι δεν υπάρχουν Failed Tasks. Το Shuffle Read ο Exec1 αλλά και Το Shuffle Write.
5. Η λογιστική παλινδρόμηση φαίνεται να χρησιμοποιεί μέρος του σκληρού δίσκου του τοπικού Server.
6. Το Garbage Collection εμφανίζεται με κόκκινο χρώμα από το ίδιο το UI, υποδηλώνοντας ότι υπήρξε κάποιο πρόβλημα στη διαδικασία.

2nd Case Study Algorithm: Αποτελέσματα Naive Bayes



Spark UI, Timeline Process: Naïve Bayes CS1

Στο παραπάνω παράρτημα απεικονίζεται το Timeline της εκτέλεσης του **Naive Bayes** . Στο πάνω μέρος της εικόνας απεικονίζονται οι Executors. Αρχικά, προστίθεται ο Executor driver που ορίζει τον τρόπο που θα κατανεμηθούν οι δουλειές στους υπόλοιπους executors. Στη συνέχεια προστίθεται ο Executor 2 και τέλος ο Executor 1 για να εκτελέσουν τις διεργασίες. Στο κάτω μέρος απεικονίζονται τα βήματα που ακολουθεί ο αλγόριθμος ως Jobs και η διαδρομή τους μέχρι να τυπωθεί το τελικό αποτέλεσμα.

Ο συνολικός αριθμός των jobs που λανβάνουν χώρα για την εκτέλεση του αλγορίμθου του Naïve Bayes είναι 6. Ο αριθμός των jobs είναι κατά 70% μικρότερος από την Λ.Π. Παρατηρείται ότι σε έκταση και σε χρόνο τα jobs είναι πανομοιότυπα. Παρ' όλ' αυτά ο χρονος εκτέλεσης του αλγορίθμου είναι μεγαλύτερος.

Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
Active(3)	0	802.4 KB / 1.2 GB	0.0 B	4	0	0	602	602	26 min (33 s)	40.3 GB	41.3 KB	41.3 KB
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
Total(3)	0	802.4 KB / 1.2 GB	0.0 B	4	0	0	602	602	26 min (33 s)	40.3 GB	41.3 KB	41.3 KB

Executors

Show 20 entries Search:

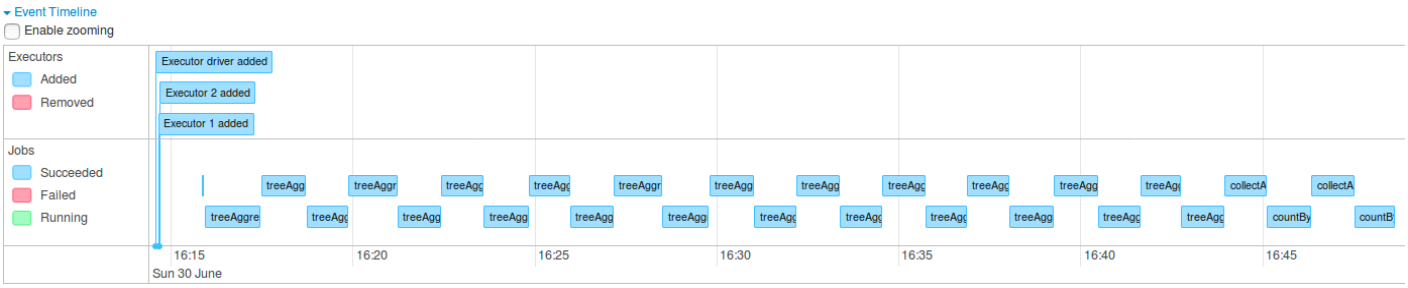
Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs
driver	hadoop1:40547	Active	0	324.9 KB / 384.1 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	
1	hadoop1:40707	Active	0	202.8 KB / 384.1 MB	0.0 B	2	0	0	258	258	13 min (18 s)	20.4 GB	20.5 KB	20.6 KB	stdout stderr
2	hadoop2:34673	Active	0	274.7 KB / 384.1 MB	0.0 B	2	0	0	344	344	13 min (15 s)	20 GB	20.8 KB	20.7 KB	stdout stderr

SPARK UI, EXECUTORS: Naïve Bayes CS2

Παρατηρήσεις:

1. Event Timeline: Η επεξεργασία του κάθε job ορίζεται από τον executor driver να γίνεται παράλληλα στους executors
2. Ο maximum αριθμός των task για κάθε job του Naive Bayes είναι 120, ενώ για την παραγωγή των μετρικών φτάνει τα 120
3. Παρατηρούμε ότι καθ' όλη τη διάρκεια του process και οι τρεις executors είναι active.
4. Τα Jobs που εκτελούνται είναι λιγότερα από την λογιστική παλινδρόμηση κατα το ⅓
5. Τα jobs (όπως επίσης φανερώνεται στο οπτικοποιημένο Timeline), χωρίζονται σε μεγαλύτερα εκτελέσιμα κομμάτια, το οποίο δυσκολεύει την ισόποση κατανομή των tasks στους Executors.
6. Executors: Το σύνολο των ολοκληρωμένων Tasks (602) μοιράζονται 43% στον πρώτο Executor και 57% στον δεύτερο.
7. Ο Naive Bayes λόγω μικρότερης πολυπλοκότητας από την λογιστική παλινδρόμηση, δεν χρησιμοποιεί μέρος του σκληρού δίσκου του Server.

3rd Case Study Algorithm: Αποτελέσματα SVM



Spark UI, Timeline Process: SVM CS1

Στο παραπάνω παράρτημα απεικονίζεται το Timeline της εκτέλεσης του **SVM** . Στο πάνω μέρος της εικόνας απεικονίζονται οι Executors. Αρχικά, προστίθεται ο Executor driver που ορίζει τον τρόπο που θα κατανεμηθούν οι δουλειές στους υπόλοιπους executors. Στη συνέχεια προστίθεται ο Executor 2 και τέλος ο Executor 1 για να εκτελέσουν τις διεργασίες. Στο κάτω μέρος απεικονίζονται τα βήματα που ακολουθεί ο αλγόριθμος ως Jobs και η διαδρομή τους μέχρι να τυπωθεί το τελικό αποτέλεσμα.

Ο συνολικός αριθμός των jobs που λανβάνουν χώρα για την εκτέλεση του αλγορίμθου SVM είναι 42. Ο αριθμός των jobs είναι πολύ μεγαλύτερος καθώς γνωρίζουμε ότι έχει τη μεγαλύτερη πολυπλοκότητα συγκριτικά. Παρατηρείται ότι σε έκταση και σε χρόνο τα jobs είναι πανομοιότυπα. Λόγο της πολυπλοκότητάς του παρουσιάζεται επίσης μεγάλη αύξηση στο χρόνο εκτέλεσης του.

Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
Active(3)	0	3.3 MB / 1.2 GB	0.0 B	4	0	0	2022	2022	2.1 h (2.7 min)	217.1 GB	476.8 KB	476.8 KB
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
Total(3)	0	3.3 MB / 1.2 GB	0.0 B	4	0	0	2022	2022	2.1 h (2.7 min)	217.1 GB	476.8 KB	476.8 KB

Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs
driver	hadoop1:37401	Active	0	1.2 MB / 384.1 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	
1	hadoop2:43141	Active	0	1.1 MB / 384.1 MB	0.0 B	2	0	0	1030	1030	1.1 h (1.4 min)	108.1 GB	272.8 KB	237.9 KB	stdout stderr
2	hadoop1:43011	Active	0	1.1 MB / 384.1 MB	0.0 B	2	0	0	992	992	1.1 h (1.4 min)	109 GB	204 KB	238.9 KB	stdout stderr

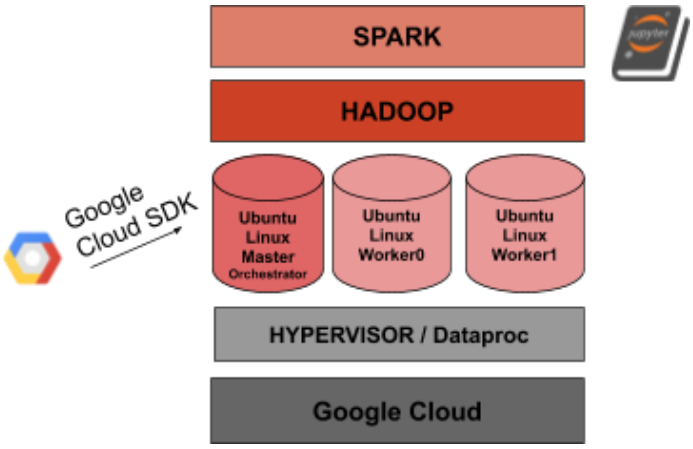
SPARK UI EXECUTORS SVM CS2

Παρατηρήσεις:

1. Event Timeline: Η επεξεργασία του κάθε job ορίζεται από τον executor driver να γίνεται παράλληλα στους executors
2. Ο maximum αριθμός των task για κάθε job του SVC είναι 67, ενώ για την παραγωγή των μετρικών φτάνει τα 120
3. Παρατηρούμε ότι καθ' όλη τη διάρκεια του process και οι τρεις executors είναι active.
4. Ο αριθμός των Jobs που δημιουργούνται είναι ο μεγαλύτερος μεταξύ των τριών αλγορίθμων, βοηθώντας την καλύτερη κατανομή των tasks μεταξύ των executors.
5. Executors: Το σύνολο των ολοκληρωμένων Tasks (2022) μοιράζονται 51% στον πρώτο Executor και 49% στον δεύτερο. Αυτό δείχνει ότι γίνεται καλύτερο distribution των tasks στους executors.
6. Ο SVC ενώ έχει την μεγαλύτερη πολυπλοκότητα από τους άλλους δύο αλγορίθμους δεν χρησιμοποιεί μέρος του σκληρού δίσκου του Server.

7.5.2 Case Scenario 2ο:

Η αρχιτεκτονική που χρησιμοποιήθηκε είναι η εξής:



Αρχιτεκτονική Case Scenario 2

Το infrastructure οικοδομήθηκε πάνω στην Υπηρεσία του Google Cloud. Για να δοθεί η δυνατότητα δημιουργίας Virtual Machines, επιλέχθηκαν τα παρακάτω resources με το εργαλείο process manager DataProc. Πάνω σε αυτόν εγκαταστάθηκαν 3 VMs με λειτουγικό Linux, τα οποία χωρίστηκαν σε Master-Orchestrator στο ένα, και Workers στα άλλα 2.

Αφού στήθηκε ολόκληρη η υποδομή του virtualization, επόμενο βήμα ήταν η εγκατάσταση του Hadoop ως file system και του Spark, το οποίο έγινε χειροκίνητα. Στη συνέχεια εγκαταστάθηκε η python ως γλώσσα προγραμματισμού για την ανάπτυξη των αλγορίθμων. Τέλος εγκαταστάθηκε το IDE περιβάλλον: Jupyter Notebook IDE το οποίο δίνει τη δυνατότητα λόγω του ότι έχει oline interface ανοίγοντας σε πόρτα του localhost μέσω του Google SDK, να μπορεί να το χρησιμοποιήσει εκτός των vm. Χρησιμοποιόντας τον kernel της python3 αναπτύχθηκαν οι αλγόριθμοι που αναφέρθηκαν στην αρχή του κεφαλαίου και στη συνέχεια συλλέχθηκαν τα αποτελέσματα που εξυπηρετούν το σκοπό της εργασίας. Το region των VMs ήταν στην δυτική Ευρώπη. Στο παρόν όπως και στο επόμενο σενάριο επιλέξαμε ισχυρά VMS σε υπολογιστική ισχύ , αποθηκευτικό χώρο και μνήμη με σκοπό να υπάρχει μεγάλη διαφορά μεταξύ του Local και των Cloud συστημάτων.

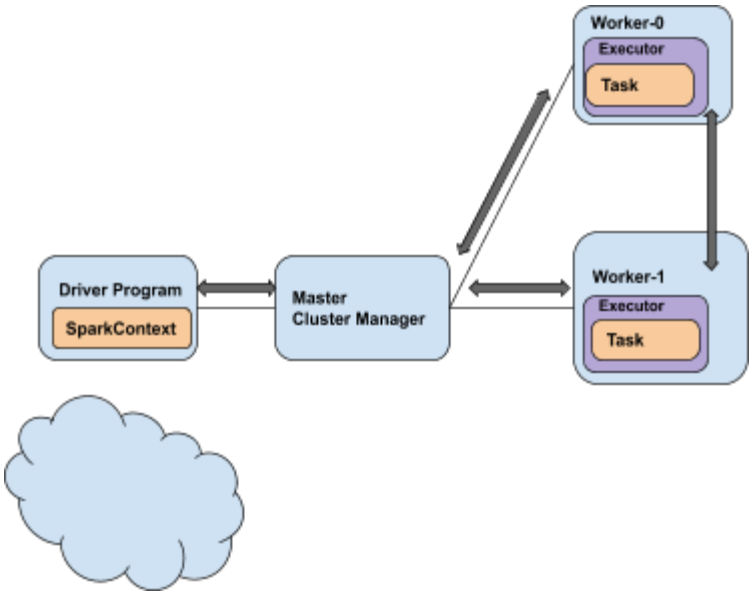
Παρακάτω αναφέρονται στον πίνακα τα χαρακτηριστικά των VM.

Scenario 2	**Master node**	**Worker node-0**	**Worker node-1**
Machine type	n1-standard-2	n1-standard-2	n1-standard-2
CPU	2 vCPU	2 vCPU	2 vCPU
RAM	7.50 GB memory	7.50 GB memory	7.50 GB memory
Primary disk type	pd-ssd	pd-ssd	pd-ssd
Primary disk size	100 GB	50 GB	50 GB
OS	1.4.3-ubuntu 18	1.4.3-ubuntu 18	1.4.3-ubuntu 18
Optional components	JUPYTER, ANACONDA	JUPYTER, ANACONDA	JUPYTER, ANACONDA

Πίνακας Υπολογιστικών Πόρων των Vms

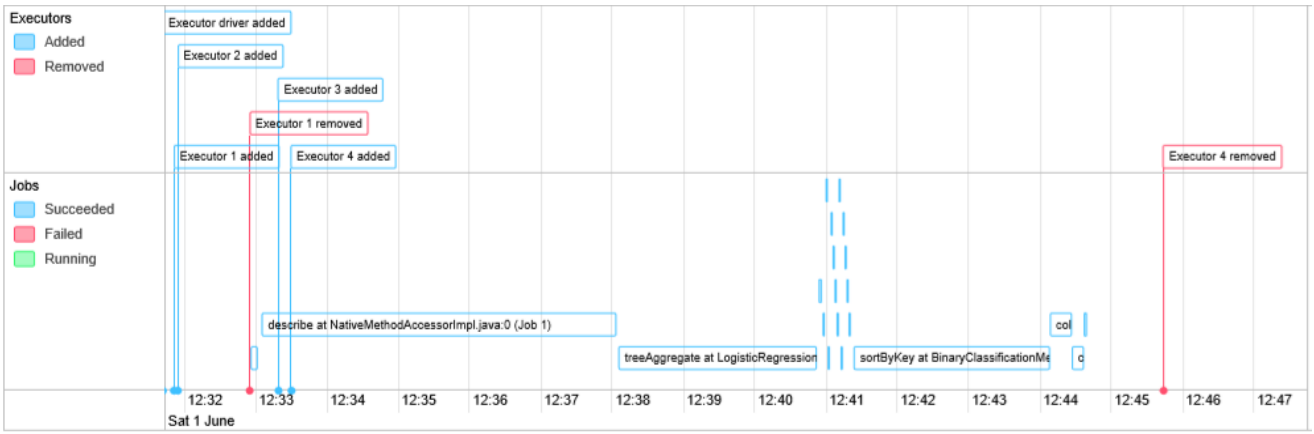
Σε αυτό το σενάριο, το cluster των υπολογιστών αποτελείται από 3 εικονικές μηχανές. Στην πρώτη περιλαμβάνεται ο Master Node/Cluster Manager ο οποίος οργανώνει και διαχειρίζεται τις υπόλοιπες 2, στη δεύτερη ο Worker Node-0 και στην τρίτη ο Worker Node-1 . Σε αυτή την αρχιτεκτονική, ο Master Node/Cluster Manager ορίζει τις διεργασίες και τις μοιράζει ανάλογα με τη διαθεσιμότητα των δύο

Workers, σε αυτό το VM δεν γίνεται κομμάτι της επεξεργασίας αλλά έχει το διπλάσιο όγκο διαθέσιμης αποθηκευτικής μνήμης (Hard Disk Size) σε σχέση με τους άλλους δύο ενώ τα υπόλοιπα resources και στους τρεις εικονικούς υπολογιστές είναι τα ίδια για την εκτέλεση των αλγορίθμων.



Internal Spark architecture CASE SCENARIO 2

1st Case Study Algorithm: Αποτελέσματα Λογιστικής Παλινδρόμησης



Spark UI, Timeline Process: Logistic Regression CS2

Στο παραπάνω παράρτημα απεικονίζεται το Timeline της εκτέλεσης της **Λογιστικής Παλινδρόμησης** . Στο πάνω μέρος της εικόνας απεικονίζονται οι Executors. Αρχικά, προστίθεται ο Executor driver που ορίζει τον τρόπο που θα κατανεμηθούν οι δουλειές στους υπόλοιπους executors. Στη συνέχεια προστίθεται ο Executor 1 και συνέχεια οι Executor 2,3,4 ενώ ο ο Executor 1 αποχωρεί πριν να προστεθούν ο 3 και ο 4 για να εκτελέσουν τις διεργασίες. Στο κάτω μέρος απεικονίζονται τα βήματα που ακολουθεί ο αλγόριθμος ως Jobs και η διαδρομή τους μέχρι να τυπωθεί το τελικό αποτέλεσμα.

Παρατηρείται ότι μεγάλο κομμάτι του χρόνου εκτέλεσης απασχολείται στο job: «describe at NativeMethodAccessorImpl: Java». Το Spark εξετάζει τη στοίβα αιτημάτων του αλγορίθμου και το χρησιμοποιεί για να ορίσει το job στο UI. Στη συνέχεια εκτελείται ο αλγόριθμος με τον ίδιο τρόπο όπως και στο πρώτο σενάριο, με τα “treeAggregate at Logistic Regression.scala” και “sortByKey at BinaryClassificationMetric.scala” jobs να καταλαμβάνουν τον μεγαλύτερο χρόνο σε εκτέλεση.

Επίσης, λόγο των ρυθμίσεων που έχουν γίνει από τον πάροχο στο cloud, οι executors αφαιρούνται αυτόματα από τη διαδικασία αμέσως μόλις τελειώσουν τα task που τους έχουν ανατεθεί από τον scheduler για να εξασφαλίζουν άμεση διαθεσιμότητα για τυχόν άλλα tasks. Στη συγκεκριμένη περίπτωση οι executors αυτοί είναι ο 1 και ο 4.

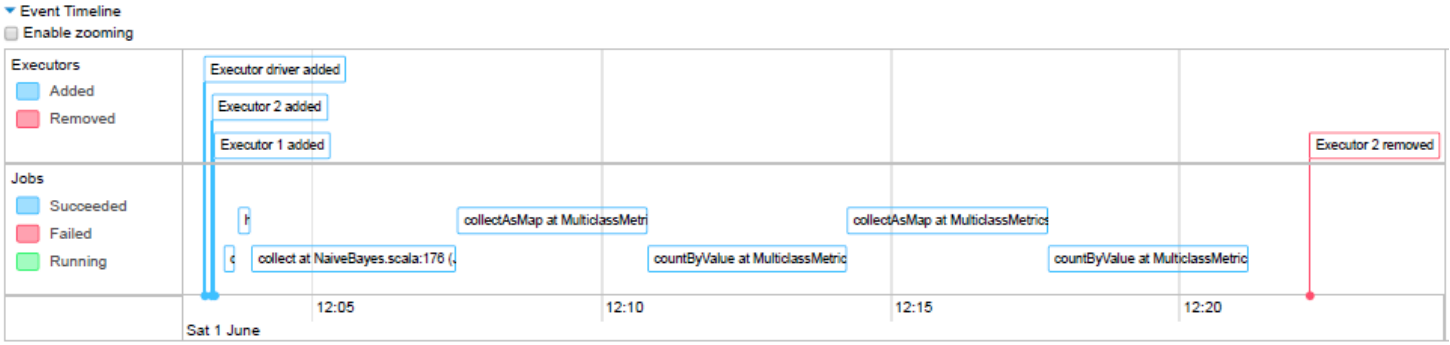
Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
driver	spark6-m.europe-west1-b.c.aerial-jigsaw-196012.internal:41787	Active	0	0.0 B / 979 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
1	spark6-w-0.europe-west1-b.c.aerial-jigsaw-196012.internal:38449	Dead	0	0.0 B / 1.4 GB	0.0 B	1	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
2	spark6-w-1.europe-west1-b.c.aerial-jigsaw-196012.internal:37845	Active	0	0.0 B / 1.4 GB	0.0 B	1	0	0	776	776	11 min (23 s)	20.3 GB	92.1 MB	46 MB
3	spark6-w-0.europe-west1-b.c.aerial-jigsaw-196012.internal:32795	Active	0	0.0 B / 1.4 GB	0.0 B	1	0	0	312	312	11 min (19 s)	8.3 GB	36.8 MB	18.2 MB
4	spark6-w-0.europe-west1-b.c.aerial-jigsaw-196012.internal:37261	Dead	0	0.0 B / 1.4 GB	0.0 B	1	0	0	342	342	11 min (22 s)	9.1 GB	37.6 MB	19.6 MB

SPARK UI, EXECUTORS: Logistic Regression CS2

Παρατηρήσεις:

1. Event Timeline: Η επεξεργασία του κάθε job ορίζεται από τον executor driver να γίνεται παράλληλα στους executors
2. Παρατηρούμε ότι καθ' όλη τη διάρκεια του process ο δεύτερος και τρίτος executor από τους τέσσερις είναι active, ενώ ο πρώτος "πέφτει" στην αρχή και ο τέταρτος στο τέλος της εκτέλεσης του αλγορίθμου .
3. Executors: Το σύνολο των ολοκληρωμένων Tasks (1430) μοιράζονται 54% στον δεύτερο Executor και 22% στον τρίτο και 24% στον τέταρτο.

2nd Case Study Algorithm: Αποτελέσματα Naive Bayes



Spark Time Line Process Naive Bayes CS2

Στο παραπάνω παράρτημα απεικονίζεται το Timeline της εκτέλεσης του **Naive Bayes** . Στο πάνω μέρος της εικόνας απεικονίζονται οι Executors. Αρχικά, προστίθεται ο Executor driver που ορίζει τον τρόπο που θα κατανεμηθούν οι δουλειές στους υπόλοιπους executors. Στη συνέχεια προστίθεται ο Executor 1 και στο τέλος ο Executor 2 ενώ ο Executor 4 αποχωρεί μετά το τέλος της εκτέλεσης των διεργασιών. Στο κάτω μέρος απεικονίζονται τα βήματα που ακολουθεί ο αλγόριθμος ως Jobs και η διαδρομή τους μέχρι να τυπωθεί το τελικό αποτέλεσμα.

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
driver	spark6-m.europe-west1-b.c.aerial-jigsaw-196012.internal:40803	Active	0	0.0 B / 979 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
1	spark6-w-0.europe-west1-b.c.aerial-jigsaw-196012.internal:46681	Active	0	0.0 B / 1.4 GB	0.0 B	1	0	0	211	211	17 min (31 s)	13.4 GB	36.2 KB	23.4 KB
2	spark6-w-1.europe-west1-b.c.aerial-jigsaw-196012.internal:45513	Dead	0	0.0 B / 1.4 GB	0.0 B	1	0	0	391	391	17 min (33 s)	26.9 GB	36 KB	48.8 KB

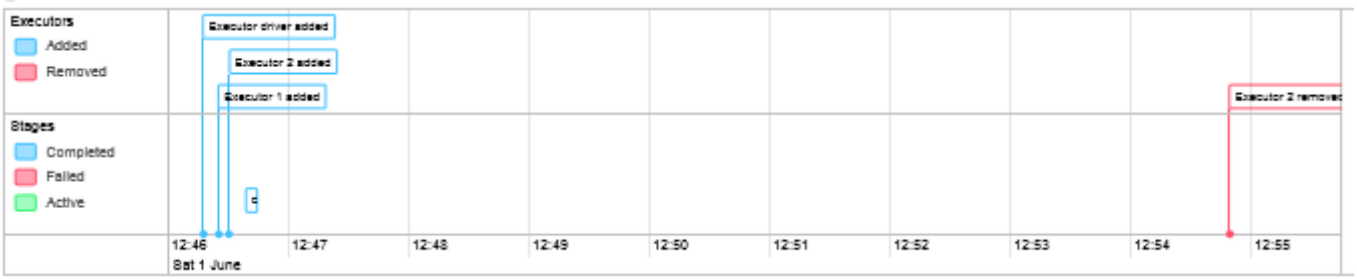
SPARK UI EXECUTORS Naïve Bayes CS2

Παρατηρήσεις:

1. Event Timeline: Η επεξεργασία του κάθε job ορίζεται από τον executor driver να γίνεται παράλληλα στους executors
2. Παρατηρούμε ότι καθ' όλη τη διάρκεια του process ο πρώτος executor από τους δυο είναι active, ενώ ο δεύτερος"πέφτει" στο τέλος της εκτέλεσης του αλγορίθμου .

3. Executors: Το σύνολο των ολοκληρωμένων Tasks (602) μοιράζονται 35% στον πρώτο Executor και 65% στον δεύτερο , μια επιπλέον παρατήρηση είναι ότι δεν υπάρχουν Failed Tasks. Το Shuffle Read ο Exec1 αλλά και Το Shuffle Write.

3nd Case Study Algorithm: Αποτελέσματα SVM



Spark Time Line Process SVM CS2

Στο παραπάνω παράρτημα απεικονίζεται το Timeline της εκτέλεσης του **SVM** . Στο πάνω μέρος της εικόνας απεικονίζονται οι Executors. Αρχικά, προστίθεται ο Executor driver που ορίζει τον τρόπο που θα κατανεμηθούν οι δουλειές στους υπόλοιπους executors. Στη συνέχεια προστίθεται ο Executor 1 και στο τέλος ο Executor 2 ενώ ο Executor 2 αποχωρεί μετά το τέλος της εκτέλεσης των διεργασιών. Στο κάτω μέρος απεικονίζονται τα βήματα που ακολουθεί ο αλγόριθμος ως Jobs και η διαδρομή τους μέχρι να τυπωθεί το τελικό αποτέλεσμα.

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
driver	spark6-m.europe-west1-b.c.aerial-jigsaw-196012.internal:44357	Active	0	0.0 B / 979 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
1	spark6-w-1.europe-west1-b.c.aerial-jigsaw-196012.internal:45019	Active	0	0.0 B / 1.4 GB	0.0 B	1	0	0	135	135	6.5 min (6 s)	11.6 GB	37.9 MB	113.6 MB
2	spark6-w-0.europe-west1-b.c.aerial-jigsaw-196012.internal:35963	Dead	0	0.0 B / 1.4 GB	0.0 B	1	0	0	136	136	6.6 min (5 s)	12.5 GB	64 B	121.5 MB

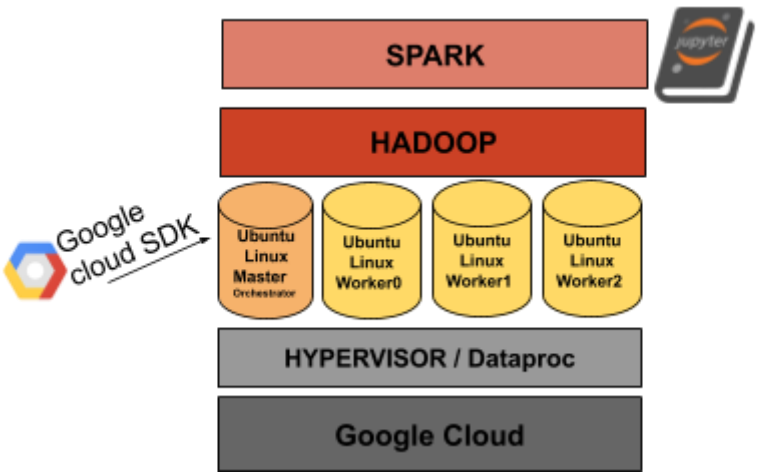
SPARK UI, EXECUTORS: SVM CS2

Παρατηρήσεις:

1. Event Timeline: Η επεξεργασία του κάθε job ορίζεται από τον executor driver να γίνεται παράλληλα στους executors
2. Παρατηρούμε ότι καθ' όλη τη διάρκεια του process ο πρώτος executor από τους δυο είναι active, ενώ ο δεύτερος"πέφτει" στο τέλος της εκτέλεσης του αλγορίθμου .
3. Executors: Το σύνολο των ολοκληρωμένων Tasks (271) μοιράζονται 49,8% στον πρώτο Executor και 50,2% στον δεύτερο , μια επιπλέον παρατήρηση είναι ότι δεν υπάρχουν Failed Tasks. Το Shuffle Read ο Exec1 αλλά και Το Shuffle Write.

7.5.3 Case Scenario 3ο:

Η αρχιτεκτονική που χρησιμοποιήθηκε είναι η εξής:



Αρχιτεκτονική Case Scenario 3

Το infrastructure οικοδομήθηκε πάνω στην Υπηρεσία του Google Cloud. Για να δοθεί η δυνατότητα δημιουργίας Virtual Machines, επιλέχθηκαν τα παρακάτω resources με το εργαλείο process manager DataProc. Πάνω σε αυτόν εγκαταστάθηκαν 4 VMs με λειτουγικό Linux, τα οποία χωρίστηκαν σε Master-Orchestrator στο ένα, και Workers στα άλλα 4.

Αφού στήθηκε ολόκληρη η υποδομή του virtualization, επόμενο βήμα ήταν η εγκατάσταση του Hadoop ως file system και του Spark, το οποίο έγινε χειροκίνητα. Στη συνέχεια εγκαταστάθηκε η python ως η γλώσσα προγραμματισμού που χρησιμοποιήθηκε για την ανάπτυξη των αλγορίθμων. Τέλος εγκαταστάθηκε το IDE περιβάλλον: Jupyter Notebook IDE το οποίο δίνει τη δυνατότητα λόγω του ότι έχει oline interface ανοίγοντας σε πόρτα του localhost μέσω του Google SDK, να μπορεί να το χρησιμοποιήσει εκτός των vm. Χρησιμοποιόντας τον kernel της python3 αναπτύχθηκαν οι αλγόριθμοι που αναφέρθηκαν στην αρχή του κεφαλαίου και στη συνέχεια συλλέχθηκαν τα αποτελέσματα που εξυπηρετούν το σκοπό της εργασίας. Το region των VMs ήταν στην δυτική Ευρώπη.

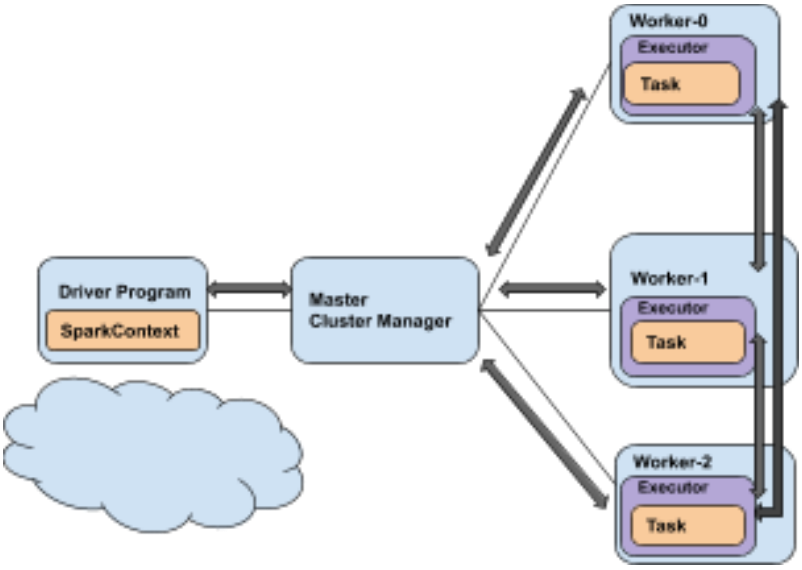
Παρακάτω αναφέρονται στον πίνακα τα χαρακτηριστικά των VM.

Τα χαρακτηριστικά του συστήματος είναι τα εξής:

Scenario 3	**Master node**	**Worker node-0**	**Worker node-1**	**Worker node-2**
Machine type	n1-standard-2	n1-standard-2	n1-standard-2	n1-standard-2
CPU	2 vCPU	2 vCPU	2 vCPU	2 vCPU
RAM	7.50 GB memory	7.50 GB memory	7.50 GB memory	7.50 GB memory
Primary disk type	pd-ssd	pd-ssd	pd-ssd	pd-ssd
Primary disk size	100 GB	50 GB	50 GB	50 GB
OS	1.4.3-ubuntu 18	1.4.3-ubuntu 18	1.4.3-ubuntu 18	1.4.3-ubuntu 18
Optional components	JUPYTER, ANACONDA	JUPYTER, ANACONDA	JUPYTER, ANACONDA	JUPYTER, ANACONDA

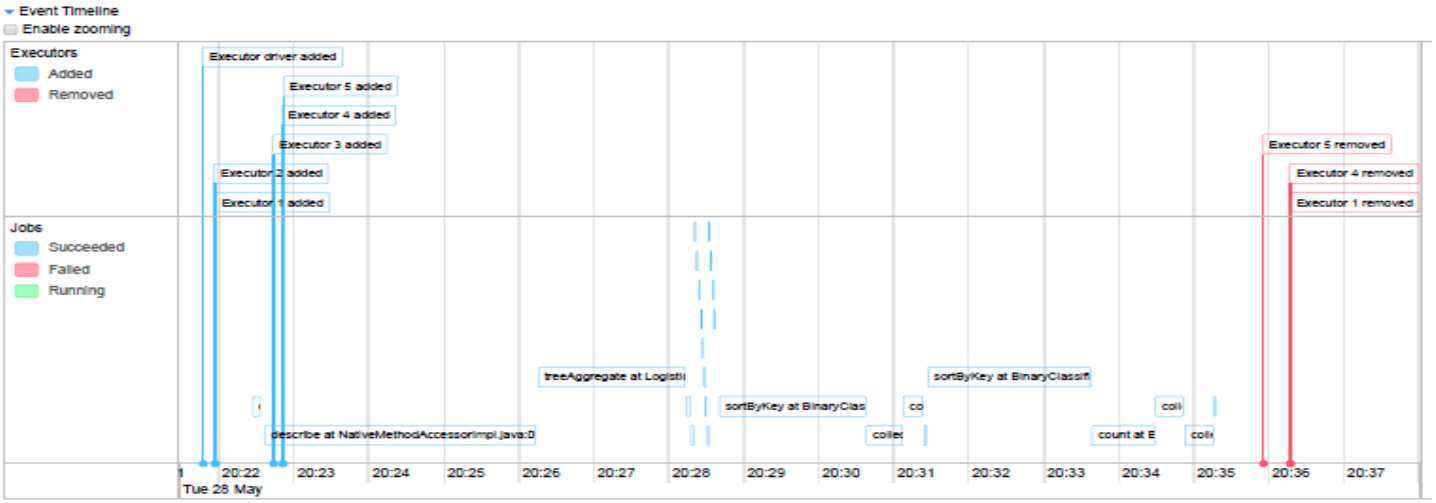
Πίνακας Υπολογιστικών Πόρων των Vms

Σε αυτό το σενάριο, το cluster των υπολογιστών αποτελείται από 4 εικονικές μηχανές. Στην πρώτη περιλαμβάνεται ο Master Node/Cluster Manager ο οποίος οργανώνει και διαχειρίζεται τις υπόλοιπες 3, στη δεύτερη ο Worker Node-0, στην τρίτη ο Worker Node-1, και στην τρίτη ο Worker Node-2. Σε αυτή την αρχιτεκτονική, ο Master Node/Cluster Manager ορίζει τις διεργασίες και τις μοιράζει ανάλογα με τη διαθεσιμότητα των τριών Workers, σε αυτό το VM δεν εκτελείται κομμάτι της επεξεργασίας αλλά έχει το διπλάσιο όγκο διαθέσιμης αποθηκευτικής μνήμης (Hard Disk Size) σε σχέση με τους άλλους τρεις ενώ τα υπόλοιπα resources και στους τρεις εικονικούς υπολογιστές είναι τα ίδια για την εκτέλεση των αλγορίθμων.



Internal Spark architecture CASE SCENARIO 3

1st Case Study Algorithm: Αποτελέσματα Λογιστικής Παλινδρόμησης



Spark Time Line Process Logistic Regression CS3

Στο παραπάνω παράρτημα απεικονίζεται το Timeline της εκτέλεσης της Λογιστικής Παλινδρόμησης. Στο πάνω μέρος της εικόνας απεικονίζονται οι Executors. Αρχικά, προστίθεται ο Executor driver που ορίζει τον τρόπο που θα κατανεμηθούν οι δουλειές στους υπόλοιπους executors. Στη συνέχεια προστίθεται ο Executor 1 και στο τέλος οι Executors 2, 3, 4, 5 ενώ οι Executors 1, 4 και 5, αποχωρούν μετά το τέλος της εκτέλεσης των διεργασιών. Στο κάτω μέρος απεικονίζονται τα βήματα που ακολουθεί ο αλγόριθμος ως Jobs και η διαδρομή τους μέχρι να τυπωθεί το τελικό αποτέλεσμα.

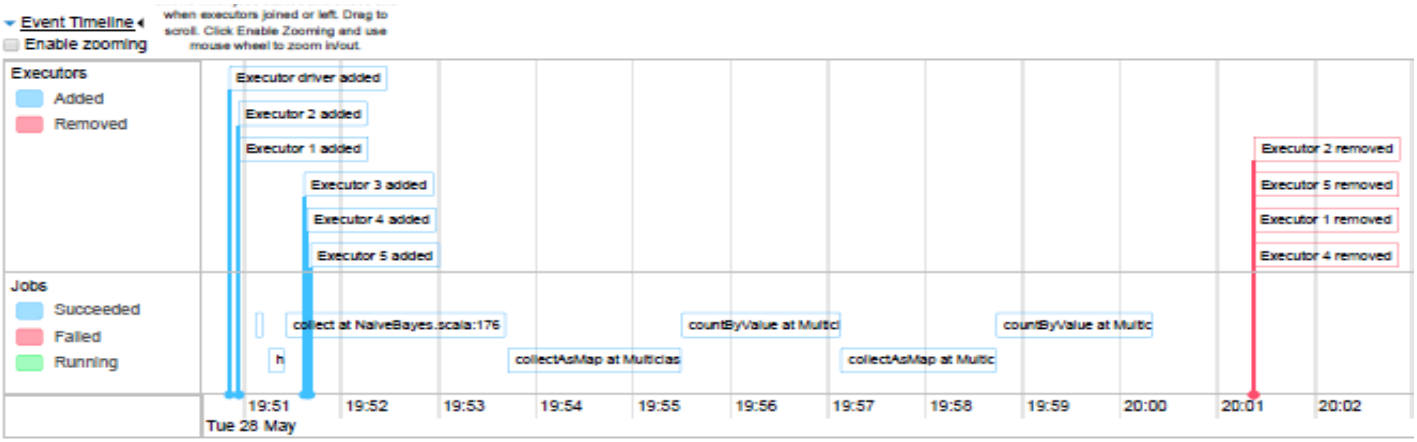
Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
driver	spark6-m.europe-west1-b.c.aerial-jigsaw-196012.internal:38883	Active	0	0.0 B / 979 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
1	spark6-w-1.europe-west1-b.c.aerial-jigsaw-196012.internal:44879	Dead	0	0.0 B / 1.4 GB	0.0 B	1	0	0	104	104	9.0 min (15 s)	6.8 GB	0.0 B	12.7 KB
2	spark6-w-2.europe-west1-b.c.aerial-jigsaw-196012.internal:38435	Dead	0	0.0 B / 1.4 GB	0.0 B	1	0	0	185	185	8.5 min (23 s)	13.8 GB	72.2 KB	25.7 KB
3	spark6-w-1.europe-west1-b.c.aerial-jigsaw-196012.internal:35153	Active	0	0.0 B / 1.4 GB	0.0 B	1	0	0	106	106	8.5 min (17 s)	6.7 GB	0.0 B	12 KB
4	spark6-w-0.europe-west1-b.c.aerial-jigsaw-196012.internal:38429	Dead	0	0.0 B / 1.4 GB	0.0 B	1	0	0	104	104	8.5 min (11 s)	6.6 GB	0.0 B	11.3 KB
5	spark6-w-0.europe-west1-b.c.aerial-jigsaw-196012.internal:35163	Dead	0	0.0 B / 1.4 GB	0.0 B	1	0	0	103	103	8.5 min (16 s)	6.4 GB	0.0 B	10.5 KB

SPARK UI EXECUTORS Logistic Regression CS3

Παρατηρήσεις:

1. Event Timeline: Η επεξεργασία του κάθε job ορίζεται από τον executor driver να γίνεται παράλληλα στους executors
2. Παρατηρούμε ότι καθ' όλη τη διάρκεια του process και οι πέντε executors είναι active, ενώ ο πρώτος, ο τέταρτος και ο πέμπτος "πέφτουν" στο τέλος της εκτέλεσης του αλγορίθμου .
3. Executors: Το σύνολο των ολοκληρωμένων Tasks (602) μοιράζονται 17,3% στον πρώτο Executor 30,7% στον δεύτερο, 17,6% στον τρίτο, 17,3% στον τέταρτο και 17,1% στον πέμπτο ,Δεν υπάρχουν Failed Tasks. Το Shuffle Read ο Exec1 αλλά και Το Shuffle Write.

2nd Case Study Algorithm: Αποτελέσματα Naive Bayes



Spark Time Line Process Naive Bayes CS3

Στο παραπάνω παράρτημα απεικονίζεται το Timeline της εκτέλεσης του **Naive Bayes** . Στο πάνω μέρος της εικόνας απεικονίζονται οι Executors. Αρχικά, προστίθεται ο Executor driver που ορίζει τον τρόπο που θα κατανεμηθούν οι δουλειές στους υπόλοιπους executors. Στη συνέχεια προστίθεται ο Executor 2 και στο τέλος οι Executors 1,3.4.5 ενώ οι Executors 2,5,1 και 4 αποχωρούν μετά το τέλος της εκτέλεσης των διεργασιών. Στο κάτω μέρος απεικονίζονται τα βήματα που ακολουθεί ο αλγόριθμος ως Jobs και η διαδρομή τους μέχρι να τυπωθεί το τελικό αποτέλεσμα.

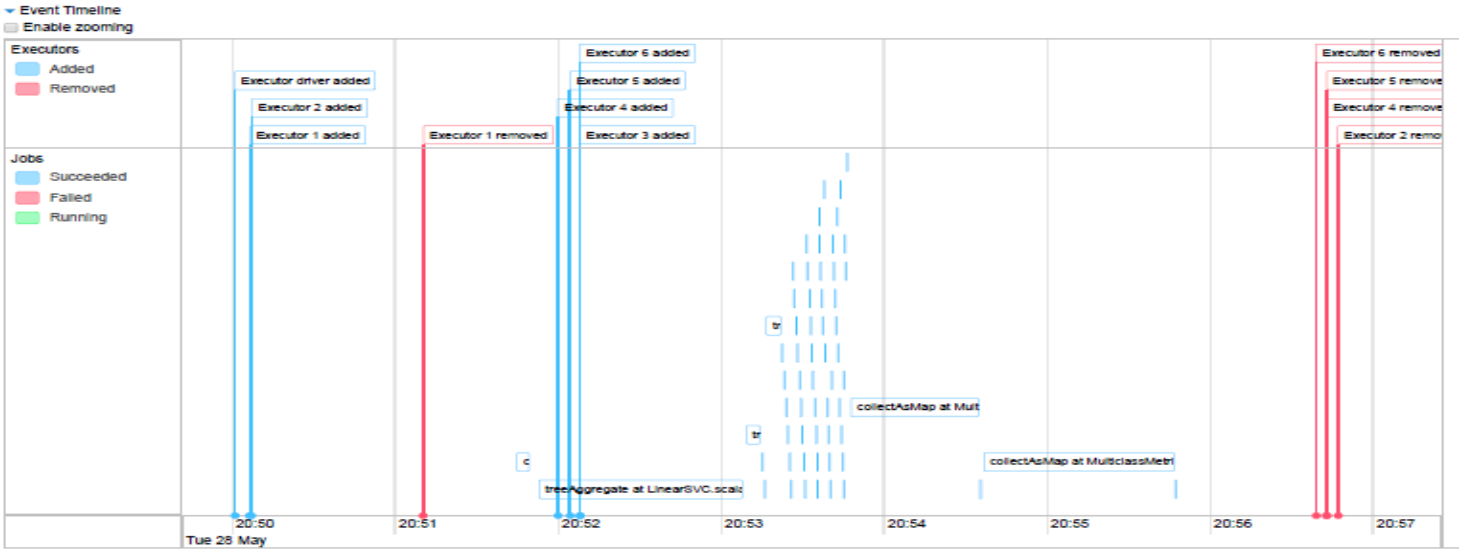
Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
driver	spark6-m.europe-west1-b.c.aerial-jigsaw-196012.internal:43987	Active	0	0.0 B / 979 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
1	spark6-w-2.europe-west1-b.c.aerial-jigsaw-196012.internal:38759	Dead	0	0.0 B / 1.4 GB	0.0 B	1	0	0	182	182	9.5 min (19 s)	7.1 GB	134.2 KB	22 MB
2	spark6-w-0.europe-west1-b.c.aerial-jigsaw-196012.internal:42285	Active	0	0.0 B / 1.4 GB	0.0 B	1	0	0	197	197	9.5 min (18 s)	7.5 GB	159.2 KB	23.4 MB
3	spark6-w-1.europe-west1-b.c.aerial-jigsaw-196012.internal:35331	Active	0	0.0 B / 1.4 GB	0.0 B	1	0	0	1078	1078	12 min (20 s)	16.2 GB	628 MB	177.5 MB
4	spark6-w-2.europe-west1-b.c.aerial-jigsaw-196012.internal:33651	Dead	0	0.0 B / 1.4 GB	0.0 B	1	0	0	197	197	9.2 min (21 s)	7.5 GB	132 KB	23.7 MB
5	spark6-w-0.europe-west1-b.c.aerial-jigsaw-196012.internal:37411	Dead	0	0.0 B / 1.4 GB	0.0 B	1	0	0	200	200	9.2 min (19 s)	7.5 GB	179.1 KB	23.7 MB

SPARK UI EXECUTORS Naïve Bayes CS3

Παρατηρησεις:

1. Event Timeline: Η επεξεργασία του κάθε job ορίζεται από τον executor driver να γίνεται παράλληλα στους executors
2. Παρατηρούμε ότι καθ' όλη τη διάρκεια του process και οι πέντε executors είναι active, ενώ ο πρώτος,ο δεύτερος, ο τέταρτος και ο πέμπτος "πέφτουν" στο τέλος της εκτέλεσης του αλγορίθμου .
3. Executors: Το σύνολο των ολοκληρωμένων Tasks (1854) μοιράζονται 9.9% στον πρώτο Executor 10.6% στον δεύτερο, 58.1% στον τρίτο, 10.6% στον τέταρτο και 10.8% πέμπτο .Δεν υπάρχουν Failed Tasks. Το Shuffle Read ο Exec1 αλλά και Το Shuffle Write.

3rd Case Study Algorithm: Αποτελέσματα SVM



Spark Time Line Process SVM CS3

Στο παραπάνω παράρτημα απεικονίζεται το Timeline της εκτέλεσης του **SVM** . Στο πάνω μέρος της εικόνας απεικονίζονται οι Executors. Αρχικά, προστίθεται ο Executor driver που ορίζει τον τρόπο που θα κατανεμηθούν οι δουλειές στους υπόλοιπους executors. Στη συνέχεια προστίθεται ο Executors 2 και 1-αλλά ο Executor 1 αποχωρεί πριν εκτέλεση της διαδικασίας- και στο τέλος οι Executors 4,5,6,3 ενώ οι Executors 6,5,4 και 2 αποχωρούν μετά το τέλος της εκτέλεσης των διεργασιών.Στο κάτω μέρος

απεικονίζονται τα βήματα που ακολουθεί ο αλγόριθμος ως Jobs και η διαδρομή τους μέχρι να τυπωθεί το τελικό αποτέλεσμα.

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
driver	spark6-m.europe-west1-b.c.aerial-jigsaw-196012.internal:37173	Active	0	0.0 B / 979 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
1	spark6-w-0.europe-west1-b.c.aerial-jigsaw-196012.internal:44007	Dead	0	0.0 B / 1.4 GB	0.0 B	1	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
2	spark6-w-1.europe-west1-b.c.aerial-jigsaw-196012.internal:45045	Dead	0	0.0 B / 1.4 GB	0.0 B	1	0	0	53	53	3.2 min (3 s)	3.9 GB	66.6 MB	37.9 MB
3	spark6-w-2.europe-west1-b.c.aerial-jigsaw-196012.internal:35287	Active	0	0.0 B / 1.4 GB	0.0 B	1	0	0	41	41	2.6 min (4 s)	4.8 GB	192 B	47 MB
4	spark6-w-0.europe-west1-b.c.aerial-jigsaw-196012.internal:34161	Dead	0	0.0 B / 1.4 GB	0.0 B	1	0	0	77	77	3.0 min (3 s)	7.5 GB	27.4 MB	73.1 MB
5	spark6-w-1.europe-west1-b.c.aerial-jigsaw-196012.internal:36719	Dead	0	0.0 B / 1.4 GB	0.0 B	1	0	0	39	39	2.9 min (3 s)	3.5 GB	11.8 MB	34 MB
6	spark6-w-2.europe-west1-b.c.aerial-jigsaw-196012.internal:40205	Dead	0	0.0 B / 1.4 GB	0.0 B	1	0	0	36	36	2.6 min (4 s)	4.4 GB	18.3 MB	43.1 MB

SPARK UI, EXECUTORS: SVM CS3

Παρατηρήσεις:

1. Event Timeline: Η επεξεργασία του κάθε job ορίζεται από τον executor driver να γίνεται παράλληλα στους executors
2. Παρατηρούμε ότι καθ' όλη τη διάρκεια του process και οι πέντε executors είναι active, ενώ ο πρώτος πέφτει στην αρχή ενώ ο έκτος,ο δεύτερος, ο τέταρτος και ο πέμπτος "πέφτουν" στο τέλος της εκτέλεσης του αλγορίθμου .
3. Executors: Το σύνολο των ολοκληρωμένων Tasks (246) μοιράζονται 0% στον πρώτο Executor 21,6% στον δεύτερο, 16,6% στον τρίτο, 31,4% στον τέταρτο, 15.8% στο πέμπτο και 14,6% στον έκτο. Δεν υπάρχουν Failed Tasks. Το Shuffle Read ο Exec1 αλλά και Το Shuffle Write.

7.6 Αποτελέσματα

	Algorithm	Time	Time(s)	Distribution of Tasks in Executors %	Dataset	Rows
1st_Scenario	LR	4.63mins	278s	53 - 47	7.48G	11bilion
	NB	6.78mins	407s	57 -43	7.48G	11bilion
	SVC	32.55mins	1953s	51 - 49	7.48G	11bilion
2nd_Scenario	LR	12.92mins	775.4s	54 - 24 - 22	7.48G	11bilion
	NB	17.68mins	1060.8s	65 - 35	7.48G	11bilion
	SVC	6.94mins	416.4s	50 -50	7.48G	11bilion
3rd_Scenario	LR	12.64mins	758.4s	31 - 18 - 17 - 17 - 17	7.48G	11bilion
	NB	9.15mins	549s	58 - 11 - 11 - 10 - 10	7.48G	11bilion
	SVC	3.88mins	232.7s	31 - 22 - 17 - 15 - 15	7.48G	11bilion

ΣΥΓΚΕΝΤΡΩΤΙΚΟΣ ΠΙΝΑΚΑΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ – ΜΕΤΡΗΣΕΩΝ

Στον πίνακα των αποτελεσμάτων παρατειρείται ότι ο χρόνος εκτέλεσης των αλγορίθμων της Λογιστικής Παλινδρόμησης και του Naïve Bayes στο πρώτο σενάριο είναι εμφανώς μειωμένος σε σχέση με τα επόμενα σενάρια. Από την άλλη ο SVM, παρά την μεγαλύτερη πολυπλοκότητά του, στο πρώτο σενάριο εμφανίζεται να υλοποιείται σε πολλαπλάσιο χρόνο απ' ότι στα επόμενα δύο σενάρια.

Όσον αναφορά την κατανομή των "tasks" στους "executors", παραιτειρείται ότι:

1. Στο 1ο σενάριο, ο SVM κατανέμει με πολύ καλύτερο τρόπο τα "tasks". Ο χειρότερος αλγόριθμος σε κατανομή εμφανίζεται να είναι ο NB, ο οποίος χωρίζει σε πολύ λιγότερα "jobs" την επεξεργασία των δεδομένων.
2. Όσο προστίθενται VM και resources όπως στα επόμενα 2 σενάρια, τα "tasks" δεν μοιράζονται ισόποσα με αποτέλεσμα κάποιοι "executors" να επιβαρύνονται περισσότερο.
3. Ο αλγόριθμος του SVM παραμένει ο καλύτερος κατανεμητής των "task" σε όλα τα σενάρια.

7.7 Case Scenario Analysis:

Spark Dependability Results:

Χρησιμοποιώντας τις ρυθμίσεις του Spark για την κατανομή των πόρων (4 CPU, 2 Executors, 8GB Memory) παρατηρούμε ότι τα περισσότερα issues προέκυψαν στην εκτέλεση του αλγορίθμου της Λογιστικής Παλινδρόμησης. Το μήνυμα που εμφανίστηκε κατ' επανάληψη ήταν το εξής: «memory.MemoryStore: Not enough space to cache».

Η αξιολόγηση και η βελτιστοποίηση των επαναληπτικών αλγορίθμων παρουσιάζουν μια ιδιαίτερα ενδιαφέρουσα συμπεριφορά στα συστήματα ροής δεδομένων όπως το Spark. Σε ορισμένες επαναληπτικές εφαρμογές, τα lineage graphs μπορούν να γίνουν αρκετά μεγάλα. Το Spark υλοποιείται στη Scala, μια λειτουργική και αντικειμενοστραφή γλώσσα προγραμματισμού που τρέχει στο Java Virtual Machine (JVM) και ενσωματώνεται απρόσκοπτα με τα υπάρχοντα προγράμματα που έχουν γραφτεί σε Java. Το Spark αποθηκεύει αντικείμενα δεδομένων και πληροφορίες DAG που σχετίζονται με κάθε στάδιο του JVM heap στον driver node.

Τα συστήματα ροής δεδομένων είναι μια αφαίρεση βασισμένη σε κατευθυνόμενες ακυκλικές γραφικές παραστάσεις, οι οποίες σχεδιάστηκαν αρχικά για εργασίες αποκοπής, φιλτραρίσματος, συσσώρευσης και μετασχηματισμού. Αυτή η αφαίρεση είναι επίσης κατάλληλη για επαναληπτικές εργασίες όπως αλγόριθμοι μηχανικής μάθησης. Το Spark δεν είναι καλά βελτιστοποιημένο για να αντιμετωπίσει τις αυξανόμενες επαναλήψεις, όπου πρέπει να ενημερωθεί μια μεταβλητή κατάσταση και να μεταφερθεί στην επόμενη επανάληψη. Το immutability των RDD επηρεάζει σημαντικά την υπολογιστική αποτελεσματικότητα των αλγορίθμων με αυξημένες επαναλήψεις, καθώς δεν μπορεί να εκμεταλλευτεί τις αραιές εξαρτήσεις αυτών των εργασιών.

Η εφαρμογή του Spark σχεδιάστηκε για να χειρίζεται ένα ορισμένο αριθμό χρηστών ή ένα ορισμένο ποσό δεδομένων. Όταν ο αριθμός των χρηστών ή ο όγκος των δεδομένων ξαφνικά σηκώνεται και διασχίζει αυτό το αναμενόμενο όριο, ενεργοποιεί το java.lang.OutOfMemoryError: Java heap space. Ένας ιδιαίτερος τύπος σφάλματος προγραμματισμού θα οδηγήσει την εφαρμογή να καταναλώνει συνεχώς περισσότερη μνήμη. Κάθε φορά που χρησιμοποιείται η διαρροή των λειτουργιών της εφαρμογής αφήνει κάποια αντικείμενα πίσω στο χώρο Java (**Garbage Collection**).
Το java.lang.OutOfMemoryError: Σφάλμα Java heap space θα ενεργοποιείται -έτσι κι αλλιώς αφού δεν υπαρχει παραμετροποίηση σε αυτό, στον αλγόριθμο και το dataset - όταν η εφαρμογή επιχειρήσει να προσθέσει περισσότερα δεδομένα στην περιοχή του σωρού, αλλά δεν υπάρχει αρκετός χώρος για αυτό ανεξάρτητα με την ρύθμιση των πόρων υπολογιστικής ισχύς μνήμης και αποθηκευτικού χώρου. Σημειώνεται ότι μπορεί να υπάρχει αρκετή διαθέσιμη φυσική μνήμη, αλλά το java.lang.OutOfMemoryError ρίχνεται κάθε φορά που η JVM φτάνει στο όριο μεγέθους σωρού.

Για να ερμηνευθεί καλύτερα το συγκεκριμένο φαινόμενο, υλοποιήθηκε ένα επιπλέον application της λογιστικής παλινδρόμησης, κόβοντας τα resources στο μισό (2 CPU, 2 Executors, 2GB Memory). Το αποτέλεσμα αυτού του σεναρίου ήταν ο αλγόριθμος να υλοποιηθεί σε μεγαλύτερο χρόνο όπως ήταν αναμενόμενο, αλλά χρησιμοποιώντας πάλι την ακριβώς την ίδια ποσότητα αποθηκευτικού χώρου στον σκληρό δίσκο.

Σε αντίθεση με την Λογιστική Παλινδρόμηση οι αλγόριθμοι του Naive Bayes (που χρησιμοποιεί λιγότερη υπολογιστική ισχύ από την Λ.Π.) και του Linear SVM (που χρησιμοποιεί περισσότερη ισχύ λόγο πολυπλοκότητας), δεν παρουσιάζουν παρόμοιο πρόβλημα.

Αυτό οδήγησε στο συμπέρασμα ότι η πιο πιθανή εξήγηση είναι ότι, η Λ.Π. δημιουργεί μεγαλύτερο αριθμό αντικειμένων στο heap της Java για να ολοκληρώση τις επαναλήψεις που θα παράγουν το αποτέλεσμα, αυξάνοντας την ποσότητα του Garbage Collection που αποθηκεύεται στη μνήμη. Το Spark ως **Fault Tolerant** σύστημα ροής δεδομένων έχει τη δυνατότητα να χρησιμοποιεί το δίσκο στις περιπτώσεις όπου η μνήμη δεν επαρκεί (java.lang.OutOfMemoryError) και δημιουργείται πρόβλημα στο heap του JVM από πιθανή αύξηση του GC. Με αυτόν τρόπο παραμένει σταθερό στην επεξεργασία δεδομένων μέχρι να ολοκληρωθεί η διεργασία, ακόμα και αν υπάρξουν καθυστερήσεις.

Επιπροσθέτως, έχει παρατηρηθεί ότι σε iterative αλγορίθμους όπως είναι η Λ.Π., το να διατηρούνται τα RDDs στη μνήμη, βελτιώνει την απόδοση σε ταχύτητα της επεξεργασίας. Γι αυτό το σκοπό όπως αναφέραμε στο θεωρητικό κομμάτι του Spark , υπάρχουν οι τεχνικές του persist και caching με δυνατότητα επεξεργασίας μόνο στη μνήμη ή ακόμα και μόνο στο σκληρό.

Ένα άλλο issue που προέκυψε στους αλγόριθμους του Naive Bayes και του SVM ήταν η απουσία της βιβλιοθήκης ‘BLAS’. Το issue εμφανίστηκε με το μήνυμα: «netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS»

Το μήνυμα “Failed to load implementation” είναι μόνο μια προειδοποίηση. Ο Spark χρησιμοποιεί το BLAS για να πραγματοποιήσει υπολογισμούς. Το BLAS έχει ενσωματωμένες εφαρμογές και υλοποίηση JVM, το εγγενές είναι πιο βελτιστοποιημένο / ταχύτερο. Ωστόσο, πρέπει να εγκαταστήσετε τη μητρική βιβλιοθήκη χειροκίνητα και μεμονωμένα.

Χωρίς αυτή τη διαμόρφωση θα εμφανιστεί το προειδοποιητικό μήνυμα και η Spark θα χρησιμοποιήσει την εφαρμογή JVM του BLAS. Τα αποτελέσματα θα πρέπει να είναι τα ίδια αλλά ο χρόνος εκτέλεσης των αλγορίθμων ενδέχεται να παρουσιάσει αρκετές καθυστερήσεις

Συμπερασματικά, όταν το Hadoop και το Spark εγκαθίσταται σε τοπικό επίπεδο υπάρχουν μία σειρά από βελτιστοποιήσεις που πρέπει να λάβει υπόψη του ο χρήστης προκειμένου να επιτύχει τα καλύτερα δυνατά αποτελέσματα. Οι βελτιστοποιήσεις αυτές μπορούν να γίνουν χειροκίνητα από τον χρήστη, ο οποίος θα πρέπει να έχει καλή γνώση των αναγκών και του infrastructure.

Παρ ‘ολ’ αυτά το Spark μπορεί να ξεπερνά τις ελλείψεις που μπορεί να εμφανίζονται στη διάρκεια εκτέλεσης των αλγορίθμων του machine learning (συνυπολογίζοντας βέβαια και το κόστος) και να διατηρεί μία σταθερή επεξεργαστική υποδομή με αυξημένο Dependability.

Βιβλιογραφία

1. Algorithmic Fairness, Jon Kleinberg. *https://www.cs.cornell.edu/home/kleinber/aer18-fairness.pdf.* χ.χ.
2. Algorithms, Introduction to. *Thomash, Charlese, Ronaldl, Clifford stein.* χ.χ.
3. Analysis, Learning Spark: Lightning-Fast Big Data, και Learning Spark: Lightning-Fast Big Data Analysis Learning Spark: Lightning-Fast Big Data Analysis Learning Spark: Lightning-Fast Big Data Analysis. *Ουέντελ, Άντυ Κονουινσκι και Πάτρικ.* χ.χ.
4. Availability,SystemReliabilityand. *https://www.eventhelix.com/RealtimeMantra/FaultHandling/system_reliability_availability.htm.* χ.χ.
5. Classification.*https://spark.apache.org/docs/latest/ml-classification-regression.html#classification.* χ.χ.
6. CloudSim, Dynamic virtual machine allocation policy in cloud computing complying with service level agreement using. *Parikh Aneri and S Sumathy 2017.* χ.χ.
7. Computing, A Survey on Resource Allocation Strategies in Cloud. *V.Vinothina, Sr.Lecturer, Dr.R.Sridaran, Dean, Dr.PadmavathiGanapathi.* χ.χ.
8. DataProc, Initialization actions. *https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/init-actions.* χ.χ.
9. Degradation, Adaptive Fault Tolerance and Graceful. *Oscar González.* χ.χ.
10. Degradation_Adaptive_Fault_Tolerance_and_Graceful. *González, Oscar.* χ.χ.
11. Dependability. *https://en.wikipedia.org/wiki/Dependability.* χ.χ.
12. Design_Fault_Tolerant. *Elena, Dubrova.* χ.χ.
13. Difference?, Containers vs. Virtual Machines (VMs): What's the. *https://blog.netapp.com/blogs/containers-vs-vms/.* χ.χ.
14. Environment, A Review: Resource Allocation Problem in Cloud. *Prof. Rupali M.Pandharpatte.* χ.χ.
15. Fault_tolerance. *https://en.wikipedia.org/wiki/Fault_tolerance.* χ.χ.
16. Flows, Spinning Fast Iterative Data. *Stephan Ewen. Kostas Tzoumas. Moritz Kaufmann. Volker Markl.* χ.χ.
17. FPGA-Acceleration of Machine Learning in Cloud Computing, a case study using Logistic Regression. *Ηλίας Ν. Κορομηλάς.* χ.χ.
18. Guide, Hadoop: The Definitive. *White, Tom.* χ.χ.
19. HIGGSDataSet. *https://archive.ics.uci.edu/ml/datasets/HIGGS?fbclid=IwAR05UYnCAsq1kJJHXrN5ThChLzx7ssOHb7ZlcGmeJTKR1g_eKG1C6KIgn6o.* χ.χ.
20. IV/SQL, Σχεδιασμός Βάσεων Δεδομένων DBASE. *Βασιλακόπουλος, Γ.* χ.χ.
21. Mapping, Characterization and Acceleration of Apache Spark Applications. *Ιωάννης Γ. Σταμέλος.* χ.χ.
22. MapReduce, A Survey of Large-Scale Analytical Query Processing in. *Christos Doulkeridis · Kjetil Nørvag.* χ.χ.
23. methods, A comparative assessment of classification. Melody Y. Kiang, χ.χ.
24. Metrics, Metrics Dashboard: A Hosted Platform for Software Quality. *George K. Thiruvathukal∗, Shilpika, Nicholas J. Hayward, and Konstantin L¨aufer .* χ.χ.

25. ML, Apache Spark and RDDs: Distributed resilient in-memory abstraction for. *https://medium.com/coinmonks/apache-spark-and-rdds-fault-tolerant-distributed-in-memory-data-processing-7e5a763d3236.* χ.χ.
26. MLlib, Introduction to. *https://blog.scalac.io/scala-spark-ml.html.* χ.χ.
27. Proc, Data. *https://cloud.google.com/dataproc/docs/.* χ.χ.
28. Projects, Intensive Metrics for the Study of the Evolution of Open Source Projects: Case Studies from Apache Software Foundation. *Santiago Gala-P´erez, Gregorio Robles.* χ.χ.
29. Spark, Memory Management for Spark Memory Management for. *https://www.scads.de/images/Events/3rdSummerSchool/Talks/SparkMemory-Salem.pdf.* χ.χ.
30. Spark, Performance Tuning and Evaluation of Iterative Algorithms in. *Janani Gururam .* χ.χ.
31. sparkMeasure.*https://index.scala-lang.org/lucacanali/sparkmeasure/spark-measure/0.13?target=_2.11.* χ.χ.
32. Springer-Verlag, Dependability: Basic Concepts and Terminology. *J.C. Laprie.* χ.χ.
33. STORAGE, ISOLATION IN CLOUD. *Ji-Yong Shin.* χ.χ.
34. Systems, Adaptive Resource Allocation for Preemptable Jobs in Cloud. *Jiayin Li, Meikang Qiu, Jian-Wei Niu, Yu Chen, Zhong Ming.* χ.χ.
35. Systems, Modern Operating. *Τανενμπάουμ, Άντριου.* χ.χ.
36. Troubleshooting, On Measuring Apache Spark Workload Metrics for Performance. *https://db-blog.web.cern.ch/blog/luca-canali/2017-03-measuring-apache-spark-workload-metrics-performance-troubleshooting.* χ.χ.
37. Visualization. *http://www.eng.ucy.ac.cy/christos/Courses/ECE325/Lectures/Analysis.pdf.* χ.χ.
38. Visualization, Understanding your Apache Spark Application Through. *https://databricks.com/blog/2015/06/22/understanding-your-spark-application-through-visualization.html.* χ.χ.
39. Αλγορίθμων, Ανάλυση. *http://repfiles.kallipos.gr/html_books/4410/Ch1.html#S3.* χ.χ.
40. αλγορίθμων, Ανάλυση. *http://www.cs.uoi.gr/~loukas/courses/Data_Structures/index.files/2-Algorithm-Analysis.pdf.* χ.χ.
41. Αλγορίθμων, Μια Εύπεπτη Εισαγωγή στην Ανάλυση Πολυπλοκότητας. *https://discrete.gr/complexity/?el.* χ.χ.
42. Αλγορίθμων, Σχεδίαση και Ανάλυση. *Κωνσταντίνος Τσίχλας, Ιωάννης Μανωλόπουλος, Αναστάσιος Γούναρης.* χ.χ.
43. Έκδοση, Τεχνητή Νοημοσύνη - Β'. *Ι. Βλαχάβας, Π. Κεφαλάς, Ν. Βασιλειάδης, Φ. Κόκκορας, Η. Σακελλαρίου .* χ.χ.
44. Πολυπλοκότητα,Αλγόριθμοι *https://www.cs.ucy.ac.cy/~mavronic/Classes/cs232/Notes/notes1.pdf.* χ.χ.

Παραρτήματα

Accuracy Table

Παρακάτω παρατίθεται ο πίνακας αποτελεσμάτων της μετρικής του Accuracy για κάθε έναν από τους αλγορίθμους σε κάθε σενάριο. Το μεγεθος του test set που ορίστηκε σε σχέση με το υπόλοιπο dataset ήταν το 70%.

	Algorithms	Accuracy
Local 1 master 2workers (Scenario 1st)	LR	
	NB	
	SVM	
GCP 1 master 2workers (Scenario 2nd)	LR	64%
	NB	57%
	SVM	60%
GCP 1 master 3workers (Scenario 3rd)	LR	64%
	NB	57%
	SVM	61%

Accuracy Metric Result for every algorithm

Spark Components Summary Table

Στον από κάτω πίνακα έχουν στοιχειοθετηθεί τα components με τα οποία το Spark χώρισε τα δεδομένα έτσι ώστε να γίνει η ταχύτερη δυνατή επεξεργασία τους. (Κεφάλαιο 4.2.9)

	Algorithms	Jobs	Stages	Executors
Local 1 master 2workers (Scenario 1st)	LR	20	42	2
	NB	7	12	2
	SVM	28	55	2
GCP 1 master 2workers (Scenario 2nd)	LR	21	44	4
	NB	7	12	2
	SVM	87	177	2
GCP 1 master 3workers (Scenario 3rd)	LR	26	45	5
	NB	6	12	5
	SVM	60	127	6

Spark App components for all case scenarios

Analysing Spark Components

"Shuffling" σημαίνει ανακατανομή δεδομένων μεταξύ πολλαπλών stages του Spark. Το "Shuffle Write" είναι το άθροισμα όλων των γραπτών σειριοποιημένων δεδομένων σε όλους τους executors πριν από τη μετάδοση (συνήθως στο τέλος ενός stage) και το "Shuffle Read" σημαίνει το άθροισμα των σειριοποιημένων δεδομένων σε όλους τους εκτελεστές στην αρχή ενός stage. Παρακάτω παραθέτουμε τα γραφήματα του shuffling και το σύνολο των Jobs, Stages και Executors ανά case scenario όπως αναφέρθηκαν στο Κεφάλαιο 7.

Case Scenario 1st Components:

Logistic Regression

Ο παρακάτω πίνακας περιέχει τα "Jobs" που δημιούργησε το Spark για την επεξεργασία των δεδωμένων με σειρά εκτέλεσης που περιγράφεται από το "ID". Για κάθε "job" έχουν καταγραφεί το description του που περιγάφει ποιο ακριβώς είναι το "job" που εκτελείται, καθώς και ο χρόνος εκτέλεσής του.

Logistic Regression: Jobs

	Job		
	ID	**Description**	**Duration**
	0	csv at NativeMethodAccessorImpl.java:0	4s
	1	treeAggregate at LogisticRegression.scala:520	2,2min
	2	treeAggregate at RDDLossFunction.scala:61	7s
	3	treeAggregate at RDDLossFunction.scala:61	4s
	4	treeAggregate at RDDLossFunction.scala:61	3s
	5	treeAggregate at RDDLossFunction.scala:61	3s
	6	treeAggregate at RDDLossFunction.scala:61	3s
	7	treeAggregate at RDDLossFunction.scala:61	3s
	8	treeAggregate at RDDLossFunction.scala:61	3s
	9	treeAggregate at RDDLossFunction.scala:61	3s

	10	treeAggregate at RDDLossFunction.scala:61	3s
	11	treeAggregate at RDDLossFunction.scala:61	3s
	12	treeAggregate at RDDLossFunction.scala:61	3s
	13	treeAggregate at RDDLossFunction.scala:61	3s
	14	treeAggregate at RDDLossFunction.scala:61	3s
	15	treeAggregate at RDDLossFunction.scala:61	3s
	16	sortByKey at BinaryClassificationMetrics.scala:155	1,8min
	17	collect at BinaryClassificationMetrics.scala:192	15s
	18	collect at SlidingRDD.scala:81	9s
	19	aggregate at AreaUnderCurve.scala:45	2s

Σε συνέχεια από τον από πάνω πίνακα, παρατίθενται τα "Stages". Σε κάθε "Job" που η αρίθμησή του είναι "wrapped text", αντιστοιχούν 2 stages. Στον πίνακα από κάτω εμφανίζονται κατά σειρά εκτέλεσης τα "Stages", όπου για κάθε "stage" έχουμε το "description", το χρόνο εκτέλεσης του, καθώς και τις πληροφορίες "input", "output", "suffle read", "suffle write".

Logistic Regression: Stages

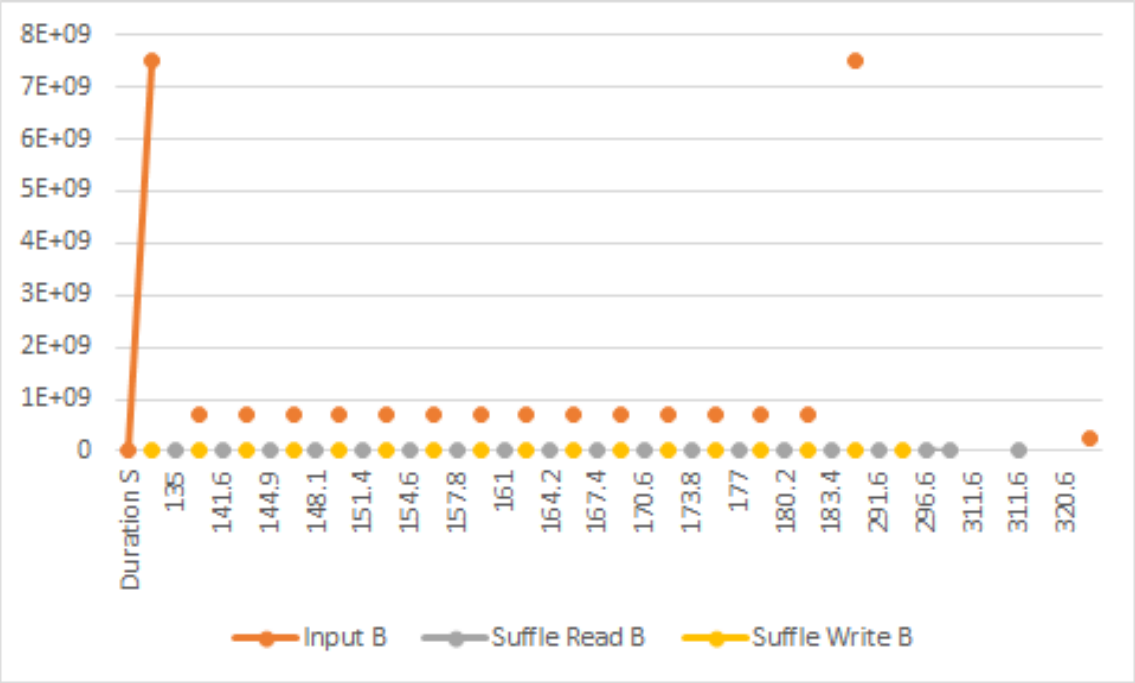
Stages							
ID	**Description**	**No Tasks**	**Duration**	**Input**	**Output**	**Suffle Read**	**Suffle Write**
0	csv at NativeMethodAccessorImpl.java:0	1	3s	64KB			
1	treeAggregate at LogisticRegression.scala:520	60	2,2 min	7,5 GB			32,4KB
2	treeAggregate at LogisticRegression.scala:520	7	0,6 s			32,4KB	
3	treeAggregate at RDDLossFunction.scala:61	60	6s	704MB			18,9KB
4	treeAggregate at RDDLossFunction.scala:61	7	0,3s			18,9KB	
5	treeAggregate at RDDLossFunction.scala:61	60	3s	704MB			18,9KB
6	treeAggregate at RDDLossFunction.scala:61	7	0,2w			18,9KB	
7	treeAggregate at RDDLossFunction.scala:61	60	3s	704MB			18,9KB

8	treeAggregate at RDDLossFunction.scala:61	7	0,3s			18,9KB	
9	treeAggregate at RDDLossFunction.scala:61	60	3s	704MB			18,9KB
10	treeAggregate at RDDLossFunction.scala:61	7	0,2s			18,9KB	
11	treeAggregate at RDDLossFunction.scala:61	60	3s	704MB			18,9KB
12	treeAggregate at RDDLossFunction.scala:61	7	0,2s			18,9KB	
13	treeAggregate at RDDLossFunction.scala:61	60	3s	704MB			18,9KB
14	treeAggregate at RDDLossFunction.scala:61	7	0,2s			18,9KB	
15	treeAggregate at RDDLossFunction.scala:61	60	3s	704MB			18,9KB
16	treeAggregate at RDDLossFunction.scala:61	7	0,2s			18,9KB	
17	treeAggregate at RDDLossFunction.scala:61	60	3s	704MB			18,9KB
18	treeAggregate at RDDLossFunction.scala:61	7	0,2s			18,9KB	
19	treeAggregate at RDDLossFunction.scala:61	60	3s	704MB			18,9KB
20	treeAggregate at RDDLossFunction.scala:61	7	0,2s			18,9KB	
21	treeAggregate at RDDLossFunction.scala:61	60	3s	704MB			18,9KB
22	treeAggregate at RDDLossFunction.scala:61	7	0,2s			18,9KB	
23	treeAggregate at RDDLossFunction.scala:61	60	3s	704MB			18,9KB
24	treeAggregate at RDDLossFunction.scala:61	7	0,2s			18,9KB	
25	treeAggregate at RDDLossFunction.scala:61	60	3s	704MB			18,9KB
26	treeAggregate at RDDLossFunction.scala:61	7	0,2s			18,9KB	
27	treeAggregate at RDDLossFunction.scala:61	60	3s	704MB			18,9KB
28	treeAggregate at RDDLossFunction.scala:61	7	0,2s			18,9KB	
29	treeAggregate at RDDLossFunction.scala:61	60	3s	704MB			18,9KB
30	treeAggregate at RDDLossFunction.scala:61	7	0,2s			18,9KB	
31	map at BinaryClassificationEvaluator.scala:81	60	1,8min	7,5GB			39MB
32	sortByKey at BinaryClassificationMetrics.scala:155	60	5s			39MB	
33			skipp				35,7MB
34	combineByKey at BinaryClassificationMetrics.scala:151	60	8s			39MB	
35	collect at BinaryClassificationMetrics.scala:192	60	7s			35,7MB	
36			skipp				
37			skipp				
38	collect at SlidingRDD.scala:81	62	9s			35,7MB	
39			skipp				
40			skipp				
41	aggregate at AreaUnderCurve.scala:45	61	2s	262,11MB			

Τα “stages” των οποίων το “duration” γράφει “skipped”, είναι εκείνα τα οποία παραλείφθηκαν από το Spark για την γρηγορότερη εκτέλεση του αλγορίθμου.

Τέλος παρατίθεται μία πρώτη προσπάθεια να απεικονιστούν τα χαρακτηριστικά των “stages” (input, read, write) σε σχέση με το συνολικό χρόνο εκτέλεσης του αλγορίθμου.

Logistic Regression: Stages Diagram



Input/Output & shuffling byte/sec 7.1.1

Δυστυχώς λόγο ακραίων τιμών του “input”, δεν μπορεί να εξαχθεί συμπέρασμα από την παρατήρηση της οπτικοποίησης των αποτελεσμάτων.

Naïve Bayes

Όπως περιγράφηκε στη Λογιστική Παλινδρόμηση, με τον ίδιο τρόπο παρατίθενται οι πίνακες των "Jobs" και των "Stages"

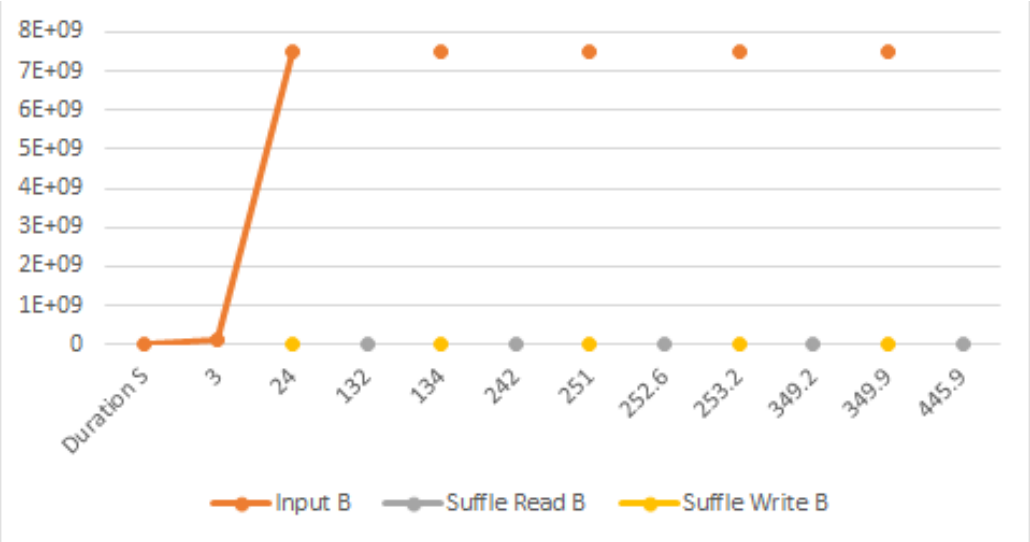
Naive Bayes: Jobs

Job		
ID	**Description**	**Duration**
0	csv at NativeMethodAccessorImpl.java:0	4s
1	head at NaiveBayes.scala:156	21s
2	collect at NaiveBayes.scala:176	1,8min
3	collectAsMap at MulticlassMetrics.scala:48	1,8min
4	countByValue at MulticlassMetrics.scala:42	1,6min
5	collectAsMap at MulticlassMetrics.scala:48	1,6min
6	countByValue at MulticlassMetrics.scala:42	1,6min

Naive Bayes: Stages

Stages							
ID	Description	No Tasks	Duration	Input	Output	Suffle Read	Suffle Write
0	csv at NativeMethodAccessorImpl.java:0	1	3s	64.0 KB			
1	csv at NativeMethodAccessorImpl.java:0	1	21s	128.1 MB			
2	map at NaiveBayes.scala:164	60	1,8min	7.5 GB			14.6 KB
3	collect at NaiveBayes.scala:176	60	2s			14.6 KB	
4	map at MulticlassMetrics.scala:45	60	1,8min	7.5 GB			6.4 KB
5	collectAsMap at MulticlassMetrics.scala:48	60	0,9s			6.4 KB	
6	countByValue at MulticlassMetrics.scala:42	60	1,6	7.5 GB			6.4 KB
7	countByValue at MulticlassMetrics.scala:42	60	0,6s			6.4 KB	
8	map at MulticlassMetrics.scala:45	60	1,6min	7.5 GB			6.4 KB
9	collectAsMap at MulticlassMetrics.scala:48	60	0,7s			6.4 KB	
10	countByValue at MulticlassMetrics.scala:42	60	1,6min	7.5 GB			6.4 KB
11	countByValue at MulticlassMetrics.scala:42	60	0,5s			6.4 KB	

Naive Bayes: Stages Diagram



Input/Output & shuffling byte/sec 7.1.2

SVM

SVM: Jobs 1

Job		
ID	**Description**	**Duration**
0	csv at NativeMethodAccessorImpl.java:0	4s
1	treeAggregate at LinearSVC.scala:190	2,1min
2	treeAggregate at RDDLossFunction.scala:61	1,7min
3	treeAggregate at RDDLossFunction.scala:61	1,7min
4	treeAggregate at RDDLossFunction.scala:61	1,7min
5	treeAggregate at RDDLossFunction.scala:61	1,7min
6		

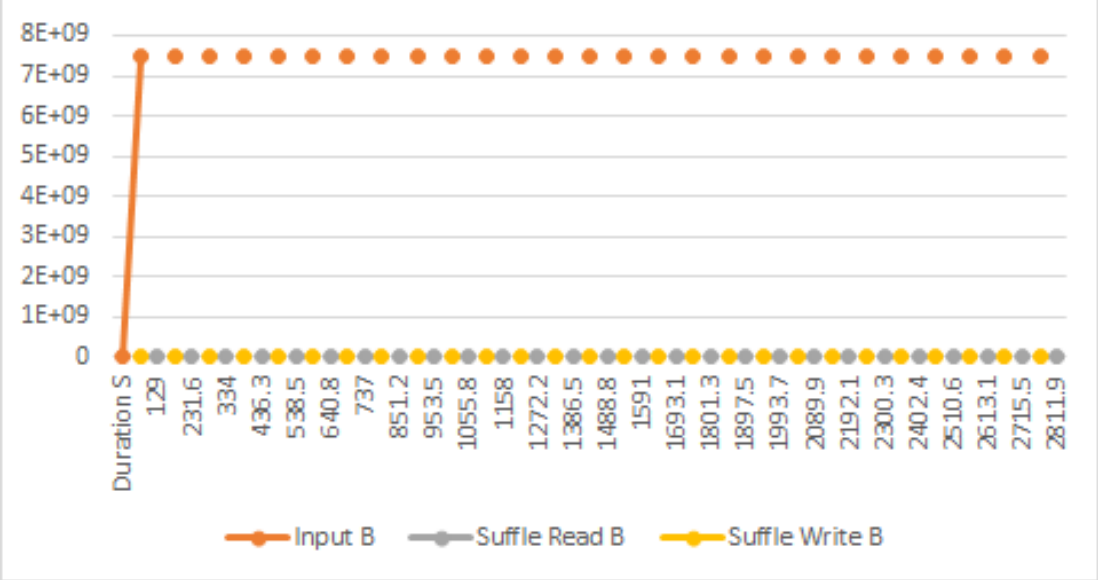
	treeAggregate at RDDLossFunction.scala:61	1,7min
7	treeAggregate at RDDLossFunction.scala:61	1,6min
8	treeAggregate at RDDLossFunction.scala:61	1,9min
9	treeAggregate at RDDLossFunction.scala:61	1,7min
10	treeAggregate at RDDLossFunction.scala:61	1,7min
11	treeAggregate at RDDLossFunction.scala:61	1,7min
12	treeAggregate at RDDLossFunction.scala:61	1,9min
13	treeAggregate at RDDLossFunction.scala:61	1,9min
14	treeAggregate at RDDLossFunction.scala:61	1,7min
15	treeAggregate at RDDLossFunction.scala:61	1,7min
16	treeAggregate at RDDLossFunction.scala:61	1,7min
17	treeAggregate at RDDLossFunction.scala:61	1,8min
18	treeAggregate at RDDLossFunction.scala:61	1,6min
19	treeAggregate at RDDLossFunction.scala:61	1,6min
20	treeAggregate at RDDLossFunction.scala:61	1,6min
21	treeAggregate at RDDLossFunction.scala:61	1,7min
22	treeAggregate at RDDLossFunction.scala:61	1,8min
23	treeAggregate at RDDLossFunction.scala:61	1,7min
24	collectAsMap at MulticlassMetrics.scala:48	1,8min
25	countByValue at MulticlassMetrics.scala:42	1,7min
26	collectAsMap at MulticlassMetrics.scala:48	1,7min
27	countByValue at MulticlassMetrics.scala:42	1,6min

SVM: Stages

Stages							
ID	Description	No Tasks	Duration	Input	Output	Suffle Read	Suffle Write
0	csv at NativeMethodAccessorImpl.java:0	1	3s	64.0 KB			
1	treeAggregate at LinearSVC.scala:190	60	2,1min	7.5 GB			32.4 KB
2	treeAggregate at LinearSVC.scala:190	7	0,6s			32.4 KB	
3	treeAggregate at RDDLossFunction.scala:61	60	1,7min	7.5 GB			18.5 KB
4	treeAggregate at RDDLossFunction.scala:62	7	0,4s			18.5 KB	
5	treeAggregate at RDDLossFunction.scala:63	60	1,7min	7.5 GB			18.5 KB
6	treeAggregate at RDDLossFunction.scala:64	7	0,3s			18.5 KB	
7	treeAggregate at RDDLossFunction.scala:65	60	1,7min	7.5 GB			18.5 KB
8	treeAggregate at RDDLossFunction.scala:66	7	0,2s			18.5 KB	
9	treeAggregate at RDDLossFunction.scala:67	60	1,7min	7.5 GB			18.5 KB
10	treeAggregate at RDDLossFunction.scala:68	7	0,3s			18.5 KB	
11	treeAggregate at RDDLossFunction.scala:69	60	1,7min	7.5 GB			18.5 KB
12	treeAggregate at RDDLossFunction.scala:70	7	0,2s			18.5 KB	
13	treeAggregate at RDDLossFunction.scala:71	60	1,6min	7.5 GB			18.5 KB
14	treeAggregate at RDDLossFunction.scala:72	7	0,2s			18.5 KB	
15	treeAggregate at RDDLossFunction.scala:73	60	1,9min	7.5 GB			18.5 KB
16	treeAggregate at RDDLossFunction.scala:74	7	0,3s			18.5 KB	
17	treeAggregate at RDDLossFunction.scala:75	60	1,7min	7.5 GB			18.5 KB
18	treeAggregate at RDDLossFunction.scala:76	7	0,3s			18.5 KB	
19	treeAggregate at RDDLossFunction.scala:77	60	1,7min	7.5 GB			18.5 KB
20	treeAggregate at RDDLossFunction.scala:78	7	0,2s			18.5 KB	
21	treeAggregate at RDDLossFunction.scala:79	60	1,7min	7.5 GB			18.5 KB
22	treeAggregate at RDDLossFunction.scala:80	7	0,2s			18.5 KB	
23	treeAggregate at RDDLossFunction.scala:81	60	1,9min	7.5 GB			18.5 KB
24	treeAggregate at RDDLossFunction.scala:82	7	0,3s			18.5 KB	
25	treeAggregate at RDDLossFunction.scala:83	60	1,9min	7.5 GB			18.5 KB
26	treeAggregate at RDDLossFunction.scala:84	7	0,3s			18.5 KB	
27	treeAggregate at RDDLossFunction.scala:85	60	1,7min	7.5 GB			18.5 KB
28	treeAggregate at RDDLossFunction.scala:86	7	0,2s			18.5 KB	
29	treeAggregate at RDDLossFunction.scala:87	60	1,7min	7.5 GB			18.5 KB
30	treeAggregate at RDDLossFunction.scala:88	7	0,1s			18.5 KB	
31	treeAggregate at RDDLossFunction.scala:89	60	1,7min	7.5 GB			18.5 KB
32	treeAggregate at RDDLossFunction.scala:90	7	0,2s			18.5 KB	
33	treeAggregate at RDDLossFunction.scala:91	60	1,8min	7.5 GB			18.5 KB
34	treeAggregate at RDDLossFunction.scala:92	7	0,2s			18.5 KB	
35	treeAggregate at RDDLossFunction.scala:93	60	1,6min	7.5 GB			18.5 KB
36	treeAggregate at RDDLossFunction.scala:94	7	0,2s			18.5 KB	
37	treeAggregate at RDDLossFunction.scala:95	60	1,6min	7.5 GB			18.5 KB

38	treeAggregate at RDDLossFunction.scala:96	7	0,2s			18.5 KB	
39	treeAggregate at RDDLossFunction.scala:97	60	1,6min	7.5 GB			18.5 KB
40	treeAggregate at RDDLossFunction.scala:98	7	0,2s			18.5 KB	
41	treeAggregate at RDDLossFunction.scala:99	60	1,7min	7.5 GB			18.6 KB
42	treeAggregate at RDDLossFunction.scala:100	7	0,2s			18.6 KB	
43	treeAggregate at RDDLossFunction.scala:101	60	1,8min	7.5 GB			18.6 KB
44	treeAggregate at RDDLossFunction.scala:102	7	0,1s			18.6 KB	
45	treeAggregate at RDDLossFunction.scala:103	60	1,7min	7.5 GB			18.6 KB
46	treeAggregate at RDDLossFunction.scala:104	7	0,2s			18.6 KB	
47	map at MulticlassMetrics.scala:45	60	1,8min	7.5 GB			6.4 KB
48	collectAsMap at MulticlassMetrics.scala:48	60	0,5s			6.4 KB	
49	countByValue at MulticlassMetrics.scala:42	60	1,7min	7.5 GB			6.4 KB
50	countByValue at MulticlassMetrics.scala:42	60	0,4s			6.4 KB	
51	map at MulticlassMetrics.scala:45	60	1,7min	7.5 GB			6.4 KB
52	collectAsMap at MulticlassMetrics.scala:48	60	0,4s			6.4 KB	
53	countByValue at MulticlassMetrics.scala:42	60	1,6min	7.5 GB			6.4 KB
54	countByValue at MulticlassMetrics.scala:42	60	0,5s			6.4 KB	

SVM: Stages Diagram 1



Input/Output & shuffling byte/sec 7.1.3

Case Scenario 2nd Components:

Logistic Regression:

Logistic Regression: Jobs

Job		
ID	Description	Duration
0	csv at NativeMethodAccessorImp	6s
1	describe at NativeMethodAccessorImpl.java:0	5.0 min
2	treeAggregate at LogisticRegression.scala:520	2.8 min
3	treeAggregate at RDDLossFunction.scala:48	3 s
4	treeAggregate at RDDLossFunction.scala:49	2 s
5	treeAggregate at RDDLossFunction.scala:50	2 s
6	treeAggregate at RDDLossFunction.scala:51	2 s
7	treeAggregate at RDDLossFunction.scala:52	2 s
8	treeAggregate at RDDLossFunction.scala:53	2 s
9	treeAggregate at RDDLossFunction.scala:54	2 s
10	treeAggregate at RDDLossFunction.scala:55	2 s
11	treeAggregate at RDDLossFunction.scala:56	2 s
12	treeAggregate at RDDLossFunction.scala:57	2 s
13	treeAggregate at RDDLossFunction.scala:58	2 s
14	treeAggregate at RDDLossFunction.scala:59	2 s
15	treeAggregate at RDDLossFunction.scala:60	2 s

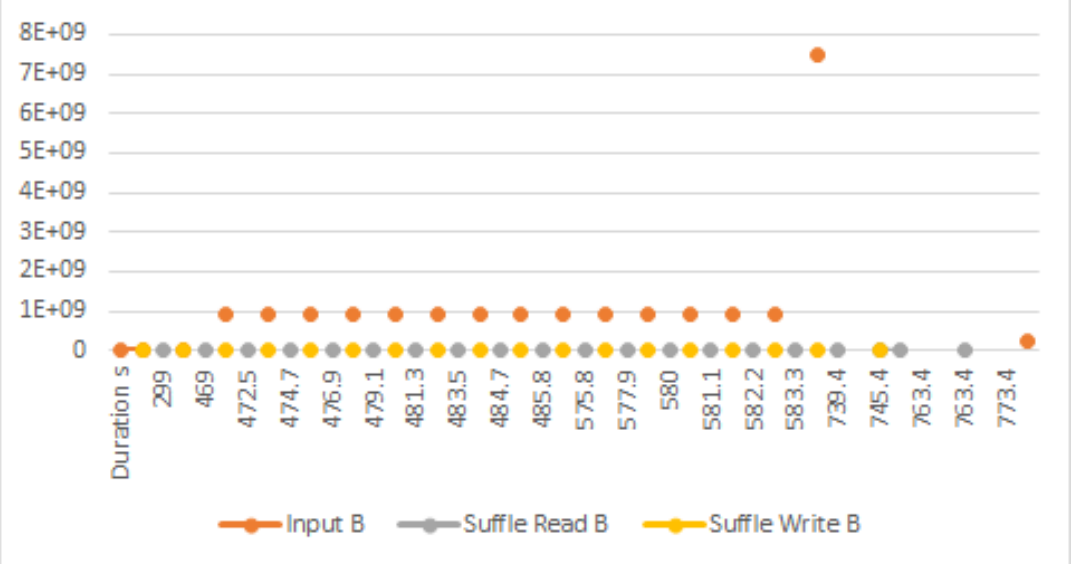
16	treeAggregate at RDDLossFunction.scala:61	2 s
17	sortByKey at BinaryClassificationMetrics.scala:155	2.7 min
18	collect at BinaryClassificationMetrics.scala:192	18 s
19	collect at SlidingRDD.scala:81	10 s
20	aggregate at AreaUnderCurve.scala:4	2 s

Logistic Regression: Stages

Stages							
ID	**Description**	**No Tasks**	**Duration**	**Input**	**Output**	**Suffle Read**	**Suffle Write**
0	csv at NativeMethodAccessorImpl.java:0	1/1	5 s	64.0 KB			
1	describe at NativeMethodAccessorImpl.java:0	60/60	4.9 m	7.5 GB			70.9 KB
2	describe at NativeMethodAccessorImpl.java:0	1/1	2 s			70.9 KB	
3	treeAggregate at LogisticRegression.scala:520	60/60	2.8 min	7.5 GB			63.8 KB
4	treeAggregate at LogisticRegression.scala:520	7/7	0.5 s			63.8 KB	
5	treeAggregate at RDDLossFunction.scala:61	60/60	3 s	910.2 MB			61.3 KB
6	treeAggregate at RDDLossFunction.scala:61	7/7	0.2 s			61.3 KB	
7	treeAggregate at RDDLossFunction.scala:61	60/60	2 s	910.2 MB			61.3 KB
8	treeAggregate at RDDLossFunction.scala:61	7/7	0.2 s			61.3 KB	
9	treeAggregate at RDDLossFunction.scala:61	60/60	2 s	910.2 MB			61.3 KB
10	treeAggregate at RDDLossFunction.scala:61	7/7	0.2 s			61.3 KB	
11	treeAggregate at RDDLossFunction.scala:61	60/60	2 s	910.2 MB			61.3 KB
12	treeAggregate at RDDLossFunction.scala:61	7/7	0.2 s			61.3 KB	
13	treeAggregate at RDDLossFunction.scala:61	60/60	2 s	910.2 MB			61.3 KB
14	treeAggregate at RDDLossFunction.scala:61	7/7	0.2 s			61.3 KB	
15	treeAggregate at RDDLossFunction.scala:61	60/60	2 s	910.2 MB			61.3 KB
16	treeAggregate at RDDLossFunction.scala:61	7/7	0.2 s			61.3 KB	
17	treeAggregate at RDDLossFunction.scala:61	60/60	1 s	910.2 MB			61.2 KB
18	treeAggregate at RDDLossFunction.scala:61	7/7	0.1 s			61.2 KB	
19	treeAggregate at RDDLossFunction.scala:61	60/60	1 s	910.2 MB			61.2 KB
20	treeAggregate at RDDLossFunction.scala:61	7/7	88 ms			61.2 KB	
21	treeAggregate at RDDLossFunction.scala:61	60/60	2 s	910.2 MB			961.2 KB

22	treeAggregate at RDDLossFunction.scala:61	7/7	0.1 s			961.2 KB	
23	treeAggregate at RDDLossFunction.scala:61	60/60	2 s	910.2 MB			61.2 KB
24	treeAggregate at RDDLossFunction.scala:61	7/7	0.1 s			61.2 KB	
25	treeAggregate at RDDLossFunction.scala:61	60/60	2 s	910.2 MB			61.2 KB
26	treeAggregate at RDDLossFunction.scala:61	7/7	0.1 s			61.2 KB	
27	treeAggregate at RDDLossFunction.scala:61	60/60	1 s	910.2 MB			61.2 KB
28	treeAggregate at RDDLossFunction.scala:61	7/7	0.1 s			61.2 KB	
29	treeAggregate at RDDLossFunction.scala:61	60/60	1 s	910.2 MB			61.2 KB
30	treeAggregate at RDDLossFunction.scala:61	7/7	0.1 s			61.2 KB	
31	treeAggregate at RDDLossFunction.scala:61	60/60	1 s	910.2 MB			41.4 MB
32	treeAggregate at RDDLossFunction.scala:61	7/7	0.1 s			61.3 KB	
33	map at BinaryClassificationEvaluator.scala:81	60/60	2.6 min	7.5 GB			41.4 MB
34	sortByKey at BinaryClassificationMetrics.scala:155	60/60	6 s			41.4 MB	
35		skip	skip	skip	skip	skip	skip
36	combineByKey at BinaryClassificationMetrics.scala:151	7/7	9 s			41.4 MB	37.5 MB
37	collect at BinaryClassificationMetrics.scala:192	60/60	9 s			37.5 MB	
38		skip	skip	skip	skip	skip	skip
39		skip	skip	skip	skip	skip	skip
40	collect at SlidingRDD.scala:81	62/62	10 s			37.5 MB	
41		skip	skip	skip	skip	skip	skip
42		skip	skip	skip	skip	skip	skip
43	aggregate at AreaUnderCurve.scala:45	61/61	2 s	259.2 MB			

Logistic Regression: Stages Diagram



Input/Output & shuffling byte/sec 7.2.1

Naïve Bayes

Naive Bayes: Jobs

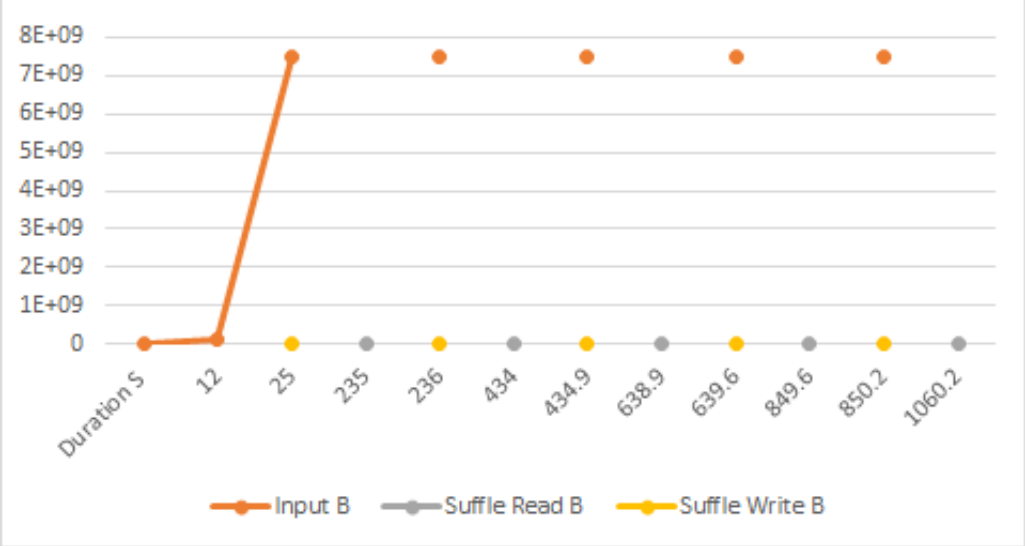
Job ID	Description	Duration
0	csv at NativeMethodAccessorImpl.java:0	3.5min
1	head at NaiveBayes.scala:156	3.5min
2	collect at NaiveBayes.scala:176	3.4min
3	collectAsMap at MulticlassMetrics.scala:48	3.3min
4	countByValue at MulticlassMetrics.scala:42	3.5min
5	collectAsMap at MulticlassMetrics.scala:48	13s
6	countByValue at MulticlassMetrics.scala:42	12s

Naive Bayes: Stages

Stages ID	Description	No Tasks	Duration	Input	Output	Suffle Read	Suffle Write
0	head at NaiveBayes.scala:156	1/1	12s	64.0KB			
1	head at NaiveBayes.scala:156	1/1	13s	128.1MB			
2	map at NaiveBayes.scala:164	60/60	3.5min	7.5GB			44.9KB
3	collect at NaiveBayes.scala:176	60/60	1s			44.9KB	
4	map at MulticlassMetrics.scala:45	60/60	3.3m	7.5GB			6.4KB
5	collectAsMap at MulticlassMetrics.scala:48	60/60	0.9s			6.4KB	
6	countByValue at MulticlassMetrics.scala:42	60/60	3.4min	7.5GB			6.4KB
7	countByValue at MulticlassMetrics.scala:42	60/60	0.7s			6.4KB	
8	map at MulticlassMetrics.scala:45	60/60	3.5min	7.5GB			6.4KB

9	collectAsMap at MulticlassMetrics.scala:48	60/60	0.6s			6.4KB	
10	countByValue at MulticlassMetrics.scala:42	60/60	3.5min	7.5GB			6.4KB
11	countByValue at MulticlassMetrics.scala:42	60/60	0.6s			6.4KB	

Naive Bayes: Stage Diagram



Input/Output & shuffling byte/sec 7.2.2

SVM

SVM: Jobs

Job ID	Description	Duration
0	csv at NativeMetho dAccessorImpl.java:0 csv	6 s
1	treeAggregate at L inearSVC.scala:190 treeA	2.0 min
2	treeAggregate at R DDLossFunction.scala:61	5 s

3	treeAggregate at R DDLossFunction.scala:61	0.7 s
4	treeAggregate at R DDLossFunction.scala:61	0.6 s
5	treeAggregate at R DDLossFunction.scala:61	0.5 s
6	treeAggregate at R DDLossFunction.scala:61	0.4 s
7	treeAggregate at R DDLossFunction.scala:61	0.7 s
8	treeAggregate at R DDLossFunction.scala:61	0.6 s
9	treeAggregate at R DDLossFunction.scala:61	0.5 s
10	treeAggregate at RDDLossFunction.scala:61	0.4 s
11	treeAggregate at RDDLossFunction.scala:61	0.5 s
12	treeAggregate at RDDLossFunction.scala:61	0.4 s
13	treeAggregate at RDDLossFunction.scala:61	0.4 s
14	treeAggregate at RDDLossFunction.scala:61	0.4 s
15	treeAggregate at RDDLossFunction.scala:61	0.4 s
16	treeAggregate at RDDLossFunction.scala:61	0.4 s
17	treeAggregate at RDDLossFunction.scala:61	0.3 s
18	treeAggregate at RDDLossFunction.scala:61	0.5 s
19	treeAggregate at RDDLossFunction.scala:61	0.4 s
20	treeAggregate at RDDLossFunction.scala:61	0.4 s
21	treeAggregate at RDDLossFunction.scala:61	0.4 s
22	treeAggregate at RDDLossFunction.scala:61	0.4 s
23	treeAggregate at RDDLossFunction.scala:61	0.3 s
24	treeAggregate at RDDLossFunction.scala:61	0.4 s

25	treeAggregate at RDDLossFunction.scala:61	0.4 s
26	treeAggregate at RDDLossFunction.scala:61	0.4 s
27	treeAggregate at RDDLossFunction.scala:61	0.3 s
28	treeAggregate at RDDLossFunction.scala:61	0.4 s
29	treeAggregate at RDDLossFunction.scala:61	0.3 s
30	treeAggregate at RDDLossFunction.scala:61	0.3 s
31	treeAggregate at RDDLossFunction.scala:61	0.3 s
32	treeAggregate at RDDLossFunction.scala:61	0.3 s
33	treeAggregate at RDDLossFunction.scala:61	0.3 s
34	treeAggregate at RDDLossFunction.scala:61	0.3 s
35	treeAggregate at RDDLossFunction.scala:61	0.3 s
36	treeAggregate at RDDLossFunction.scala:61	0.3 s
37	treeAggregate at RDDLossFunction.scala:61	0.4 s
38	treeAggregate at RDDLossFunction.scala:61	0.4 s
39	treeAggregate at RDDLossFunction.scala:61	0.3 s
40	treeAggregate at RDDLossFunction.scala:61	0.3 s
41	treeAggregate at RDDLossFunction.scala:61	0.2 s
42	treeAggregate at RDDLossFunction.scala:61	0.4 s
43	treeAggregate at RDDLossFunction.scala:61	0.3 s
44	treeAggregate at RDDLossFunction.scala:61	0.2 s
45	treeAggregate at RDDLossFunction.scala:61	0.3 s
46	treeAggregate at RDDLossFunction.scala:61	0.2 s

47	treeAggregate at RDDLossFunction.scala:61	0.2 s
48	treeAggregate at RDDLossFunction.scala:61	0.3 s
49	treeAggregate at RDDLossFunction.scala:61	0.2 s
50	treeAggregate at RDDLossFunction.scala:61	0.3 s
51	treeAggregate at RDDLossFunction.scala:61	0.3 s
52	treeAggregate at RDDLossFunction.scala:61	0.3 s
53	treeAggregate at RDDLossFunction.scala:61	0.2 s
54	treeAggregate at RDDLossFunction.scala:61	0.2 s
55	treeAggregate at RDDLossFunction.scala:61	0.3 s
56	treeAggregate at RDDLossFunction.scala:61	0.2 s
57	treeAggregate at RDDLossFunction.scala:61	0.2 s
58	treeAggregate at RDDLossFunction.scala:61	0.4 s
59	treeAggregate at RDDLossFunction.scala:61	0.2 s
60	treeAggregate at RDDLossFunction.scala:61	0.2 s
61	treeAggregate at RDDLossFunction.scala:61	0.4 s
62	treeAggregate at RDDLossFunction.scala:61	0.3 s
63	treeAggregate at RDDLossFunction.scala:61	0.3 s
64	treeAggregate at RDDLossFunction.scala:61	0.3 s
65	treeAggregate at RDDLossFunction.scala:61	0.2 s
66	treeAggregate at RDDLossFunction.scala:61	0.2 s
67	treeAggregate at RDDLossFunction.scala:61	0.2 s
68	treeAggregate at RDDLossFunction.scala:61	0.2 s

69	treeAggregate at RDDLossFunction.scala:61	0.2 s
70	treeAggregate at RDDLossFunction.scala:61	0.2 s
71	treeAggregate at RDDLossFunction.scala:61	0.2 s
72	treeAggregate at RDDLossFunction.scala:61	0.2 s
73	treeAggregate at RDDLossFunction.scala:61	0.2 s
74	treeAggregate at RDDLossFunction.scala:61	0.2 s
75	treeAggregate at RDDLossFunction.scala:61	0.2 s
76	treeAggregate at RDDLossFunction.scala:61	0.2 s
77	treeAggregate at RDDLossFunction.scala:61	0.2 s
78	treeAggregate at RDDLossFunction.scala:61	0.3 s
79	treeAggregate at RDDLossFunction.scala:61	0.2 s
80	treeAggregate at RDDLossFunction.scala:61	0.2 s
81	treeAggregate at RDDLossFunction.scala:61	0.2 s
82	treeAggregate at RDDLossFunction.scala:61	0.2 s
83	collectAsMap at MulticlassMetrics.scala:4	1.9 min
84	countByValue at MulticlassMetrics.scala:4	2 s
85	collectAsMap at MulticlassMetrics.scala:4	2.4 min
86	countByValue at MulticlassMetrics.scala:4	

SVM: Stages

Stages							
ID	**Description**	**No Tasks**	**Duration**	**Input**	**Output**	**Suffle Read**	**Suffle Write**
0	rdd at LinearSVC.scala:169	1/1	6s	64.0KB			

1	csv at NativeMethodAccessorImpl.java: 0	60/60	1.9min	7.5GB			74.7MB
2	treeAggregate at LinearSVC.scala:190	1/1	5s			36.1MB	
3	(1 skipped)	(1 skipped)					
4	treeAggregate at RDDLossFunction.scala:61	1/1	5s				
5	(1 skipped)	(1 skipped)					
6	treeAggregate at RDDLossFunction.scala:61	1/1	0.6s				
7	(1 skipped)	(1 skipped)					
8	treeAggregate at RDDLossFunction.scala:62	1/1	0.6s				
9	(1 skipped)	(1 skipped)					
10	treeAggregate at RDDLossFunction.scala:62	1/1	0.5s				
11	(1 skipped)	(1 skipped)					
12	treeAggregate at RDDLossFunction.scala:62	1/1	0.4s				
13	(1 skipped)	(1 skipped)					
14	treeAggregate at RDDLossFunction.scala:62	1/1	0.6s				
15	(1 skipped)	(1 skipped)					
16	treeAggregate at RDDLossFunction.scala:62	1/1	0.6s				
17	(1 skipped)	(1 skipped)					
18	treeAggregate at RDDLossFunction.scala:63	1/1	0.4s				
19	(1 skipped)	(1 skipped)					
20	treeAggregate at RDDLossFunction.scala:63	1/1	0.4s				
21	(1 skipped)	(1 skipped)					
22	treeAggregate at RDDLossFunction.scala:63	1/1	0.4s				
23	(1 skipped)	(1 skipped)					
24	treeAggregate at RDDLossFunction.scala:63	1/1	0.4s				
25	(1 skipped)	(1 skipped)					
26	treeAggregate at RDDLossFunction.scala:63	1/1	0.4s				
27	(1 skipped)	(1 skipped)					
28	treeAggregate at RDDLossFunction.scala:64	1/1	0.3s				
29	(1 skipped)	(1 skipped)					
30	treeAggregate at RDDLossFunction.scala:64	1/1	0.4s				
31	(1 skipped)	(1 skipped)					
32	treeAggregate at RDDLossFunction.scala:64	1/1	0.3s				
33	(1 skipped)	(1 skipped)					
34	treeAggregate at RDDLossFunction.scala:64	1/1	0.3s				
35	(1 skipped)	(1 skipped)					
36	treeAggregate at RDDLossFunction.scala:64	1/1	0.4s				
37	(1 skipped)	(1 skipped)					
38	treeAggregate at RDDLossFunction.scala:65	1/1	0.4s				

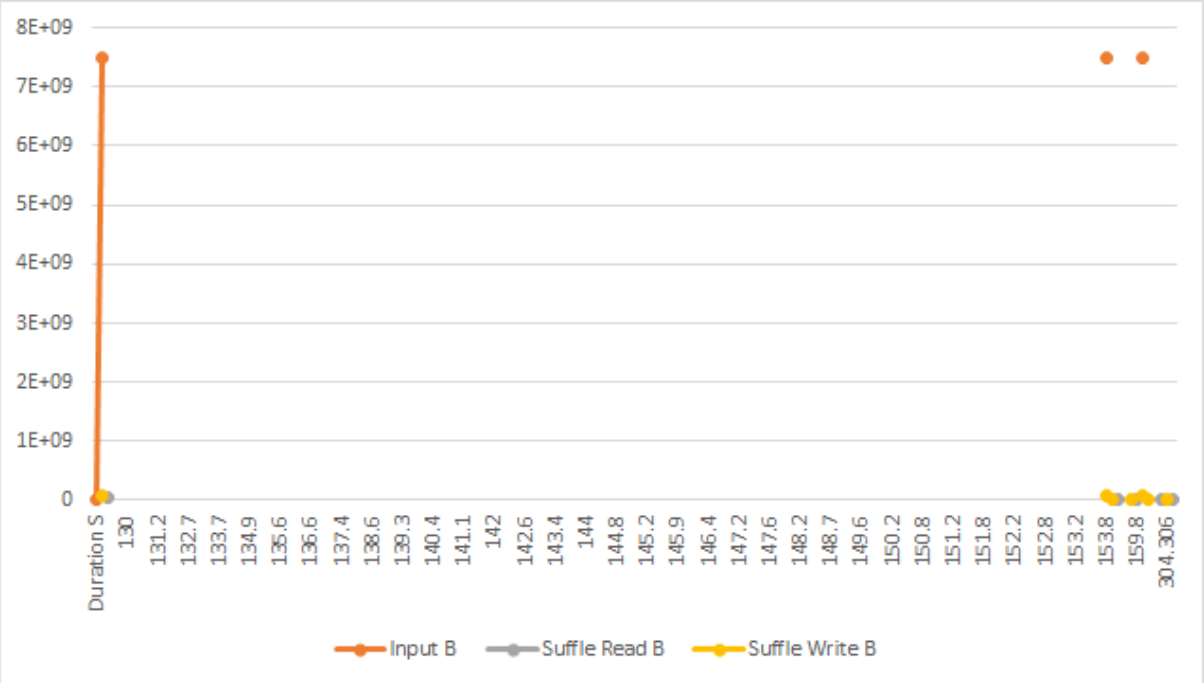
39	(1 skipped)	(1 skipped)					
40	treeAggregate at RDDLossFunction.scala:65	1/1	0.4s				
41	(1 skipped)	(1 skipped)					
42	treeAggregate at RDDLossFunction.scala:65	1/1	0.4s				
43	(1 skipped)	(1 skipped)					
44	treeAggregate at RDDLossFunction.scala:65	1/1	0.4s				
45	(1 skipped)	(1 skipped)					
46	treeAggregate at RDDLossFunction.scala:65	1/1	0.3s				
47	(1 skipped)	(1 skipped)					
48	treeAggregate at RDDLossFunction.scala:66	1/1	0.4s				
49	(1 skipped)	(1 skipped)					
50	treeAggregate at RDDLossFunction.scala:66	1/1	0.4s				
51	(1 skipped)	(1 skipped)					
52	treeAggregate at RDDLossFunction.scala:66	1/1	0.4s				
53	(1 skipped)	(1 skipped)					
54	treeAggregate at RDDLossFunction.scala:66	1/1	0.3s				
55	(1 skipped)	(1 skipped)					
56	treeAggregate at RDDLossFunction.scala:66	1/1	0.4s				
57	(1 skipped)	(1 skipped)					
58	treeAggregate at RDDLossFunction.scala:67	1/1	0.3s				
59	(1 skipped)	(1 skipped)					
60	treeAggregate at RDDLossFunction.scala:67	1/1	0.3s				
61	(1 skipped)	(1 skipped)					
62	treeAggregate at RDDLossFunction.scala:67	1/1	0.3s				
63	(1 skipped)	(1 skipped)					
64	treeAggregate at RDDLossFunction.scala:67	1/1	0.3s				
65	(1 skipped)	(1 skipped)					
66	treeAggregate at RDDLossFunction.scala:67	1/1	0.3s				
67	(1 skipped)	(1 skipped)					
68	treeAggregate at RDDLossFunction.scala:68	1/1	0.3s				
69	(1 skipped)	(1 skipped)					
70	treeAggregate at RDDLossFunction.scala:68	1/1	0.3s				
71	(1 skipped)	(1 skipped)					
72	treeAggregate at RDDLossFunction.scala:68	1/1	0.2s				
73	(1 skipped)	(1 skipped)					
74	treeAggregate at RDDLossFunction.scala:68	1/1	0.3s				
75	(1 skipped)	(1 skipped)					
76	treeAggregate at RDDLossFunction.scala:68	1/1	0.3s				

77	(1 skipped)	(1 skipped)					
78	treeAggregate at RDDLossFunction.scala:69	1/1	0.3s				
79	(1 skipped)	(1 skipped)					
80	treeAggregate at RDDLossFunction.scala:69	1/1	0.3s				
81	(1 skipped)	(1 skipped)					
82	treeAggregate at RDDLossFunction.scala:69	1/1	0.2s				
83	(1 skipped)	(1 skipped)					
84	treeAggregate at RDDLossFunction.scala:69	1/1	0.3s				
85	(1 skipped)	(1 skipped)					
86	treeAggregate at RDDLossFunction.scala:69	1/1	0.2s				
87	(1 skipped)	(1 skipped)					
88	treeAggregate at RDDLossFunction.scala:70	1/1	0.2s				
89	(1 skipped)	(1 skipped)					
90	treeAggregate at RDDLossFunction.scala:70	1/1	0.3s				
91	(1 skipped)	(1 skipped)					
92	treeAggregate at RDDLossFunction.scala:70	1/1	0.2s				
93	(1 skipped)	(1 skipped)					
94	treeAggregate at RDDLossFunction.scala:70	1/1	0.2s				
95	(1 skipped)	(1 skipped)					
96	treeAggregate at RDDLossFunction.scala:70	1/1	0.3s				
97	(1 skipped)	(1 skipped)					
98	treeAggregate at RDDLossFunction.scala:71	1/1	0.2s				
99	(1 skipped)	(1 skipped)					
100	treeAggregate at RDDLossFunction.scala:71	1/1	0.2s				
101	(1 skipped)	(1 skipped)					
102	treeAggregate at RDDLossFunction.scala:71	1/1	0.3s				
103	(1 skipped)	(1 skipped)					
104	treeAggregate at RDDLossFunction.scala:71	1/1	0.3s				
105	(1 skipped)	(1 skipped)					
106	treeAggregate at RDDLossFunction.scala:71	1/1	0.2s				
107	(1 skipped)	(1 skipped)					
108	treeAggregate at RDDLossFunction.scala:72	1/1	0.2s				
109	(1 skipped)	(1 skipped)					
110	treeAggregate at RDDLossFunction.scala:72	1/1	0.2s				
111	(1 skipped)	(1 skipped)					
112	treeAggregate at RDDLossFunction.scala:72	1/1	0.2s				
113	(1 skipped)	(1 skipped)					
114	treeAggregate at RDDLossFunction.scala:72	1/1	0.2s				

115	(1 skipped)	(1 skipped)					
116	treeAggregate at RDDLossFunction.scala:72	1/1	0.3s				
117	(1 skipped)	(1 skipped)					
118	treeAggregate at RDDLossFunction.scala:73	1/1	0.2s				
119	(1 skipped)	(1 skipped)					
120	treeAggregate at RDDLossFunction.scala:73	1/1	0.2s				
121	(1 skipped)	(1 skipped)					
122	treeAggregate at RDDLossFunction.scala:73	1/1	0.4s				
123	(1 skipped)	(1 skipped)					
124	treeAggregate at RDDLossFunction.scala:73	1/1	0.3s				
125	(1 skipped)	(1 skipped)					
126	treeAggregate at RDDLossFunction.scala:73	1/1	0.3s				
127	(1 skipped)	(1 skipped)					
128	treeAggregate at RDDLossFunction.scala:74	1/1	0.3s				
129	(1 skipped)	(1 skipped)					
130	treeAggregate at RDDLossFunction.scala:74	1/1	0.2s				
131	(1 skipped)	(1 skipped)					
132	treeAggregate at RDDLossFunction.scala:74	1/1	0.2s				
133	(1 skipped)	(1 skipped)					
134	treeAggregate at RDDLossFunction.scala:74	1/1	0.2s				
135	(1 skipped)	(1 skipped)					
136	treeAggregate at RDDLossFunction.scala:74	1/1	0.2s				
137	(1 skipped)	(1 skipped)					
138	treeAggregate at RDDLossFunction.scala:75	1/1	0.2s				
139	(1 skipped)	(1 skipped)					
140	treeAggregate at RDDLossFunction.scala:75	1/1	0.2s				
141	(1 skipped)	(1 skipped)					
142	treeAggregate at RDDLossFunction.scala:75	1/1	0.2s				
143	(1 skipped)	(1 skipped)					
144	treeAggregate at RDDLossFunction.scala:75	1/1	0.2s				
145	(1 skipped)	(1 skipped)					
146	treeAggregate at RDDLossFunction.scala:75	1/1	0.2s				
147	(1 skipped)	(1 skipped)					
148	treeAggregate at RDDLossFunction.scala:76	1/1	0.2s				
149	(1 skipped)	(1 skipped)					
150	treeAggregate at RDDLossFunction.scala:76	1/1	0.2s				
151	(1 skipped)	(1 skipped)					
152	treeAggregate at RDDLossFunction.scala:76	1/1	0.2s				

153	(1 skipped)	(1 skipped)					
154	treeAggregate at RDDLossFunction.scala:76	1/1	0.2s				
155	(1 skipped)	(1 skipped)					
156	treeAggregate at RDDLossFunction.scala:76	1/1	0.2s				
157	(1 skipped)	(1 skipped)					
158	treeAggregate at RDDLossFunction.scala:77	1/1	0.2s				
159	(1 skipped)	(1 skipped)					
160	treeAggregate at RDDLossFunction.scala:77	1/1	0.2s				
161	(1 skipped)	(1 skipped)					
162	treeAggregate at RDDLossFunction.scala:77	1/1	0.2s				
163	(1 skipped)	(1 skipped)					
164	treeAggregate at RDDLossFunction.scala:77	1/1	0.2s				
165	rdd at MulticlassClassification	60/60	1.9min	7.5GB			74.7MB
166	map at MulticlassMetrics.scala:45	1/1	2s				64.0 B
167	collectAsMap at MulticlassMetrics.scala:48	1/1	0.1s			64.0 B	
168	(1 skipped)	(1 skipped)	(1 skipped)				
169	countByValue at MulticlassMetrics.scala:42	1/1	2s				64.0 B
170	countByValue at MulticlassMetrics.scala:42	1/1	69ms			64.0 B	
171	rdd at MulticlassClassificationEvaluator	60/60	2.4min	7.5GB			74.7MB
172	map at MulticlassMetrics.scala:45	1/1	0.4s				64.0 B
173	collectAsMap at MulticlassMetrics.scala:48	1/1	37ms				
174	(1 skipped)	(1 skipped)				64.0 B	
175	countByValue at MulticlassMetrics.scala:42	1/1	0.4s				64.0 B
176	countByValue at MulticlassMetrics.scala:42	1/1	36ms			64.0 B	

SVM: Stages Diagram



Input/Output & shuffling byte/sec 7.2.3

Case Scenario 3rd Components:

Logistic Regression

Logistic Regression: Jobs

Job		
ID	**Description**	**Duration**
0	csv at NativeMethodAccessorImpl.java:0	7 s
1	describe at NativeMethodAccessorImpl.java:0	3.6 min
2	treeAggregate	2.0 min

3	treeAggregate	3 s
4	treeAggregate	2 s
5	treeAggregate	2 s
6	treeAggregate	1 s
7	treeAggregate	1 s
8	treeAggregate	1 s
9	treeAggregate	1 s
10	treeAggregate	1 s
11	treeAggregate	1 s
12	treeAggregate	1 s
13	treeAggregate	1 s
14	treeAggregate	1 s
15	treeAggregate	1 s
16	treeAggregate	1 s
17	sortByKey at BinaryClassificationMetrics.scala:155	2.0min
18	collect at BinaryClassificationMetrics.scala:192 c	29 s
19	collect at SlidingRDD.scala:81	16 s
20	aggregate at AreaUnderCurve.scala:45	3 s
21	sortByKey at BinaryClassificationMetrics.scala:155	2.2min
22	count at BinaryClassificationMetrics.scala:163	51s
23	collect at BinaryClassificationMetrics.scala:192	23s
24	collect at <ipython-input-3-583a8e4d239f>:51	23s

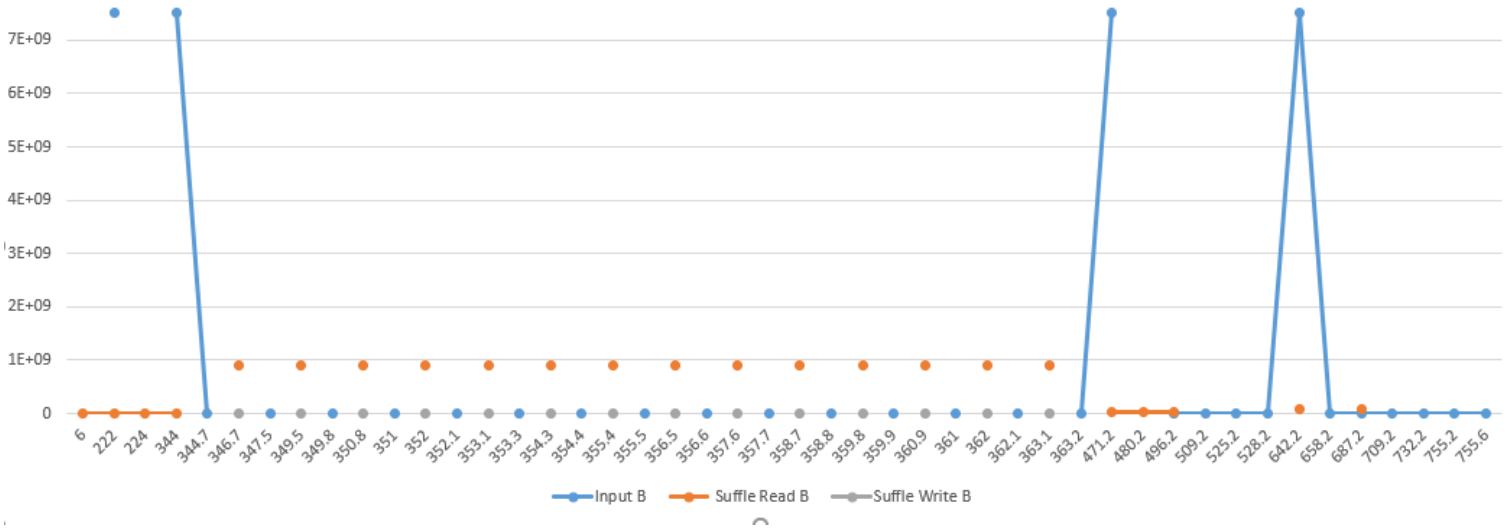
25	collect at <ipython-input-3-583a8e4d239f>:52	0.4s

Logistic Regression: Stages

Stages							
ID	Description	No Tasks	Duration	Input	Output	Suffle Read	Suffle Write
0	default csv at NativeMethodAccessorImpl.java:0	1/1	6s			64.0 KB	
1	default describe at NativeMethodAccessorImpl.java:0	60/60	3.6min	7.5 GB		70.9 KB	
2	default describe at NativeMethodAccessorImpl.java:0	1/1	2s			70.9 KB	
3	default treeAggregate at LogisticRegression.scala:520	60/60	2.0min	7.5 GB		63.8 KB	
4	default treeAggregate at LogisticRegression.scala:520	7/7	0.7s	63.8 KB			
5	default treeAggregate at RDDLossFunction.scala:61	60/60	2s			910.2 MB	61.3 KB
6	default treeAggregate at RDDLossFunction.scala:61	7/7	0.8s	61.3 KB			
7	default treeAggregate at RDDLossFunction.scala:61	60/60	2s			910.2 MB	61.3 KB
8	default treeAggregate at RDDLossFunction.scala:61	7/7	0.3s	61.3 KB			
9	default treeAggregate at RDDLossFunction.scala:61	60/60	1s			910.2 MB	61.3 KB
10	default treeAggregate at RDDLossFunction.scala:61	7/7	0.2s	61.3 KB			
11	default treeAggregate at RDDLossFunction.scala:61	60/60	1s			910.2 MB	61.3 KB
12	default treeAggregate at RDDLossFunction.scala:61	7/7	0.1s	61.3 KB			
13	default treeAggregate at RDDLossFunction.scala:61	60/60	1s			910.2 MB	61.3 KB
14	default treeAggregate at RDDLossFunction.scala:61	7/7	0.2s	61.3 KB			
15	default treeAggregate at RDDLossFunction.scala:61	60/60	1s			910.2 MB	61.3 KB
16	default treeAggregate at RDDLossFunction.scala:61	7/7	0.1s	61.3 KB			
17	default treeAggregate at RDDLossFunction.scala:61	60/60	1s			910.2 MB	61.3 KB
18	default treeAggregate at RDDLossFunction.scala:61	7/7	0.1s	61.2 KB			
19	default treeAggregate at RDDLossFunction.scala:61	60/60	1s			910.2 MB	61.3 KB
20	default treeAggregate at RDDLossFunction.scala:61	7/7	0.1s	61.2 KB			
21	default treeAggregate at RDDLossFunction.scala:61	60/60	1s			910.2 MB	61.3 KB
22	default treeAggregate at RDDLossFunction.scala:61	7/7	0.1s	61.2 KB			
23	default treeAggregate at RDDLossFunction.scala:61	60/60	1s			910.2 MB	61.3 KB
24	default treeAggregate at RDDLossFunction.scala:61	7/7	0.1s	61.2 KB			
25	default treeAggregate at RDDLossFunction.scala:61	60/60	1s			910.2 MB	61.3 KB
26	default treeAggregate at RDDLossFunction.scala:61	7/7	0.1s	61.2 KB			
27	default treeAggregate at RDDLossFunction.scala:61	60/60	1s			910.2 MB	61.3 KB
28	default treeAggregate at RDDLossFunction.scala:61	7/7	0.1s	61.2 KB			
29	default treeAggregate at RDDLossFunction.scala:61	60/60	1s			910.2 MB	61.3 KB

30	default treeAggregate at RDDLossFunction.scala:61	7/7	0.1s	61.2 KB			
31	default treeAggregate at RDDLossFunction.scala:61	60/60	1s			910.2 MB	61.3 KB
32	default treeAggregate at RDDLossFunction.scala:61	7/7	0.1s	61.3 KB			
33	default map at BinaryClassificationEvaluator.scala:81	60/60	1.8min	7.5 GB		41.5 MB	
34	default sortByKey at BinaryClassificationMetrics.scala:155	60/60	9s			41.5 MB	
35	default combineByKey at BinaryClassificationMetrics.scala:151	60/60	16s	41.5 MB		37.5 MB	
36	default collect at BinaryClassificationMetrics.scala:192	60/60	13s	37.5 MB			
37	default collect at SlidingRDD.scala:81	62/62	16s	37.5 MB			
38	default aggregate at AreaUnderCurve.scala:45	61/61	3s	262.1 MB			
39	default map at LogisticRegression.scala:1531	60/60	1.9min	7.5 GB		92.7 MB	
40	default sortByKey at BinaryClassificationMetrics.scala:155	60/60	16s	92.7 MB			
41	default combineByKey at BinaryClassificationMetrics.scala:151	60/60	29s	92.7 MB		85.1 MB	
42	default count at BinaryClassificationMetrics.scala:163	60/60	22s	85.1 MB			
43	default collect at BinaryClassificationMetrics.scala:192	60/60	23s	85.1 MB			
44	default collect at <ipython-input-3-583a8e4d239f>:51	62/62	23s	85.1 MB			
45	default collect at <ipython-input-3-583a8e4d239f>:52	62/62	0.4s	11.3 KB			

Logistic Regression: Stages Diagram



Input/Output & shuffling byte/sec 7.3.1

Naïve Bayes

Naïve Bayes: Jobs

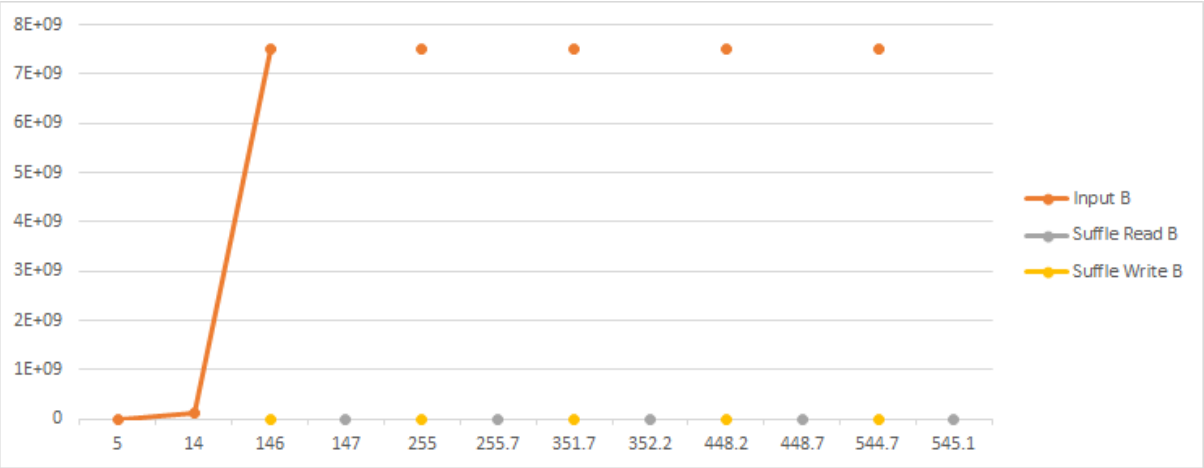
Job ID	Description	Duration
0	csv at NativeMethodAccessorImpl.java:0	6s
1	head at NaiveBayes.scala:156	9s3
2	collect at NaiveBayes.scala:176	2.3min
3	collectAsMap at MulticlassMetrics.scala:48	1.8min
4	countByValue at MulticlassMetrics.scala:42	1.6min
5	collectAsMap at MulticlassMetrics.scala:48	1.6min
6	countByValue at MulticlassMetrics.scala:42	1.6min

Naïve Bayes: Stages

Stages ID	Description	No Tasks	Duration	Input	Output	Suffle Read	Suffle Write
0	csv at NativeMethodAccessorImpl.java:0	1/1	5 s	64.0 KB			
1	head at NaiveBayes.scala:156	1/1	9 s	128.1 MB			
2	map at NaiveBayes.scala:164	60/60	2.2 min	7.5 GB			44.9 KB
3	collect at NaiveBayes.scala:176	60/60	1 s			44.9 KB	
4	map at MulticlassMetrics.scala:45	60/60	1.8 min	7.5 GB			6.4 KB
5	collectAsMap at MulticlassMetrics.scala:48	60/60	0.7 s			6.4 KB	
6	countByValue at MulticlassMetrics.scala:42	60/60	1.6 min	7.5 GB			6.4 KB
7	countByValue at MulticlassMetrics.scala:42	60/60	0.5 s			6.4 KB	
8	map at MulticlassMetrics.scala:45	60/60	1.6 min	7.5 GB			6.4 KB
9	collectAsMap at MulticlassMetrics.scala:48	60/60	0.5 s			6.4 KB	
10	countByValue at MulticlassMetrics.scala:42	60/60	1.6 min	7.5 GB			6.4 KB

11	countByValue at MulticlassMetrics.scala:42	60/60	0.4 s			6.4 KB	

Naïve Bayes: Stages Diagram



Input/Output & shuffling byte/sec 7.3.2

SVM

SVM: Jobs

Job ID	Description	Duration
0	treeAggregate at LinearSVC.scala:190	1.3min
1	treeAggregate at RDDLossFunction.scala:61	5 s
2	treeAggregate at RDDLossFunction.scala:61	1 s
3	treeAggregate at RDDLossFunction.scala:61	0.7 s
4	treeAggregate at RDDLossFunction.scala:61	5 s
5	treeAggregate at RDDLossFunction.scala:61	0.7 s
6	treeAggregate at RDDLossFunction.scala:61	0.7 s

7	treeAggregate at RDDLossFunction.scala:61	0.5 s
8	treeAggregate at RDDLossFunction.scala:61	0.7 s
9	treeAggregate at RDDLossFunction.scala:61	0.6 s
10	treeAggregate at RDDLossFunction.scala:61	0.5 s
11	treeAggregate at RDDLossFunction.scala:61	0.6 s
12	treeAggregate at RDDLossFunction.scala:61	0.7 s
13	treeAggregate at RDDLossFunction.scala:61	0.5 s
14	treeAggregate at RDDLossFunction.scala:61	0.5 s
15	treeAggregate at RDDLossFunction.scala:61	0.4 s
16	treeAggregate at RDDLossFunction.scala:61	0.5 s
17	treeAggregate at RDDLossFunction.scala:61	0.6 s
18	treeAggregate at RDDLossFunction.scala:61	0.4 s
19	treeAggregate at RDDLossFunction.scala:61	0.5 s
20	treeAggregate at RDDLossFunction.scala:61	0.5 s
21	treeAggregate at RDDLossFunction.scala:61	0.6 s
22	treeAggregate at RDDLossFunction.scala:61	0.4 s
23	treeAggregate at RDDLossFunction.scala:61	0.4 s
24	treeAggregate at RDDLossFunction.scala:61	0.4 s
25	treeAggregate at RDDLossFunction.scala:61	0.4 s
26	treeAggregate at RDDLossFunction.scala:61	0.4 s
27	treeAggregate at RDDLossFunction.scala:61	0.3 s
28	treeAggregate at RDDLossFunction.scala:61	0.3 s
29	treeAggregate at RDDLossFunction.scala:61	0.4 s
30	treeAggregate at RDDLossFunction.scala:61	0.3 s
31	treeAggregate at RDDLossFunction.scala:61	0.3 s
32	treeAggregate at RDDLossFunction.scala:61	0.3 s
33	treeAggregate at RDDLossFunction.scala:61	0.3 s
34	treeAggregate at RDDLossFunction.scala:61	0.3 s
35	treeAggregate at RDDLossFunction.scala:61	0.5 s
36	treeAggregate at RDDLossFunction.scala:61	0.6 s
37	treeAggregate at RDDLossFunction.scala:61	0.3 s
38	treeAggregate at RDDLossFunction.scala:61	0.4 s
39	treeAggregate at RDDLossFunction.scala:61	0.4 s
40	treeAggregate at RDDLossFunction.scala:61	0.4 s
41	treeAggregate at RDDLossFunction.scala:61	0.4 s
42	treeAggregate at RDDLossFunction.scala:61	0.3 s
43	treeAggregate at RDDLossFunction.scala:61	0.4 s
44	treeAggregate at RDDLossFunction.scala:61	0.4 s
45	treeAggregate at RDDLossFunction.scala:61	0.4 s
46	treeAggregate at RDDLossFunction.scala:61	0.4 s
47	treeAggregate at RDDLossFunction.scala:61	0.3 s
48	treeAggregate at RDDLossFunction.scala:61	0.4 s
49	treeAggregate at RDDLossFunction.scala:61	0.3 s
50	treeAggregate at RDDLossFunction.scala:61	0.4 s

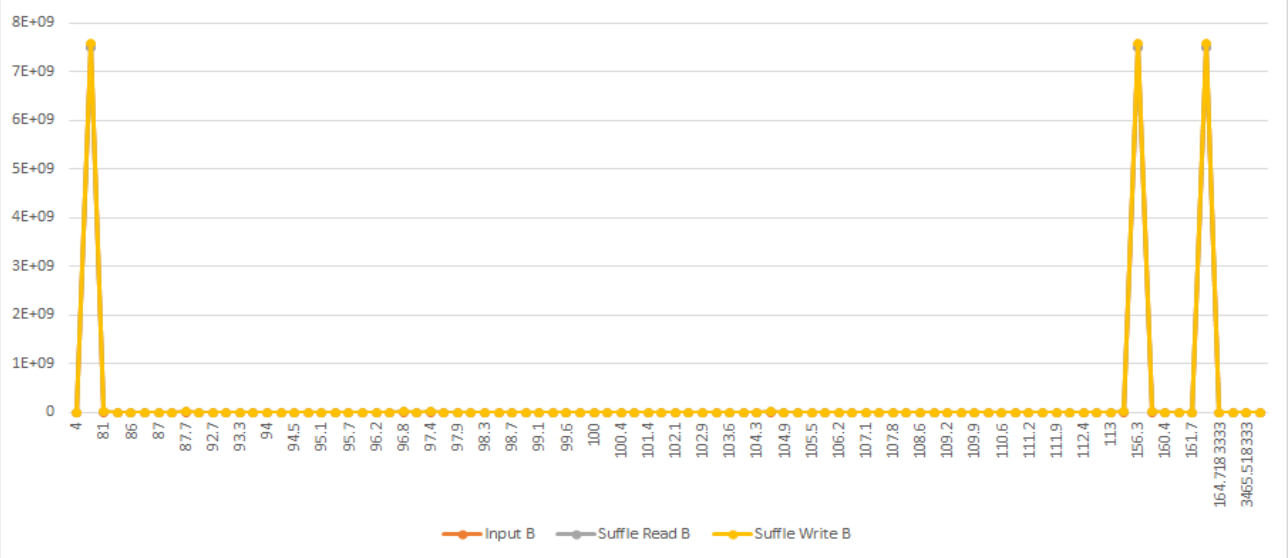
51	treeAggregate at RDDLossFunction.scala:61	0.3 s
52	treeAggregate at RDDLossFunction.scala:61	0.3 s
53	treeAggregate at RDDLossFunction.scala:61	0.3 s
54	treeAggregate at RDDLossFunction.scala:61	0.3 s
55	treeAggregate at RDDLossFunction.scala:61	0.3 s
56	collectAsMap at MulticlassMetrics.scala:48	0.3 s
57	countByValue at MulticlassMetrics.scala:42	47 s
58	collectAsMap at MulticlassMetrics.scala:48	1 s
59	countByValue at MulticlassMetrics.scala:42	1.2 min

SVM: Stages

Stages							
ID	**Description**	**No Tasks**	**Duration**	**Input**	**Output**	**Suffle Read**	**Suffle Write**
0	csv at NativeMethodAccessorImpl.java:0	1/1	4 s	64.0 K			
1	rdd at LinearSVC.scala:169	60/60	1.2 min	7.5 GB			74.7 MB
2	treeAggregate at LinearSVC.scala:190	1/1	5 s			14.9 MB	
4	treeAggregate at RDDLossFunction.scala:61	1/1	5 s				
6	treeAggregate at RDDLossFunction.scala:61	1/1	1.0 s				
8	treeAggregate at RDDLossFunction.scala:61	1/1	0.7 s			16.2 MB	
10	treeAggregate at RDDLossFunction.scala:61	1/1	5 s				
12	treeAggregate at RDDLossFunction.scala:61	1/1	0.6 s				
14	treeAggregate at RDDLossFunction.scala:61	1/1	0.7 s				
16	treeAggregate at RDDLossFunction.scala:61	1/1	0.5 s				
18	treeAggregate at RDDLossFunction.scala:61	1/1	0.6 s				
20	treeAggregate at RDDLossFunction.scala:61	1/1	0.6 s				
22	treeAggregate at RDDLossFunction.scala:61	1/1	0.5 s				
24	treeAggregate at RDDLossFunction.scala:61	1/1	0.6 s			11.2 MB	
26	treeAggregate at RDDLossFunction.scala:61	1/1	0.6 s			26.2 MB	
28	treeAggregate at RDDLossFunction.scala:61	1/1	0.5 s				
30	treeAggregate at RDDLossFunction.scala:61	1/1	0.4 s				
32	treeAggregate at RDDLossFunction.scala:61	1/1	0.4 s				
34	treeAggregate at RDDLossFunction.scala:61	1/1	0.4 s				
36	treeAggregate at RDDLossFunction.scala:61	1/1	0.5 s				
38	treeAggregate at RDDLossFunction.scala:61	1/1	0.4 s				
40	treeAggregate at RDDLossFunction.scala:61	1/1	0.4 s				
42	treeAggregate at RDDLossFunction.scala:61	1/1	0.4 s				
44	treeAggregate at RDDLossFunction.scala:61	1/1	0.6 s				
46	treeAggregate at RDDLossFunction.scala:61	1/1	0.4 s				
48	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
50	treeAggregate at RDDLossFunction.scala:61	1/1	0.4 s				

52	treeAggregate at RDDLossFunction.scala:61	1/1	0.4 s				
54	treeAggregate at RDDLossFunction.scala:61	1/1	0.4 s				
56	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
58	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
60	treeAggregate at RDDLossFunction.scala:61	1/1	0.4 s				
62	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s			17.4 MB	
64	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
66	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
68	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
70	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
72	treeAggregate at RDDLossFunction.scala:61	1/1	0.4 s				
74	treeAggregate at RDDLossFunction.scala:61	1/1	0.6 s				
76	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
78	treeAggregate at RDDLossFunction.scala:61	1/1	0.4 s				
80	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
82	treeAggregate at RDDLossFunction.scala:61	1/1	0.4 s				
84	treeAggregate at RDDLossFunction.scala:61	1/1	0.4 s				
86	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
88	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
90	treeAggregate at RDDLossFunction.scala:61	1/1	0.4 s				
92	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
94	treeAggregate at RDDLossFunction.scala:61	1/1	0.4 s				
96	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
98	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
100	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
102	treeAggregate at RDDLossFunction.scala:61	1/1	0.4 s				
104	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
106	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
108	treeAggregate at RDDLossFunction.scala:61	1/1	0.2 s				
110	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
112	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s				
114	treeAggregate at RDDLossFunction.scala:61	1/1	0.3 s			14.9 MB	
115	rdd at MulticlassClassificationEvaluator.scala:79	60/60	43 s	7.5 GB			74.7 MB
116	map at MulticlassMetrics.scala:45	1/1	4 s			17.4 MB	64.0 B
117	collectAsMap at MulticlassMetrics.scala:48	1/1	0.1 s			64.0 B	
119	countByValue at MulticlassMetrics.scala:42	1/1	0.7 s				64.0 B
120	countByValue at MulticlassMetrics.scala:42	1/1	0.6 s			64.0 B	
121	rdd at MulticlassClassificationEvaluator.scala:79	60/60	1.1 min	7.5 GB			74.7 MB
122	map at MulticlassMetrics.scala:45	1/1	3 s				64.0 B
123	collectAsMap at MulticlassMetrics.scala:48	1/1	55 ms			64.0 B	
125	countByValue at MulticlassMetrics.scala:42	1/1	0.8 s				64.0 B
126	countByValue at MulticlassMetrics.scala:42	1/1	39 ms			64.0 B	

SVM: Stages Diagram



Input/Output & shuffling byte/sec 7.3.3