



ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

**Apache Spark : μια γενική πλατφόρμα για παράλληλη
επεξεργασία μεγάλου όγκου δεδομένων**



Του φοιτητή

Ιωάννη Απόμαχου

Αρ. Μητρώου: 113703

Επιβλέπων καθηγητής

Διαμαντάρας Κωνσταντίνος

Θεσσαλονίκη 2017

ΠΡΟΛΟΓΟΣ

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία αποσκοπεί στην παρουσίαση του μοντέλου και του τρόπου λειτουργίας του Spark, την παρουσίαση των δυο δημοφιλών επιλογών για την συγγραφή κώδικα και την σύγκρισή του με τον άμεσο ανταγωνιστή του για την υλοποίηση εφαρμογών μηχανικής μάθησης το Scikit-learn . Τέλος, παρουσιάζονται οι εφαρμογές που υλοποιήθηκαν, οι οποίες βασίζονται στους αλγορίθμους: K-means, Regression και SVM. Η εργασία πραγματοποιήθηκε από τον Ιωάννη Απόμαχο, φοιτητή του Αλεξάνδρειου Τεχνολογικού Ιδρύματος Θεσσαλονίκης με επιβλέποντα καθηγητή τον κύριο Κωνσταντίνο Διαμαντάρα.

Λέξεις κλειδιά: Spark, Hadoop, Machine Learning, Analyzing Big Data, K-means, Regression, SVM, Scala

ABSTRACT

The present thesis aims to present Spark's model and to explain how its sub-components work, to demonstrate the two most popular programming languages available choices and to compare Spark with its immediate competitor in implementation of machine learning applications, the Scikit-learn. Last but not least, this thesis present the applications, which have been created to support this thesis statements, and are based on the following algorithms: K-means, Regression and SVM. The thesis was conducted by Ioannis Apomachos, student of Alexandrian Technological Institute of Thessaloniki and was supervised by professor Konstantino Diamantara.

Keywords: Spark, Hadoop, Machine Learning, Analyzing Big Data, K-means, Regression, SVM, Scala

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω θερμά τον καθηγητή κ. Κωνσταντίνο Διαμαντάρα για την καθοδήγηση που μου πρόσφερε στην εκπόνηση της πτυχιακής μου εργασίας, όπως επίσης και για την πολύτιμη βοήθεια του για την επίλυση διαφόρων θεμάτων.

Επίσης, θα ήθελα να ευχαριστήσω την οικογένειά μου για τις αμέτρητες θυσίες που πραγματοποίησαν όλα αυτά τα χρόνια, προκειμένου να μου εξασφαλίσουν το δικαίωμα στην μάθηση.

Τέλος θα ήθελα να ευχαριστήσω την αρραβωνιαστικιά μου, Πόπη και όλους τους συναδέλφους και φίλους που είχα την τύχη να γνωρίσω και να συνεργαστώ.

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΡΟΛΟΓΟΣ.....	2
ΠΕΡΙΛΗΨΗ.....	2
ABSTRACT	3
ΕΥΧΑΡΙΣΤΙΕΣ	4
ΠΕΡΙΕΧΟΜΕΝΑ	5
Ευρετήριο πινάκων.....	6
ΕΙΣΑΓΩΓΗ.....	7
ΚΕΦΑΛΑΙΟ 1 : Βασικά χαρακτηριστικά του Spark.....	8
ΕΙΣΑΓΩΓΗ.....	8
ΥΠΟΚΕΦΑΛΑΙΟ 1.1 Ιστορική αναδρομή	9
ΥΠΟΚΕΦΑΛΑΙΟ 1.2 Πλεονεκτήματα Spark	9
ΥΠΟΚΕΦΑΛΑΙΟ 1.3 Κλασσικές περιπτώσεις χρήσης Spark	10
ΥΠΟΚΕΦΑΛΑΙΟ 1.4 Για ποιους προορίζεται	11
ΥΠΟΚΕΦΑΛΑΙΟ 1.5 Το μέλλον του Spark (Spark 2).....	12
ΚΕΦΑΛΑΙΟ 2 : Αρχιτεκτονικό μοντέλο και τρόπος λειτουργίας.....	12
ΕΙΣΑΓΩΓΗ.....	12
ΥΠΟΚΕΦΑΛΑΙΟ 2.1 Περιγραφή του μοντέλου Hadoop.....	13
ΥΠΟΚΕΦΑΛΑΙΟ 2.2 Το Spark σε σχέση με το MapReduce του Hadoop	13
ΥΠΟΚΕΦΑΛΑΙΟ 2.3 Τα υποσυστήματα του Spark.....	16
ΥΠΟΚΕΦΑΛΑΙΟ 2.3.1 Spark Core	17
ΥΠΟΚΕΦΑΛΑΙΟ 2.3.2 Mlib	17
ΥΠΟΚΕΦΑΛΑΙΟ 2.3.3 Spark Streaming.....	18
ΥΠΟΚΕΦΑΛΑΙΟ 2.3.4 Spark SQL	19
ΥΠΟΚΕΦΑΛΑΙΟ 2.3.5 GraphX.....	20
ΥΠΟΚΕΦΑΛΑΙΟ 2.3.4 Cluster Managers και η δομή των κόμβων	21
ΥΠΟΚΕΦΑΛΑΙΟ 2.4 Web Interfaces	25
ΚΕΦΑΛΑΙΟ 3 : Υλοποίηση Εφαρμογών σε Scala και Python	29
ΕΙΣΑΓΩΓΗ.....	29
ΥΠΟΚΕΦΑΛΑΙΟ 3.1 Η γλώσσα Scala	29
ΥΠΟΚΕΦΑΛΑΙΟ 3.2 Η γλώσσα Python.....	29
ΥΠΟΚΕΦΑΛΑΙΟ 3.3 Scala ή Python.....	30
ΥΠΟΚΕΦΑΛΑΙΟ 3.3.1 Ευκολία εκμάθησης.....	32
ΥΠΟΚΕΦΑΛΑΙΟ 3.3.2 Ευκολία χρήσης.....	32

ΥΠΟΚΕΦΑΛΑΙΟ 3.3.3 Προηγμένες δυνατότητες	32
ΥΠΟΚΕΦΑΛΑΙΟ 3.3.4 Performance	33
ΥΠΟΚΕΦΑΛΑΙΟ 3.3.5 TypeSafety	33
ΥΠΟΚΕΦΑΛΑΙΟ 3.4 The Spark Programming model	34
ΥΠΟΚΕΦΑΛΑΙΟ 3.4.1 Resilient Distributed Datasets (RDDs) και DAG.....	34
ΥΠΟΚΕΦΑΛΑΙΟ 3.4.2 Parallel Operations	37
ΥΠΟΚΕΦΑΛΑΙΟ 3.4.3 Shared Variables.....	37
ΚΕΦΑΛΑΙΟ 4 : Αλγόριθμοι μηχανικής μάθησης βασισμένοι στις βιβλιοθήκες ML και MLLIB	39
ΕΙΣΑΓΩΓΗ.....	39
ΥΠΟΚΕΦΑΛΑΙΟ 4.1 Μηχανική Μάθηση και ML lib.....	39
ΥΠΟΚΕΦΑΛΑΙΟ 4.2 Kmeans (Clustering)	41
ΥΠΟΚΕΦΑΛΑΙΟ 4.3 SVM (Classification).....	42
ΥΠΟΚΕΦΑΛΑΙΟ 4.4 Regression	42
ΚΕΦΑΛΑΙΟ 5 : Δημιουργία Εφαρμογών	44
ΕΙΣΑΓΩΓΗ.....	44
ΥΠΟΚΕΦΑΛΑΙΟ 5.1 Analyzing Big Data.....	44
ΥΠΟΚΕΦΑΛΑΙΟ 5.2 Scikit	45
ΥΠΟΚΕΦΑΛΑΙΟ 5.3 Spark vs Scikit	46
ΥΠΟΚΕΦΑΛΑΙΟ 5.4 Εφαρμογές Kmeans.....	46
ΥΠΟΚΕΦΑΛΑΙΟ 5.5 Εφαρμογές SVM	48
ΥΠΟΚΕΦΑΛΑΙΟ 5.6 Εφαρμογές Regression	50
ΣΥΜΠΕΡΑΣΜΑΤΑ.....	52
ΒΙΒΛΙΟΓΡΑΦΙΑ	52
Το παράδειγμα SparkPi.....	55
ΟΔΗΓΟΣ ΧΡΗΣΗΣ ΛΟΓΙΣΜΙΚΟΥ	56
Εγκατάσταση του VirtualBox της Oracle και δημιουργία Virtual Machine	56
Εγκατάσταση Python 2.7 (με anaconda).....	56
Εγκατάσταση Scikit.....	57
Εγκατάσταση Spark	57
Οδηγίες για το setup των cluster manager στο Spark	57

Ευρετήριο πινάκων

Πίνακας 1 Συγκριτικό για τον αλγόριθμο Kmeans	46
Πίνακας 2 Συγκριτικό για τον αλγόριθμο SVM	48
Πίνακας 3 Συγκριτικό για τον αλγόριθμο Regression	50

ΕΙΣΑΓΩΓΗ

Το πεδίο τη μηχανικής μάθησης και της ανάλυσης «big data» αποτελεί ολοένα και πιο ισχυρό πόλο έλξης ερευνητικού ενδιαφέροντος σε ολόκληρη την επιστημονική κοινότητα. Η επιστήμη ανάλυσης «big data» καλείται να απαντήσει σε σύνθετα προβλήματα της σύγχρονης εποχής, γεγονός το οποίο πριν από μερικά χρόνια ήταν αδύνατο. Προβλήματα όπως η δημιουργία ενός μοντέλου, όπου θα ανακαλύπτει απάτες με πιστωτικές κάρτες απαιτούν νέες επιστημονικές προσεγγίσεις εξαιτίας της μεγάλης πολυπλοκότητας και του μεγάλου μεγέθους των δεδομένων. Στην παρούσα εργασία προσεγγίζουμε τις επιστήμες αυτές με την ανάλυση τους σε βάθος, καθώς και την δημιουργία αντιπροσωπευτικών εφαρμογών.

Οι στόχοι στους οποίους αποσκοπεί η συγκεκριμένη πτυχιακή εργασία μπορούν να συνοψιστούν παρακάτω και αφορούν :

- Στη διερεύνηση των δυνατοτήτων και των παροχών της πλατφόρμας του Spark, καθώς και την ανάλυση των δομικών στοιχείων του.
- Στην ολοκληρωμένη και αντικειμενική σύγκριση των τεχνολογιών και των γλωσσών προγραμματισμού που απαρτίζουν τον τομέα της μηχανικής μάθησης.
- Στην πλήρη και προσεκτική καταγραφή ζητημάτων που προέκυψαν, ώστε να αποτελέσει χρήσιμο και ολοκληρωμένο εκπαιδευτικό υλικό για οποιονδήποτε σπουδαστή που θα αποφασίσει να ασχοληθεί με το ίδιο ή παρεμφερές θέμα.

Η εργασία που ακολουθεί αποτελείται από πέντε κεφάλαια.

Στο πρώτο κεφάλαιο παρουσιάζεται μια ιστορική αναδρομή, καθώς και βασικές πληροφορίες για το Spark, όπως είναι οι περιπτώσεις χρήσης του και για ποιους προορίζεται.

Στο δεύτερο κεφάλαιο παρουσιάζεται αναλυτικά η σχέση του Spark με το Hadoop, καθώς και το αρχιτεκτονικό μοντέλο και τρόπος λειτουργίας του Spark. Επιπλέον γίνεται ανάλυση όλων των οντοτήτων που συνθέτουν την «συστάδα» του Spark, όπως για παράδειγμα Spark SQL και GraphX.

Στο τρίτο κεφάλαιο παρουσιάζονται οι δυο πιο δημοφιλείς γλώσσες προγραμματισμού που χρησιμοποιούνται από το Spark για την δημιουργία του driver προγράμματος, καθώς και εκτενής σχολιασμός και σύγκριση των χαρακτηριστικών των δυο αυτών γλωσσών (Scala και Python). Τέλος, γίνεται παρουσίαση του προγραμματιστικού μοντέλου του Spark.

Στο τέταρτο κεφάλαιο γίνεται αναφορά στους τρεις αλγορίθμους μηχανικής μάθησης, δηλαδή Kmeans, SVM και Regression, που χρησιμοποιήθηκαν για την πτυχιακή αυτή.

Στο πέμπτο και τελευταίο κεφάλαιο της πτυχιακής παρουσιάζονται οι προκλήσεις που αντιμετωπίζει η Επιστήμη Δεδομένων, καθώς και ένα εκτενές συγκριτικό ανάμεσα στο Spark και στο Scikit Learn, το οποίο είναι ο άμεσος ανταγωνιστής του Spark στον τομέα της μηχανικής μάθησης. Τέλος, παρουσιάζεται ο κώδικας και τα datasets που χρησιμοποιήθηκαν για την υλοποίηση των εφαρμογών μαζί με αναλυτικούς πίνακες τιμών και χρόνων, των εκάστοτε εφαρμογών.

ΚΕΦΑΛΑΙΟ 1 : Βασικά χαρακτηριστικά του Spark

ΕΙΣΑΓΩΓΗ

Στο κεφάλαιο αυτό αρχικά, παρουσιάζεται η ιστορική αναδρομή του Spark και πραγματοποιείται μια εκτενής ανάλυση των κύριων χαρακτηριστικών και υπηρεσιών του. Επίσης, παρουσιάζεται ο σκοπός και οι στόχοι που πρόκειται να εκπληρώσει , καθώς και το κοινό για το οποίο αποσκοπεί .

ΥΠΟΚΕΦΑΛΑΙΟ 1.1 Ιστορική αναδρομή

Το Spark ξεκίνησε το 2009 , ως έργο του εργαστηρίου AMPLab του πανεπιστημίου Berkeley της Καλιφόρνια. Οι ερευνητές του εργαστηρίου προηγουμένως μελετούσαν το MapReduce του Hadoop και παρατήρησαν πως αυτό ήταν ακατάλληλο για επαναλαμβανόμενες και διαδραστικές υπολογιστικές διεργασίες, όπου ένα σύνολο δεδομένων επαναχρησιμοποιείται σε διάφορες επαναλήψεις. Πιο συγκεκριμένα, το έργο δημιουργήθηκε λόγω της αναγκαιότητας να γίνει αξιολόγηση του νεοσύστατου cluster manager εν ονόματι «Mesos» , το οποίο επίσης δημιουργήθηκε στο AMPLab. Η πρώτη επίσημη αναφορά έγινε σε επιστημονικό άρθρο με τίτλο «Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center» των Benjamin Hindman, Andy Konwinski και Matei Zaharia .

Το Spark έγινε πλέον, αναπόσπαστο κομμάτι του Apache Software Foundation το 2013, και στις αρχές του 2014 έφτασε να είναι ένα από τα κορυφαία και πιο ενεργά projects του ιδρύματος, έχοντας στο δυναμικό της μια τεράστια κοινότητα , η οποία αποτελείται τόσο από μεμονωμένα άτομα που συνεισφέρουν με τις ιδέες τους όσο και από φορείς ,κερδίζοντας έτσι μεγάλα ονόματα του είδους με αποτέλεσμα να επενδύουν στο project. Ενδεικτικά παραδείγματα αποτελούν η IBM, η Databricks και η Huawei.

Ο αριθμός των συμμετεχόντων που συνεισφέρουν αυξάνεται ραγδαία με κάθε καινούρια έκδοση, καθώς η κοινότητα συνεχίζει να προσφέρει αναβαθμισμένες εκδόσεις του Spark με σταθερό ρυθμό. Παράδειγμα στην πρώτη έκδοση (Spark 1.0) υπήρχαν πάνω από 100 μοναδικοί συμμετέχοντες. (Scott,2015:7)

ΥΠΟΚΕΦΑΛΑΙΟ 1.2 Πλεονεκτήματα Spark

Τα πλεονεκτήματα του Spark συνοψίζονται στους παρακάτω 3 άξονες:

- **Απλότητα.** Οι δυνατότητες του Spark είναι προσβάσιμες μέσω από μιας σειράς από APIs. Τα APIs είναι σχεδιασμένα ειδικά ώστε να αλληλεπιδρούν γρήγορα και με ευκολία σε μεγάλου όγκου δεδομένα. Τα APIs είναι πολύ καλά τεκμηριωμένα και δομημένα με αποτέλεσμα οι αναλυτές δεδομένων και οι προγραμματιστές να μπορούν να χρησιμοποιούν πολύ γρήγορα και αποτελεσματικά το Spark.

- **Ταχύτητα.** Το Spark έχει σχεδιαστεί ώστε να λειτουργεί ταχύτατα, λειτουργώντας τόσο στην μνήμη όσο και στον σκληρό δίσκο. Το 2014 το Spark συμμετείχε στον διαγωνισμό με όνομα «Daytona Gray Sort benchmarking challenge» όπου επεξεργάστηκε 100 terabytes δεδομένα, τα οποία ήταν αποθηκευμένα σε δίσκους SSD σε μόλις 23 λεπτά κερδίζοντας έτσι την πρώτη θέση. Αξίζει να σημειωθεί ότι το προηγούμενο ρεκόρ το κατείχε το MapReduce του Hadoop με χρόνο 72 λεπτά χρησιμοποιώντας 2100 συστάδες υπολογιστών (clusters), αναδεικνύοντας τον ηγετικό ρόλο του Spark όπου επεξεργάστηκε 3 φορές γρηγορότερα τα δεδομένα με 10 φορές λιγότερους clusters. Οι επιδόσεις του Spark μπορούν να γίνουν ακόμα καλύτερες όταν χρησιμοποιείται για διαδραστικά ερωτήματα σε δεδομένα (interactive queries) που είναι αποθηκευμένα στην μνήμη.
- **Υποστήριξη.** Το Spark υποστηρίζει ένα μεγάλο εύρος από γλώσσες προγραμματισμού όπως: Java, Python, R, και Scala. Αν και το Spark συσχετίζεται συχνά με το «κατώτερο σύστημα αποθήκευσης» (underlying storage system) του Hadoop το HDFS (White,2015:43), το Spark περιλαμβάνει εγγενή υποστήριξη για ενσωμάτωση (tight integration) σε μεγάλο αριθμό λύσεων αποθήκευσης τόσο από το οικοσύστημα του Hadoop όσο και διαφόρων άλλων συστημάτων. Επιπρόσθετα, η κοινότητα του Apache Spark είναι εκτενής, δραστήρια καθώς και διεθνής.. Μια αναπτυσσόμενη ομάδα από εμπορικούς προμηθευτές συμπεριλαμβανομένων των Databricks, IBM, καθώς και άλλων βασικών προμηθευτών του Hadoop, παραδίδουν μια περιεκτική υποστήριξη για εφαρμογές και λύσεις που βασίζονται στο Spark. (Scott,2015:8)

ΥΠΟΚΕΦΑΛΑΙΟ 1.3 Κλασσικές περιπτώσεις χρήσης Spark

Οι κλασσικές περιπτώσεις χρήσης του Spark συνοψίζονται στους παρακάτω 4 άξονες:

- **Επεξεργασία ροών δεδομένων.** Από αρχεία καταγραφών συμβάντων έως και δεδομένα που προέρχονται από αισθητήρες, οι προγραμματιστές εφαρμογών όλο και περισσότερο έρχονται αντιμέτωποι με ροές δεδομένων. Αυτά τα δεδομένα καταφθάνουν σε μια σταθερή ροή, συχνά από διαφορετικές πηγές ταυτόχρονα. Συνήθως, αν και είναι εφικτό αυτές οι ροές δεδομένων να αποθηκευτούν στον δίσκο και να αναλυθούν σε βάθος χρόνου, είναι προτιμότερο ή ακόμα και σπουδαιότερο να γίνει η επεξεργασία και να παρθούν οι κατάλληλες αποφάσεις κατά την διάρκεια που τα δεδομένα καταφθάνουν. Οι ροές δεδομένων που αφορούν για παράδειγμα οικονομικές συναλλαγές, πρέπει να επεξεργαστούν σε πραγματικό χρόνο, έτσι ώστε να αναγνωριστούν και να απορριφτούν πιθανές επισφαλείς συναλλαγές.
- **Μηχανική μάθηση.** Καθώς το σύνολο των δεδομένων αυξάνεται διαρκώς, οι προσεγγίσεις με βάση την μηχανική μάθηση γίνονται πιο πρακτικές και πιο ακριβείς με την πάροδο του χρόνου. Το λογισμικό μπορεί να εκπαιδευτεί ώστε να αναγνωρίζει τα δεδομένα, να τα επεξεργάζεται και να ενεργεί κατάλληλα παίρνοντας την ανάλογη απόφαση. Η εκπαίδευση

πραγματοποιείται χρησιμοποιώντας γνωστά πρότυπα. Μετά την εκπαίδευση το λογισμικό είναι σε θέση να χρησιμοποιηθεί σε παρόμοιες λύσεις καινούριων αλλά αγνώστων προτύπων. Η ικανότητα του Spark να αποθηκεύει δεδομένα στην μνήμη και στην συνέχεια να πραγματοποιεί συνεχώς ερωτήματα (queries) πάνω σε αυτά, το καθιστά ιδανικό για εκμάθηση αλγορίθμων μηχανικής μάθησης. Αξίζει να σημειωθεί πως όταν πραγματοποιούνται παρόμοια ερωτήματα συνεχώς, σε κλίμακα, μειώνουμε δραστικά τον χρόνο που χρειάζεται για να πλοηγηθούμε μέσα σε μια σειρά από λύσεις έτσι ώστε να επιλέξουμε τον πιο αποτελεσματικό αλγόριθμο.

- **Interactive analytics.** Οι οικονομικοί/επιχειρηματικοί αναλυτές και οι επιστήμονες χρειάζονται όλο και περισσότερο να εξερευνήσουν τα δεδομένα τους ασκώντας περισσότερο ειδικευμένες ερωτήσεις, να βλέπουν το αποτέλεσμα και μετέπειτα να τροποποιούν έστω και ελάχιστα την αρχική ερώτηση με σκοπό να διεισδύσουν περισσότερο στα δεδομένα τους, ώστε να δημιουργήσουν νέες πληροφορίες υπό τη μορφή πινάκων. Για παράδειγμα οι πίνακες μπορεί να βασίζονται στις πωλήσεις, στις γραμμές παραγωγής ή ακόμα και στις τιμές των αγορών. Όλη η παραπάνω διαδικασία πραγματοποιείται με την χρήση διαδραστικών ερωτημάτων και είναι προτιμότερη για προφανής λόγους, σε σχέση με την χρήση προκαθορισμένων ερωτημάτων. Τα διαδραστικά ερωτήματα χρειάζονται συστήματα που λειτουργούν όπως το Spark έτσι ώστε να είναι σε θέση να ανταποκρίνονται και να προσαρμόζονται πιο γρήγορα.
- **Data integration.** Τα δεδομένα που έχουν παραχθεί από διαφορετικές πηγές μιας επιχείρησης, είναι σπανίως ξεκάθαρα ή συνεπή έτσι ώστε να συνδυαστούν με απλό και εύκολο τρόπο, ώστε να παραχθεί μια αναφορά ή μια ανάλυση. Η εξόρυξη, η μετατροπή και το «φόρτωμα» (Extract, Transform, Load) -από δω και στο εξής θα αναφέρεται ως ETL -είναι διεργασίες που χρησιμοποιούνται συχνά για να «εξορυχθούν» τα δεδομένα από τις διάφορες πηγές, καθαρά και τυποποιημένα, ώστε μετά να φορτωθούν σε κάποια υπολογιστικά συστήματα με σκοπό να αναλυθούν. Το Spark χρησιμοποιείται ολοένα και περισσότερο με σκοπό να μειωθεί το κόστος και ο χρόνος που χρειάζεται για την διαδικασία ETL. (Scott,2015:10)

ΥΠΟΚΕΦΑΛΑΙΟ 1.4 Για ποιους προορίζεται

Ένα μεγάλο εύρος επενδυτών της τεχνολογίας, έσπευσαν να υποστηρίξουν το Spark, αναγνωρίζοντας την ευκαιρία να επεκτείνουν τα ήδη υπάρχοντα προϊόντα «big data» σε τομείς όπως τα διαδραστικά ερωτήματα (interactive querying) και την μηχανική μάθηση (machine learning) όπου το Spark ξεχωρίζει, δείχνοντας έτσι την πραγματική του αξία.

Γνωστές επιχειρήσεις/οργανισμοί όπως η IBM και η Huawei έχουν επενδύσει σημαντικά ποσά στην τεχνολογία του Spark, καθώς επίσης και ένα αυξανόμενος αριθμός από startup εταιρίες βασίζονται σε αυτό, είτε εξολοκλήρου είτε ως ένα μέρος τους. Για παράδειγμα το 2013 η ομάδα Berkeley Team, που είναι υπεύθυνη για την δημιουργία του Spark, δημιούργησε την Databricks, παρέχοντας έτσι, μια ολοκληρωμένη και αυτόνομη (end-to-end) πλατφόρμα βασισμένη στο Spark. Η εταιρία, η οποία έχει πλούσια χρηματοδότηση, έχει λάβει 47 εκατομμύρια δολάρια μέσα σε δυο περιόδους επενδύσεων, το 2013 και το 2014, και οι υπάλληλοι της Databricks συνεχίζουν να διαδραματίζουν έναν επιφανή ρόλο στην βελτίωση και στην επέκταση του ανοικτού κώδικα του Apache Spark project.

Οι βασικοί διανομείς του Hadoop, συμπεριλαμβανομένων των MapR, Cloudera και Hortonworks έχουν στα σχέδια τους την υποστήριξη ως προς το Spark παράλληλα με τις υπόλοιπες υπηρεσίες που παρέχουν.

Τέλος μερικές Web-based εταιρίες όπως η Κινέζικη μηχανή αναζήτησης Baidu, e-commerce operation Alibaba Taobao, και η social networking Tencent όλες εργασίες είναι βασισμένες στο Spark (Spark-based operations) σε κλίμακα. Για παράδειγμα η Tencent έχει 800 εκατομμύρια ενεργούς χρήστες, οι οποίοι σύμφωνα με πληροφορίες παράγουν ημερησίως 700TB δεδομένων για επεξεργασία σε ένα cluster αποτελούμενο από 8000 compute nodes. Επιπλέον μια φαρμακευτική εταιρία κολοσσός με όνομα Novartis βασίζεται στο Spark, με σκοπό την μείωση του χρόνου που χρειάζεται για να παραχθούν «modeling data» στα χέρια των ερευνητών, διατηρώντας παράλληλα ηθικά και γραφειοκρατικά (contractual safeguards) ζητήματα, όσον αφορά τα συμβόλαια μεταξύ εταιριών.(Scott,2015:9)

ΥΠΟΚΕΦΑΛΑΙΟ 1.5 Το μέλλον του Spark (Spark 2)

Το Apache Spark 2.0.0 αποτελεί την πρώτη έκδοση της σειράς 2.x . Τα βασικά χαρακτηριστικά της έκδοσης αυτής είναι API usability, SQL 2003 support, βελτιώσεις στην απόδοση, structured streaming, R UDF support, καθώς και λειτουργικές βελτιώσεις. Επιπλέον, αυτή η έκδοση περιλαμβάνει 2500 patches από 300 διαφορετικούς συνεισφέροντες. (<https://0x0fff.com/apache-spark-future/>)

ΚΕΦΑΛΑΙΟ 2 : Αρχιτεκτονικό μοντέλο και τρόπος λειτουργίας

ΕΙΣΑΓΩΓΗ

Στο τρέχον κεφάλαιο, γίνεται παρουσίαση του μοντέλου Hadoop , καθώς και της σχέσης του με το Spark. Επιπλέον ,γίνεται αναλυτική παρουσίαση όλων των τμημάτων που απαρτίζουν το Spark, μαζί με τον τρόπο λειτουργίας τους.

ΥΠΟΚΕΦΑΛΑΙΟ 2.1 Περιγραφή του μοντέλου Hadoop

Το Apache Hadoop, είναι μια πλατφόρμα ανοικτού κώδικα για κατανεμημένο αποθηκευτικό χώρο και επεξεργασία μεγάλου όγκου δεδομένων πάνω σε μια σειρά από clusters (συστάδες) , φτιαγμένες από χαμηλού κόστους hardware , εξειδικευμένο να εκτελεί εντολές παράλληλα (commodity hardware). Όλα τα τμήματα του Hadoop είναι σχεδιασμένα έχοντας σαν θεμελιώδη γνώση ότι το hardware κάποια στιγμή θα πάθει βλάβη (δηλαδή θα «αστοχήσει») γεγονός το οποίο, θα πρέπει να την χειρισθεί το λογισμικό.

Ο πυρήνας του Apache Hadoop απαρτίζεται από δυο τμήματα: ένα που λειτουργεί ως χώρος αποθήκευσης, γνωστό και ως Hadoop Distributed File System (HDFS) καθώς και άλλο ένα τμήμα υπεύθυνο για την επεξεργασία δεδομένων που ονομάζεται MapReduce. Το Hadoop, διασπά τα αρχεία σε μεγάλα τμήματα και τα διανέμει σε κόμβους (nodes) που ανήκουν σε ένα cluster. Για να πραγματοποιηθεί η επεξεργασία των δεδομένων, το Hadoop μεταφέρει αρχεία τύπου Jar (packaged code) στους κόμβους ,στους οποίους η επεξεργασία πραγματοποιείται παράλληλα. Ο συγκεκριμένος τρόπος προσέγγισης, έχει το πλεονέκτημα ότι οι κόμβοι επεξεργάζονται δεδομένα, στα οποία έχουν πρόσβαση τοπικά (τεχνική του data locality), επιτρέποντας τα δεδομένα να τα επεξεργαστούν πιο γρήγορα και πιο αποτελεσματικά σε σχέση με πιο συμβατικές αρχιτεκτονικές υπέρ-υπολογιστών, όπου βασίζονται σε ένα σύστημα αποθήκευσης, αποτελούμενο από διακομιστές (servers) .Σε αυτήν την αρχιτεκτονική οι οδηγίες/υπολογισμοί (computation) και τα δεδομένα διαμοιράζονται μέσω ενός δικτύου υψηλής ταχύτητας.

Το Hadoop framework είναι υλοποιημένο κατά κόρων στην προγραμματιστική γλώσσα Java , καθώς και σε ένα μικρό ποσοστό στην γλώσσα C , η οποία παρέχει command line utilities που χρησιμοποιούνται ως shell scripts. Αν και στο MapReduce ο κώδικας σε Java είναι συνηθισμένος ,οποιαδήποτε προγραμματιστική γλώσσα μπορεί να χρησιμοποιηθεί μαζί με το "Hadoop Streaming" για να ενσωματώσει τα τμήματα "map" και "reduce" του προγράμματος του χρήστη. (<http://hadoop.apache.org/>)

ΥΠΟΚΕΦΑΛΑΙΟ 2.2 Το Spark σε σχέση με το MapReduce του Hadoop

Από την αρχή, το Spark ήταν βελτιστοποιημένο, προκειμένου να τρέχει στην μνήμη (RAM), βοηθώντας με αυτόν τον τρόπο την επεξεργασία δεδομένων να πραγματοποιείται πολύ πιο γρήγορα σε σχέση με εναλλακτικές προσεγγίσεις όπως το MapReduce του Hadoop, το οποίο τείνει να γράφει δεδομένα από και προς τον σκληρό δίσκο ενός υπολογιστή, σε κάθε στάδιο της επεξεργασίας.

Οι υποστηρικτές του Spark ισχυρίζονται πως το Spark είναι 10 φορές ταχύτερο όταν χρησιμοποιεί δεδομένα που βρίσκονται στον σκληρό δίσκο σε σχέση με το MapReduce του Hadoop. Αν το

πρόγραμμα τρέχει στην μνήμη (RAM) , δυνατότητα μόνο του Spark, τότε το Spark είναι 100 φορές πιο γρήγορο.

Βέβαια η σύγκριση των δύο τεχνολογιών δεν είναι εντελώς δίκαιη γιατί η άμεση υπολογιστική δύναμη τείνει να είναι πιο σημαντική στις κλασικές περιπτώσεις χρήσης του Spark σε σχέση με την τεχνική επεξεργασίας του batch processing, δηλαδή η εκτέλεση μιας σειράς από διεργασίες ενός προγράμματος χωρίς την χειροκίνητη παρέμβαση του χρήστη όπου οι διεργασίες ομαδοποιούνται με σκοπό να έχουμε πολλαπλές εισόδους δεδομένων, στην οποία οι λύσεις τύπου MapReduce υπερτερούν με διαφορά.

Το Spark διατηρεί την γραμμική κλιμάκωση και την ανοχή στα σφάλματα όπως και το MapReduce, αλλά τα επεκτείνει με τρεις διαφορετικούς τρόπους:

- 1) Αντί να βασίζεται στο μοντέλο map-then-reduce το οποίο είναι άκαμπτο, ο πυρήνας του μπορεί να δημιουργήσει ένα «κατευθυνόμενο άκυκλο γράφο» (DAG), βασιζόμενο στην εκάστοτε διεργασία που έχει αναλάβει. Το γεγονός αυτό ,αποδεικνύει ότι, στις περιπτώσεις όπου το MapReduce πρέπει να καταγράψει κάποια ενδιάμεσα αποτελέσματα σε κάποιο distributed filesystem, το Spark περνάει τα δεδομένα αυτά στο επόμενο βήμα του pipeline. Ο συγκεκριμένος τρόπος λειτουργίας είναι παρόμοιος με το Dryad, ένα project όπου προήλθε από το τμήμα της Microsoft Research.
- 2) Συμπληρώνει αυτές τις λειτουργίες με ένα πλούσιο σετ από μετασχηματισμούς, επιτρέποντας στους χρήστες να εκφράσουν τους υπολογισμούς τους πιο εύκολα. Γενικά το Spark, εστιάζει στην διευκόλυνση των προγραμματισμών, παρέχοντας ένα εύχρηστο API το οποίο μπορεί να εκφράσει πολύπλοκα pipelines, σε λίγες γραμμές κώδικα.
- 3) Το Spark επεκτείνει την δυνατότητα του πρόγονού του για επεξεργασία δεδομένων στην κύρια μνήμη. Η αφαιρετικότητα των Resilient Distributed Dataset (RDD) του Spark, επιτρέπει στους προγραμματιστές να αποθηκεύσουν ένα οποιοδήποτε σημείο ενός υπο-επεξεργασία pipeline στην μνήμη του cluster. Το γεγονός αυτό, σημαίνει, πως σε οποιαδήποτε μελλοντικά βήματα χρειαστεί να επεξεργαστούν τα ίδια δεδομένα, δεν υπάρχει ανάγκη για τον επανυπολογισμό τους ή την άντληση τους από τον δίσκο. Η συγκεκριμένη δυνατότητα αυξάνει τον αριθμό των περιπτώσεων χρήσης όπου οι παλαιότερες κατανεμημένες μηχανές επεξεργασίας αδυνατούσαν να αναλάβουν και να ολοκληρώσουν. Το Spark λοιπόν, είναι κατάλληλο για άκρως επαναληπτικούς αλγορίθμους , στους οποίους χρειάζεται τα dataset να επεξεργαστούν σε πολλαπλούς κύκλους, καθώς και για reactive applications που χρειάζονται να ανταποκριθούν με συντομία στα ερωτήματα των χρηστών ,ερευνώντας μεγάλου μεγέθους in-memory datasets.

Το Spark λοιπόν, μπορεί να «υπερηφανεύεται» ότι παρέχει ισχυρό «integration» σε μια ποικιλία εργαλείων από το οικοσύστημα του Hadoop:

- Έχει την δυνατότητα να διαβάσει και να γράψει δεδομένα από όλα τα διαθέσιμα formats που υποστηρίζονται από το MapReduce, όπως τα Avro,Parquet,CSV.
- Επιπλέον ,χει τη δυνατότητα να διαβάζει και να γράφει σε NoSQL databases ,όπως η HBase και Cassandra.
- Η βιβλιοθήκη που αφορά στην επεξεργασία stream, η επονομαζόμενη Spark Streaming, έχει τη δυνατότητα να δέχεται δεδομένα από συστήματα όπως Flume και Kafka.
- Η βιβλιοθήκη SQL, μπορεί να αλληλεπιδράσει με το Hive Metastore.
- Τέλος,μπορεί να τρέξει τον cluster manager του Hadoop,το YARN, όπου παρέχεται υποστήριξη για τον διαμοιρασμό των πόρων δυναμικά στον cluster και να διαχειρίζεται με τις ίδιες πολιτικές παρομοίων processing engine όπως το MapReduce and Impala(Ryza, Laserson, Owen &Wills,2015:6)

Hadoop vs Spark –Η Απάντηση στην λάθος ερώτηση

Σε αντίθεση με την επικρατούσα εντύπωση, το Spark δεν δημιουργήθηκε για να αντικαταστήσει το Hadoop ούτε ότι το MapReduce έχει σταματήσει να χρησιμοποιείται. Το Spark έχει τη δυνατότητα να τρέξει πάνω από το Hadoop (αναφερόμαστε στα επίπεδα) και να επωφελείται από το cluster manager, το επονομαζόμενο YARN, καθώς και από το κατανομημένο σύστημα αποθήκευσης (HDFS, HBase, ...). Το Spark μπορεί να λειτουργήσει λοιπόν, και ξεχωριστά από το Hadoop, ενσωματώνοντας εναλλακτικούς cluster managers όπως το Mesos ,καθώς και εναλλακτικές πλατφόρμες για την αποθήκευση όπως η Cassandra και το Amazon S3.

Αυτή η λανθασμένη εντύπωση που αφορά στην σχέση του Spark με το Hadoop , χρονολογείται από τα αρχικά χρόνια της ανάπτυξης του Spark. Εκείνη την περίοδο, το Hadoop βασιζόταν στο MapReduce για την επεξεργασία του όγκου δεδομένων, όπως επίσης ήταν υπεύθυνο και για την διαχείριση και την οργάνωση των διεργασιών μέσα στους clusters. Αξίζει να σημειωθεί, πως το MapReduce διαχειριζόταν φόρτο εργασίας ακατάλληλο για batch processing , προσθέτοντας κατά αυτό τον τρόπο πολυπλοκότητα και μειωμένες επιδόσεις.

Το MapReduce είναι ένα προγραμματιστικό μοντέλο. Μέσα στο Hadoop MapReduce, πολλαπλές MapReduce διεργασίες τροποποιούνται με τέτοιο τρόπο, ώστε να γίνουν data pipeline. Μεταξύ του κάθε σταδίου του pipeline, ο κώδικας του MapReduce διαβάζει δεδομένα από τον δίσκο και όταν τελειώσει , γράφει τα δεδομένα πίσω σε αυτόν. Η διαδικασία αυτή, δεν ήταν αποδοτική καθώς έπρεπε να διαβάζει όλα τα δεδομένα από τον δίσκο σε κάθε ξεκίνημα του κάθε σταδίου . Στο σημείο αυτό το Spark «μπαίνει στο παιχνίδι», διότι παίρνει το ίδιο προγραμματιστικό μοντέλο του MapReduce αλλά με 10X άμεση αύξηση της απόδοσης, γιατί δεν χρειάζεται πλέον να αποθηκεύει τα δεδομένα πίσω στον δίσκο ,εφόσον όλες οι δραστηριότητες έμεναν στην μνήμη. Το Spark λοιπόν, προσφέρει έναν πολύ πιο γρήγορο τρόπο για την επεξεργασία δεδομένων καθώς αποφεύγονται περιττές διεργασίες. Επίσης, ο cluster manager YARN στάματησε να εξαρτιέται αποκλειστικά από το Hadoop, απελευθερώνοντας έτσι το project, επιτράπηκε η χρήση του και από το Spark. Το MapReduce είναι ακόμα διαθέσιμο μέσα στο Hadoop για την εκτέλεση «static batch processes». Άλλες διεργασίες επεξεργασίας δεδομένων μπορούν να ανατεθούν σε διαφορετικές επεξεργαστικές μηχανές (συμπεριλαμβανομένου και του Spark), χρησιμοποιώντας το YARN για την διαχείριση και την κατανομή των πόρων(από/των clusters) . Το Spark είναι μια βιώσιμη εναλλακτική του Hadoop MapReduce σε ένα μεγάλο εύρος περιστάσεων. Το Spark δεν είναι ο αντικαταστάτης του Hadoop, αλλά αντίθετα είναι ένας σπουδαίος συνεργάτης ως προς την ανάπτυξη ενός μοντέρνου Hadoop cluster.(Scott,2015:21-22)

Τι προσφέρει το Hadoop στο Spark

Το Apache Spark είναι σχεδόν παράλληλα αναπτυσσόμενο μαζί με το Hadoop cluster, και το Spark είναι σε θέση να επωφεληθεί από έναν αριθμό από δυνατότητες ως αποτέλεσμα αυτής της σχέσης. Το Spark είναι ένα ισχυρό εργαλείο για την επεξεργασία μεγάλου όγκου δεδομένων αλλά ακόμα από μόνο του δεν είναι προσαρμοσμένο καλά, για τις εργασίες στις μεγάλες επιχειρήσεις. Η ενσωμάτωση του με το Hadoop , δίνει στο Spark πολλές δυνατότητες για ευρεία υιοθέτηση και χρήση μέσα στο παραγωγικό περιβάλλον, όπου χρειάζονται τα εξής :

- YARN resource manager (διαχειριστής πόρων) ο οποίος αναλαμβάνει την ευθύνη για την χρονοδρομολόγηση των διεργασιών στους διαθέσιμους κόμβους του cluster.

- Distributed File System (κατανεμημένο σύστημα αρχείων), το οποίο αποθηκεύει τα δεδομένα όταν το cluster δεν έχει διαθέσιμη ελεύθερη μνήμη, και η οποία αποθηκεύει συνεχώς ιστορικά δεδομένα (λογικά log files) όταν Spark δεν τρέχει.
- Disaster Recovery capabilities, είναι έμφυτα στο Hadoop και επιτρέπουν την ανάκτηση των δεδομένων όταν οι επιμέρους κόμβοι αποτυγχάνουν. Οι δυνατότητες αυτές περιλαμβάνουν τις βασικές -αλλά αξιόπιστες- δυνατότητες εξόρυξης δεδομένων πέραν του κόμβου, πλούσια στιγμιότυπα (snapshot) και mirroring capabilities παρόμοια με αυτά που προσφέρονται από το MapR Data Platform.
- Data Security (ασφάλεια δεδομένων), η οποία γίνεται όλο και πιο σημαντική καθώς το Spark αντιμετωπίζει φόρτο εργασίας της παραγωγής σε οργανωμένες βιομηχανίες όπως στην υγειονομική περίθαλψη και στις οικονομικές υπηρεσίες. Έργα όπως το Apache Knox και Apache Ranger προσφέρουν δυνατότητες ασφάλειας δεδομένων κάνοντας καλύτερο κατά αυτόν τον τρόπο το Hadoop. Κάθε ένας από τους μεγάλους προμηθευτές (vendors) έχει μια εναλλακτική προσέγγιση για τις υλοποιήσεις που αφορούν την προσθήκη/συμπλήρωση ασφάλειας στο Spark. Επίσης, αναγνωρίζεται ολοένα και πιο πολύ ότι ο κώδικας στον πυρήνα του Hadoop χρειάζεται να εμπλουτιστεί με προηγμένες δυνατότητες ασφάλειας, αυξάνοντας κατά αυτόν τον τρόπο και τις παροχές ασφάλειας του Spark, αφού είναι σε θέση να τις εκμεταλλευτεί.
- A distributed data platform (κατανεμημένη πλατφόρμα δεδομένων), όπου επωφελείται από όλα τα προηγούμενα σημεία που αναφέραμε και σημαίνει ότι εργασίες του Spark μπορούν να αναπτυχθούν στους διαθέσιμους πόρους οπουδήποτε σε ένα κατανεμημένο cluster, χωρίς την ανάγκη της χειροκίνητης παρακολούθησης και τοποθέτησης για κάθε επιμέρους εργασία. (Scott,2015:22-23)

ΥΠΟΚΕΦΑΛΑΙΟ 2.3 Τα υποσυστήματα του Spark

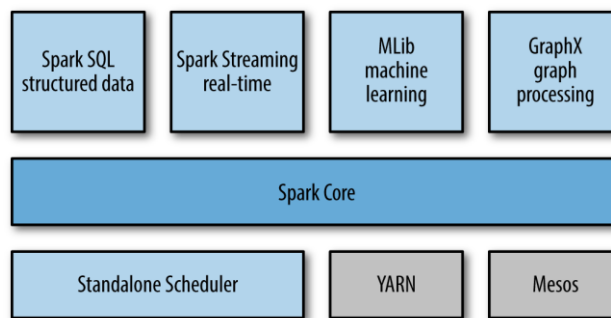
Το Spark αποτελείται από πολλαπλά στενά συνδεδεμένα συστατικά, τα οποία είναι σοφά επιλεγμένα. Στον πυρήνα του (core) Spark βρίσκεται η υπολογιστική μηχανή του, η οποία είναι υπεύθυνη για την χρονοδρομολόγηση, την διανομή και τον έλεγχο των εφαρμογών, διασπώντας τις σε πολλές υπολογιστικές εργασίες σε πολλά worker μηχανήματα, ή σε ένα σύμπλεγμα υπολογιστών. Λόγω του ότι ο πυρήνας αυτός είναι ταυτόχρονα γρήγορος και γενικού σκοπού, παρέχει δυνατότητες για ποικίλες λειτουργίες υψηλού επιπέδου (higher-level component) ειδικευμένες στην επίλυση παντός είδους εργασιών. Ένα παράδειγμα τέτοιων λειτουργιών είναι η SQL και η μηχανική μάθηση. Οι λειτουργίες αυτές, έχουν σχεδιαστεί ώστε να συνεργάζονται στενά μεταξύ τους, επιτρέποντας στους προγραμματιστές να τις συνδυάζουν σαν να ήταν απλά βιβλιοθήκες.

Η φιλοσοφία της στενής ενοποίησης (tight integration) παρέχει πολλαπλά οφέλη. Αρχικά, όλες οι βιβλιοθήκες και οι υψηλού επιπέδου λειτουργίες, που ανήκουν σε αυτήν την στοίβα(stack) επωφελούνται από τις βελτιώσεις των χαμηλότερων επιπέδων. Για παράδειγμα, όταν ο πυρήνας του Spark δέχεται βελτιστοποίηση, αυτόματα η SQL και οι βιβλιοθήκες μηχανικής μάθησης δέχονται αύξηση στις επιδόσεις τους. Δεύτερον, το κόστος που σχετίζεται με την λειτουργία της σωρού μειώνεται επειδή δεν υπάρχει η ανάγκη σε ένα οργανισμό να χρησιμοποιηθούν 5 ή 10 ανεξάρτητα συστήματα λογισμικού, παρά μόνο ένα. Το κόστος αυτό συμπεριλαμβάνει τα κόστη ανάπτυξης, συντήρησης, ελέγχου, υποστήριξης κλπ. Επίσης, κάθε φορά που μια καινούρια λειτουργία προστίθεται στην στοίβα του Spark, όλοι οι οργανισμοί που το χρησιμοποιούν έχουν

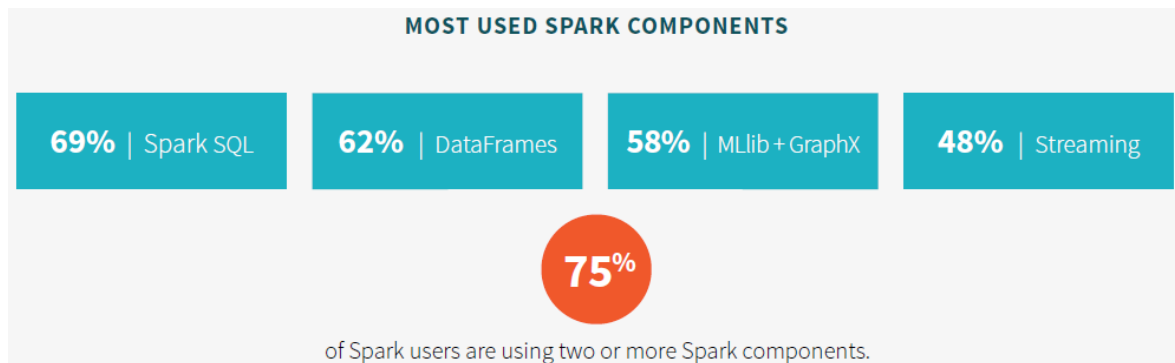
άμεσα πρόσβαση στην λειτουργία αυτή. Με αυτόν τον τρόπο αλλάζει το κόστος του κατεβάσματος, της δοκιμής και της μάθησης κάποιου νέου τύπου ανάλυσης δεδομένων για την αναβάθμιση του Spark.

Τέλος, ένα από τα μεγαλύτερα πλεονεκτήματα του tight integration, είναι η δυνατότητα της δημιουργίας εφαρμογών ,οι οποίες απρόσκοπτα συνδυάζουν διάφορα μοντέλα επεξεργασίας (processing models). Για παράδειγμα, στο Spark οι προγραμματιστές μπορούν να δημιουργήσουν μια εφαρμογή ,η οποία θα χρησιμοποιεί μηχανική μάθηση για την ταξινόμηση των δεδομένων σε πραγματικό χρόνο ,καθώς τα δεδομένα λαμβάνονται μέσω streaming. Παράλληλα ,με τα παραπάνω, οι αναλυτές θα μπορούν να θέτουν ερωτήματα στα δεδομένα που προκύπτουν, εξίσου πάλι σε πραγματικό χρόνο, μέσω της SQL. Επιπρόσθετα, πιο έμπειροι μηχανικοί και επιστήμονες δεδομένων μπορούν να έχουν πρόσβαση στα δεδομένα μέσω Python shell για ad hoc ανάλυση τους. Τέλος, όλο αυτό το διάστημα η IT ομάδα, έχει να συντηρήσει μόνο ένα σύστημα. (Karau, Konwinski, Wendell & Zaharia, 2015:17)

Τα διαθέσιμα συστήματα συνοψίζονται στα παρακάτω υπο-κεφάλαια.



Εικόνα 2.1 Τα υποσυστήματα του Spark



Εικόνα 2.2 Ποσοστά χρήσης του εκάστοτε υποσυστήματος με βάση της έρευνας του Databricks το 2015

ΥΠΟΚΕΦΑΛΑΙΟ 2.3.1 Spark Core

Στο πυρήνα του Spark (Spark core), βρίσκονται οι βασικές λειτουργίες του , συμπεριλαμβανομένων των τμημάτων υπεύθυνα για την χρονοδρομολόγηση, την διαχείριση της μνήμης , την αποκατάσταση βλαβών και την αλληλεπίδραση με τα συστήματα αποθήκευσης. Επίσης στον πυρήνα αυτόν, κατοικεί το API όπου ορίζει τα RDDs, που θεωρούνται η κύρια μορφή προγραμματιστικής αφαίρεσης του Spark. (Karau, Konwinski, Wendell & Zaharia,2015:19)

ΥΠΟΚΕΦΑΛΑΙΟ 2.3.2 Mllib

Το Spark παρέχει την βιβλιοθήκη Mllib, η οποία είναι υπεύθυνη για την υποστήριξη κοινών προβλημάτων μηχανικής μάθησης. Η Mllib παρέχει πολλαπλούς αλγόριθμους μηχανικής μάθησης συμπεριλαμβανομένων, classification, regression, clustering, και collaborative filtering, καθώς και υποστηρίζει κάποιες επιπλέον λειτουργίες όπως model evaluation, data import και lower-level ML primitives. Όλοι οι παραπάνω αλγόριθμοι και μέθοδοι είναι σχεδιασμένοι να λειτουργούν σε cluster, ανεξαρτήτου μεγέθους του cluster ή του προβλήματος που επικαλούνται να λύσουν. Περισσότερα για την ML lib αναφέρονται στο κεφάλαιο 4.

ΥΠΟΚΕΦΑΛΑΙΟ 2.3.3 Spark Streaming

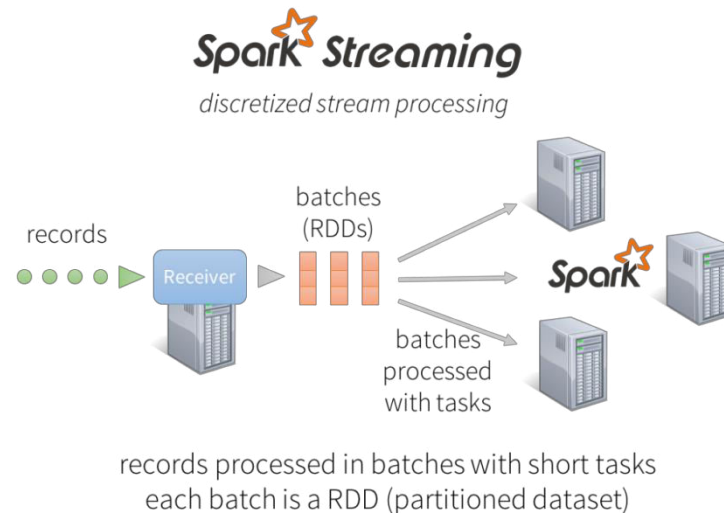
Το Spark Streaming είναι μια λειτουργία του Spark όπου επιτρέπει την επεξεργασία live streams δεδομένων. Για παράδειγμα, μερικά data streams συμπεριλαμβάνουν logfiles (αρχεία καταγραφών), τα οποία παράγονται από web servers ή ακόμα και από ουρές (queues) μηνυμάτων που περιέχουν ενημερώσεις καταστάσεων δημοσιευμένες από χρήστες μιας web υπηρεσίας. Το Spark Streaming παρέχει ένα API με σκοπό τον έλεγχο των data streams, που ταιριάζει στενά με το Spark Core's RDD API, καθιστώντας το έτσι εύκολο για τους προγραμματιστές να το μάθουν και να προσαρμόσουν άμεσα σε εφαρμογές όπου επεξεργάζονται δεδομένα στην μνήμη, στον δίσκο ή ακόμα και όταν καταφθάνουν σε πραγματικό χρόνο.

Το Spark Streaming έχει σχεδιαστεί ώστε να ικανοποιεί τις παρακάτω απαιτήσεις:

- **Fast failure and straggler recovery** - Καθώς αυξάνεται η κλίμακα, αυξάνονται και οι πιθανότητες της εμφάνισης κάποιας αστοχίας σε ένα cluster node ή ένα cluster node να καθυστερεί απρόοπτα (πχ stragglers). Το σύστημα θα πρέπει να είναι σε θέση να ανακάμψει αυτόματα από τις αστοχίες και τα stragglers, ώστε να παράξει τα δεδομένα σε πραγματικό χρόνο. Τα παραδοσιακά συστήματα δυσκολεύονται να τηρήσουν τα παραπάνω λόγω του ότι χρησιμοποιούν στατική κατανομή συνεχών operators στους worker nodes.
- **Load balancing** – Η ανομοιόμορφη κατανομή του υπο-επεξεργασία φόρτου εργασίας στους workers, μπορεί να προκαλέσει bottlenecks σε ένα continuous operator system. Οι πιθανότητες για bottlenecks αυξάνονται, όταν μεγαλώνει το πλήθος των clusters καθώς και όταν υπάρξει η ανάγκη για επεξεργασία δυναμικά μεταβαλλόμενων εργασιών. Το σύστημα πρέπει να είναι σε θέση να προσαρμόζει δυναμικά την κατανομή των πόρων με βάση το φόρτο εργασίας.
- **Unification of streaming, batch and interactive workloads** – Σε πολλές περιπτώσεις, εμφανίζεται η ανάγκη για την άσκηση διαδραστικών ερωτημάτων που αφορούν τα streaming data ή ακόμη και η ανάγκη για τον συνδυασμό τους με static datasets (e.g. pre-computed models). Στα παραδοσιακά συστήματα κάτι τέτοιο είναι ακατόρθωτο καθώς δεν είναι σχεδιασμένα να δημιουργούν δυναμικά καινούριους operators για ad-hoc queries. Για να επιτευχθούν τα παραπάνω, χρειάζεται ένα αυτόνομο λογισμικό το οποίο να μπορεί να συνδυάσει και να υποστηρίξει combine batch, streaming and interactive queries.

- *Advanced analytics όπως machine learning και SQL queries* – Οι πιο πολύπλοκοι φόρτοι εργασίας απαιτούν συνεχή μάθηση και ενημέρωση των μοντέλων δεδομένων, ή ακόμα και την άσκηση ερωτημάτων στην πιο πρόσφατη μορφή τους μέσω SQL queries. Για την επίτευξη αυτού και την διευκόλυνση της δουλειάς των προγραμματιστών, χρειάζεται μια κοινή αφαιρετική πλατφόρμα, η οποία να υποστηρίζει όλες αυτές τις λειτουργίες.

Για την ικανοποίηση των παραπάνω απαιτήσεων, το Spark Streaming χρησιμοποιεί μια νεοσύστατη αρχιτεκτονική με όνομα discretized streams, η οποία αξιοποιεί άμεσα τις πλούσιες βιβλιοθήκες και την ανοχή στα σφάλματα του Spark engine.



Εικόνα 2.3 Το μοντέλο του Spark Streaming

Το Spark Streaming χρησιμοποιώντας την αρχιτεκτονική αυτή, αντί να επεξεργάζεται ένα-ένα τις εγγραφές των streaming data, διασπά τα streaming data σε μικρά, sub-second micro-batches. Με άλλα λόγια, οι παραλήπτες (Receivers) του Spark Streaming δέχονται τα δεδομένα παράλληλα και τα κάνουν buffer στην μνήμη των workers nodes. Αμέσως το latency-optimized Spark engine εκτελεί μικρές διεργασίες, διαρκούν κλάσματα του δευτερόλεπτου, για την επεξεργασία των batches και παράγει τα αποτελέσματα για την αποστολή τους στα άλλα συστήματα. Αξίζει να αναφερθεί ότι σε αντίθεση με το παραδοσιακό μοντέλο « continuous operator», όπου ο υπολογισμός διατίθεται στατικά σε έναν κόμβο, οι διεργασίες στο Spark διατίθενται δυναμικά στους workers, με βάση τους διαθέσιμους πόρους και την locality of the data. Αυτό επιτρέπει την καλύτερη εξισορρόπηση και την ταχύτερη αποκατάσταση βλάβης του φόρτου εργασίας.

Τέλος, κάθε παρτίδα δεδομένων είναι ένα Resilient Distributed Dataset (RDD), επιτρέποντας έτσι την επεξεργασία των streaming data χρησιμοποιώντας οποιοδήποτε κώδικα ή βιβλιοθήκες του Spark. (<https://databricks.com/blog/2015/07/30/diving-into-apache-spark-streamings-execution-model.html>)

ΥΠΟΚΕΦΑΛΑΙΟ 2.3.4 Spark SQL

Το Spark SQL είναι το πακέτο του Spark, που παρέχει την δυνατότητα εργασίας με δομημένα δεδομένα (structured data). Επιτρέπει την διεξαγωγή ερωτημάτων στα δεδομένα, χρησιμοποιώντας την SQL καθώς και την αντίστοιχη παραλλαγή της SQL για το Apache Hive, την επονομαζόμενη Hive Query Language (HQL). Το Spark SQL υποστηρίζει πολλές πηγές δεδομένων, συμπεριλαμβανομένου Hive tables, Parquet, και JSON. Πέραν της παροχής μιας SQL διεπαφής

για το Spark, το Spark SQL επιτρέπει στους προγραμματιστές να συνδυάσουν SQL ερωτήματα με προγραμματιστικούς χειρισμούς δεδομένων που υποστηρίζονται από τα RDDs στην Python, Java, και Scala. Αυτή η ιδιότητα πραγματοποιείται σε μια μόνο εφαρμογή, συνδυάζοντας έτσι την SQL με complex analytics. Λόγο της στενής ενοποίησης της, με το πλούσιο υπολογιστικό περιβάλλον του Spark, η Spark SQL καθίσταται μοναδική σε σχέση με άλλα εργαλεία ανοικτού κώδικα.

Ακολουθούν οι δυνατότητες και τα χαρακτηριστικά του Spark SQL:

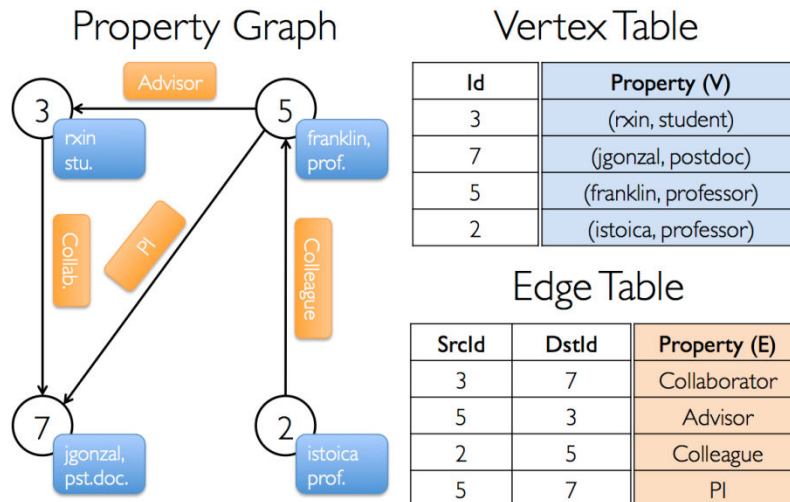
- **Integrated** – Απρόσκοπτα οι προγραμματιστές μπορούν να συνδυάσουν ερωτήματα SQL μέσα στα Spark προγράμματά τους. Η Spark SQL επιτρέπει να τεθούν ερωτήματα σε δομημένα δεδομένα, όπως είναι τα RDD, μέσω ολοκληρωμένων APIs στην Python, Scala και Java. Το tight integration καθιστά εύκολη την εκτέλεση SQL ερωτημάτων παράλληλα με πολύπλοκους αλγορίθμους ανάλυσης δεδομένων.
- **Unified Data Access** – Ο προγραμματιστής έχει την δυνατότητα να φορτώσει και να κάνει ερωτήσεις σε δεδομένα από ποικίλες πηγές. Τα Schema-RDDs παρέχουν μια απλή διεπαφή για την αποτελεσματική εργασία σε δομημένα δεδομένα, συμπεριλαμβανομένων τα Apache Hive tables, parquet files and JSON files.
- **Hive Compatibility** – Η Spark SQL παρέχει πλήρη υποστήριξη στην χρήση των ήδη υπάρχοντων Hive queries, χωρίς να υπάρχει η ανάγκη των μετασχηματισμών τους. Με αυτόν τον τρόπο αν οι προγραμματιστές εγκαταστήσουν Spark SQL παράλληλα με το Hive στα συστήματά τους, η πρώτη θα χρησιμοποιήσει τα Hive frontend και MetaStore, παρέχοντας με αυτόν τον τρόπο πλήρη υποστήριξη στα ήδη υπάρχοντα Hive data, queries, and UDFs.
- **Standard Connectivity** – Επιτρέπει την σύνδεση χρησιμοποιώντας JDBC(Java Database Connectivity) ή και ODBC (Open Database Connectivity).
- **Scalability** – Οι προγραμματιστές χρησιμοποιούν την ίδια βάση τόσο για διαδραστικά όσο και για μεγάλα ερωτήματα. Το Spark SQL εκμεταλλεύεται το μοντέλο των RDDs, θέτοντας έτσι ασφάλεια και μη απώλεια δεδομένων κατά την διάρκεια των ερωτημάτων, με την δυνατότητα αυτή η λειτουργία να κλιμακώνεται και σε πιο απαιτητικότερες και μεγαλύτερες εργασίες.
(https://www.tutorialspoint.com/spark_sql/spark_sql_introduction.htm)

ΥΠΟΚΕΦΑΛΑΙΟ 2.3.5 GraphX

Η GraphX είναι μια βιβλιοθήκη για τον χειρισμό γραφημάτων (για παράδειγμα ένα social network's friend graph) και εκτελεί graph-parallel computations. Η GraphX επεκτείνει το Spark RDD API, όπως τα Spark Streaming και Spark SQL, επιτρέποντάς μας να δημιουργήσουμε κατευθυνόμενα γραφήματα με την παρουσία αυθαίρετων ιδιοτήτων σε κάθε κορυφή και ακμή τους (Resilient Distributed Property Graph). Τέλος, η GraphX παρέχει ποικίλους operators για τον χειρισμό και την επεξεργασία γραφημάτων, για παράδειγμα subgraph και mapVertices, καθώς και μια βιβλιοθήκη κοινών αλγορίθμων για γραφήματα όπως είναι η PageRank και το triangle counting.

Τα property graphs είναι κατευθυνόμενα πολύ-γραφήματα, με τα αντικείμενά τους να είναι ορισμένα από τον χρήστη και να είναι επισυναπτόμενα σε κάθε κόμβο και κορυφή. Τα

κατευθυνόμενα πολύ-γραφήματα είναι παρόμοια με τα κατευθυνόμενα γραφήματα με την διαφορά ότι υπάρχει η δυνατότητα για πολλαπλές παράλληλες ακμές να μοιράζονται τον ίδιο πηγαίο και καταληκτικό κόμβο. Η δυνατότητα αυτή, απλοποιεί τα σενάρια που χρειάζεται να μοντελοποιήσουμε πολλαπλές σχέσεις μεταξύ των κορυφών. Κάθε κορυφή αποκτά ένα μοναδικό κλειδί μήκους 64-bit για την ταυτοποίηση του. Τέλος, η GraphX βελτιστοποιεί την απεικόνιση των ακμών και κόμβων, όταν βασίζονται σε απλούς τύπους δεδομένων, μειώνοντας το «αποτύπωμα» τους στην μνήμη, με την αποθήκευση τους σε ειδικούς πίνακες.



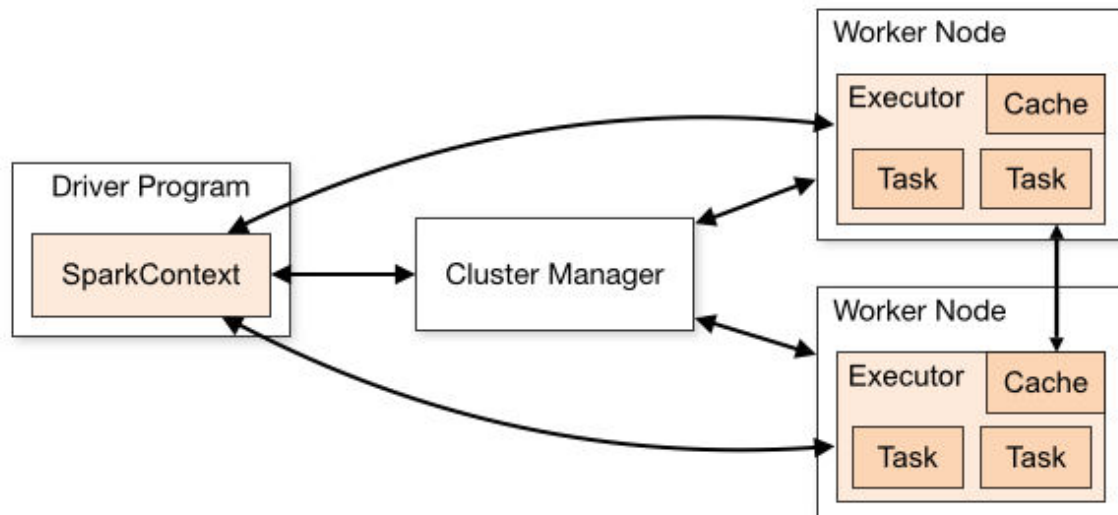
Εικόνα 2.4 Το μοντέλο του GraphX

Παρομοίως με τα RDD, τα property graphs είναι αμετάβλητα, διανεμόμενα, και ανεκτικά στα σφάλματα. Οι αλλαγές των τιμών ή στην δομή του γραφήματος, φέρουν ως αποτέλεσμα την δημιουργία ενός νέου γραφήματος, το οποίο περιέχει τις αλλαγές αυτές. Αξίζει να σημειωθεί, πως τα σημαντικά και ουσιώδη τμήματα του πρωτότυπου γραφήματος (πχ τμήματα της δομής που δεν άλλαξαν, attributes, και indices) επαναχρησιμοποιούνται στο καινούριο και μόλις δημιουργημένο γράφημα, με αποτέλεσμα την μείωση του κόστους (χρόνος, μέγεθος). Τα γραφήματα διασπώνται και μοιράζονται στους workers, χρησιμοποιώντας ένα πλήθος από vertex-partitioning heuristics. Κάθε τμήμα του γραφήματος μπορεί να αναδημιουργηθεί σε ένα διαφορετικό μηχάνημα στην περίπτωση απώλειας δεδομένων ή βλάβης. Την ιδιότητα αυτή την μοιράζονται και τα RDDs.

Πριν την διανομή του GraphX, ο υπολογισμός γραφημάτων στο Spark εκφραζόταν χρησιμοποιώντας την Bagel, η οποία είναι μια υλοποίηση της Pregel. Το GraphX βελτιώνει την Bagel με την διανομή ενός πλουσιότερου API για γραφήματα, μιας πιο σύγχρονης και βελτιστοποιημένη εκδοχής της αφαιρετικότητας της Bagel, καθώς και με την παροχή βελτιστοποιήσεων στο σύστημά της με σκοπό την αύξηση των επιδόσεων και την μείωση χρήσης της κυρίας μνήμης. Το Spark συνεχίζει να υποστηρίζει την Bagel αλλά στο μέλλον αυτή η υποστήριξη θα πάψει να υφίσταται. (<https://spark.apache.org/docs/0.9.0/graphx-programming-guide.html>)

ΥΠΟΚΕΦΑΛΑΙΟ 2.3.4 Cluster Managers και η δομή των κόμβων

Το παρακάτω διάγραμμα που υπάρχει στην ιστοσελίδα του spark.apache.org, παρουσιάζει τον ρόλο του Apache Spark cluster manager με τους όρους του master, worker node, executor:



Εικόνα 2.5 Ο «σκελετός» του Spark

Spark Context

Το Spark context ανήκει στο driver πρόγραμμα, δηλαδή αυτό που έχει φτιάξει και διαχειρίζεται ο προγραμματιστής, και συνδέεται με τον επιλεγμένο cluster manager, ο οποίος στην συνέχεια κατανέμει τους πόρους της εφαρμογής σε όλα τα worker nodes. Μόλις πραγματοποιηθεί η σύνδεση, δημιουργούνται οι executors στους κόμβους του cluster. Οι executors, θεωρούνται διαδικασίες οι οποίες πραγματοποιούν υπολογισμούς και αποθηκεύουν τα δεδομένα της εκάστοτε εφαρμογής. Το επόμενο βήμα είναι η αποστολή του κώδικα της εφαρμογής στους executors (ο κώδικας μπορεί να είναι είτε JAR είτε αρχεία Python). Το τελευταίο βήμα είναι η αποστολή των διεργασιών από το SparkContext στους executors για την εκτέλεση τους. (Frampton, 2015:8)

Επιλογή cluster manager

Μέχρι στιγμής οι προγραμματιστές έχουν την δυνατότητα να επιλέξουν ανάμεσα σε 3 Cluster Managers:

- Standalone, ένας απλός cluster manager ο οποίος συμπεριλαμβάνετε μαζί με το Spark.
- Apache Mesos, ένας γενικής χρήσης cluster manager οπότε μπορεί να εκτελέσει το MapReduce του Hadoop καθώς και εφαρμογές υπηρεσιών.
- Hadoop Yarn, ο βασικός διαχειριστής πόρων στο Hadoop 2.

Αν ο προγραμματιστής δεν έχει προηγούμενη εμπειρία είναι πιο συνετό, να ξεκινήσει με τον standalone cluster στην περίπτωση που θα χρησιμοποιήσει μόνο το Spark. Το Standalone mode είναι το πιο εύκολο να γίνει setup και παρέχει σχεδόν τα ίδια χαρακτηριστικά και δυνατότητες που παρέχουν οι άλλοι cluster managers. Σε αντίθετη περίπτωση, αν ο προγραμματιστής θέλει να λειτουργήσει το Spark παράλληλα με άλλες εφαρμογές ή να θέλει να χρησιμοποιήσει πλουσιότερες δυνατότητες χρονοδρομολόγησης των πόρων (πχ queues) και το Yarn και το Mesos παρέχουν αυτές τις δυνατότητες/χαρακτηριστικά, με το Yarn να έχει περισσότερες πιθανότητες να είναι προ-εγκατεστημένο σε πολλές διανομές του Hadoop.

Ένα πλεονέκτημα του Mesos σε σχέση με το Yarn και το Standalone mode είναι η ύπαρξη του «fine-grained sharing option», το οποίο επιτρέπει στις διαδραστικές εφαρμογές, όπως το Spark Shell, να περιορίσουν την κατανομή τους ανάμεσα στις εντολές μέσα στην CPU. Η λειτουργία αυτή το καθιστά ελκυστικό για τις περιπτώσεις όπου πολλαπλοί χρήστες τρέχουν διαδραστικά shells.

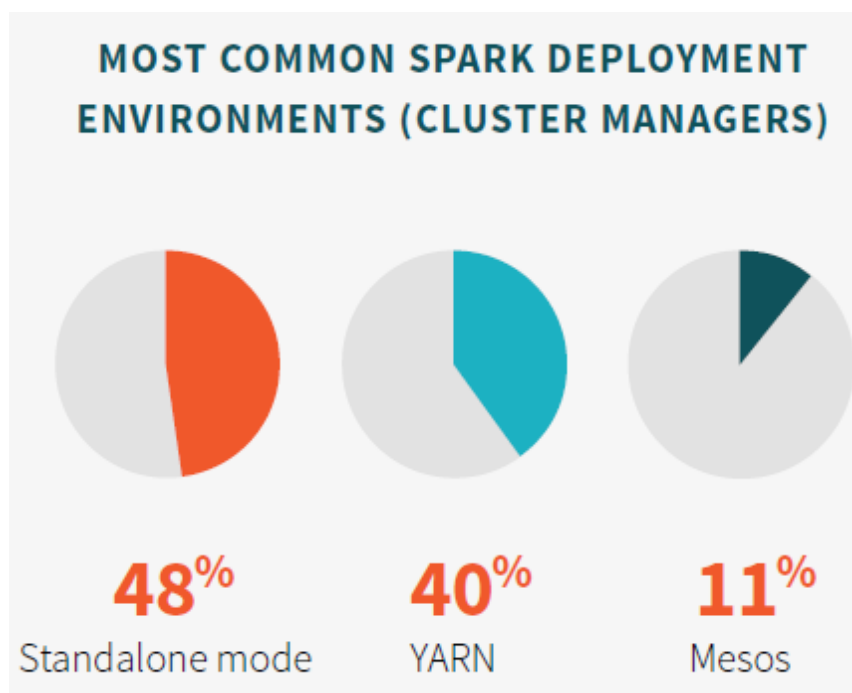
Το Spark Standalone Manager ανταποκρίνεται ικανοποιητικά μόνο για μικρού μεγέθους clusters (single digit nodes). Αντίθετα ο YARN ή ο Mesos είναι επιθυμητοί για μεγαλύτερου μεγέθους clusters.

Μερικά πλεονεκτήματα του YARN σε σχέση με το Standalone και το Mesos:

1. Το YARN επιτρέπει στον προγραμματιστή, να μοιράζεται δυναμικά και να ρυθμίζει με ευκολία το ίδιο σύνολο πόρων ενός cluster, μεταξύ όλων των frameworks που τρέχουν στο YARN.
2. Οι προγραμματιστές έχουν τη δυνατότητα να εκμεταλλευτούν όλες τις παροχές των YARN schedulers που αφορούν στην κατηγοριοποίηση (categorizing), απομόνωση (isolating), και ιεράρχηση (prioritizing) των φόρτων εργασίας.
3. Το Spark standalone mode απαιτεί από κάθε εφαρμογή να τρέξει από έναν executor σε κάθε node του cluster, ενώ με το YARN ο προγραμματιστής μπορεί να επιλέξει τον αριθμό των executors που θέλει να χρησιμοποιήσει.
4. Ο YARN είναι ο μόνος cluster manager του Spark που υποστηρίζει λύσεις ασφαλείας. Με το YARN, το Spark είναι σε θέση να ανταγωνιστεί clusters βασισμένους στο Hadoop Kerberos και να χρησιμοποιεί secure authentication μεταξύ των διεργασιών.
5. Ο YARN είναι υπεύθυνο και αναλαμβάνει να τηρηθούν «rack και machine locality» στα αιτήματα (request)s, το οποίο είναι βολικό.
6. Το μοντέλο του YARN είναι λιγότερο ευέλικτο, σε αντίθεση με το Mesos, αλλά χρειάζεται λιγότερο κόπος και χρόνος για τους προγραμματιστές, να υλοποιήσουν το framework του.
7. Τέλος, αν οι προγραμματιστές χρησιμοποιούν ένα μεγάλο Hadoop cluster παράλληλα με το Spark, ο YARN είναι η προτιμότερη λύση.
(<http://stackoverflow.com/questions/28664834/which-cluster-type-should-i-choose-for-spark>)

Mesos	Yarn
Written in C++, good for time sensitive works	Written in Java, JVM based app
Memory & CPU scheduling – Push based	Mainly memory scheduling – Pull based
Use Linux Container groups	Use Simple Unix processes
Framework get Resource offer to choose – very minimal information as just needed	Framework ask a container with specification + preferences(like local). Lots of information passed
Core Mesos is lighter but one need to write a Framework	It's a Framework of its own and so its 3x code vs.Mesos
Mesos, you need to deal with the security	Yarn inherit Hadoop security
Mesos is general purpose scheduler for Data Center. Application writer deploy applications the way wanted	Mainly exists on Hadoop world – it's a application scheduler. Framework setup unix process/application
The Framework takes care the application specific items	Can supports clustered applications
Fault tolerance, app portability, etc. - the Framework has to deal with	Can enforce global constraints and local – so application can deploy to right place
High performance Actor Style Messaging passing	Hadoop RPC Architecture – direction piggy backs heart beat
Lower level abstraction	Can run Yarn on Mesos (Myriad)

Εικόνα 2.6 Οι βασικές διαφορές ανάμεσα στους cluster managers Yarn και Mesos



Εικόνα 2.7 Ποσοστά χρήσης των εκάστοτε Cluster Manager με βάση την έρευνα του Databricks το 2015

Worker Nodes και Executors

Executors ονομάζονται οι διεργασίες (processes) των worker κόμβων, και είναι υπεύθυνοι για την εκτέλεση των επιμέρους εργασιών από μια καθορισμένη δουλειά του Spark. Η αρχικοποίηση τους πραγματοποιείται κατά την εκκίνηση μιας εφαρμογής του Spark και τυπικά θα συνεχίζουν να υφίστανται καθ' όλη την διάρκεια ζωής αυτής της εφαρμογής. Όταν ολοκληρώσουν την εργασία την οποία έχουν αναλάβει, αποστέλλουν τα αποτελέσματα στο driver πρόγραμμα.

Επίσης, παρέχουν χώρο αποθήκευσης στην κύρια μνήμη τους, για την αποθήκευση των RDDs, τα οποία αποθηκεύονται προσωρινά από τα προγράμματα του χρήστη μέσω του Block Manager.

Όταν δημιουργηθούν οι executors, καταγράφουν τους εαυτούς τους μέσα στο driver πρόγραμμα, ώστε από εδώ και στο εξής να έχουν άμεση επικοινωνία με αυτό. Οι workers είναι υπεύθυνοι για την επικοινωνία με τον cluster manager, για την διαθεσιμότητα των πόρων τους. (<http://spark.apache.org/docs/latest/cluster-overview.html>)

ΥΠΟΚΕΦΑΛΑΙΟ 2.4 Web Interfaces

Κάθε ένα SparkContext δημιουργεί ένα web UI (User Interface), με προεπιλεγμένη στην πύλη 4040, και εμφανίζει χρήσιμες πληροφορίες που αφορούν την εφαρμογή. Οι διαθέσιμες πληροφορίες είναι οι εξής:

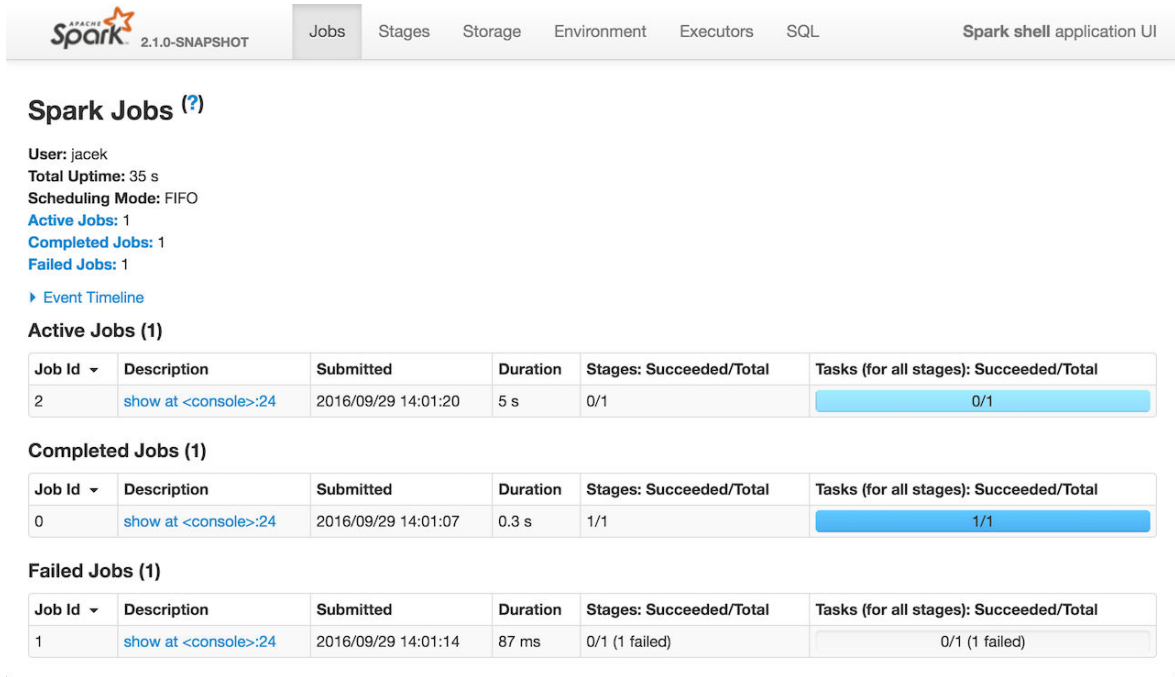
- Μια λίστα με τα «scheduler stages» και «tasks».
- Μια σύνοψη των διαστάσεων των RDDs καθώς και την χρήση μνήμης του συστήματος.
- Πληροφορίες περιβάλλοντος.
- Πληροφορίες σχετικά με τους ενεργούς executors.

Οι προγραμματιστές έχουν πρόσβαση σε αυτήν την διεπαφή (interface), πληκτρολογώντας απλά `http://<driver-node>:4040` σε κάποιον web browser. Στην περίπτωση που πολλαπλά SparkContexts είναι ενεργά σε κάποιον κοινό κόμβο (node), τότε θα δεσμεύσουν διαδοχικές ports ξεκινώντας από την 4040 (4041, 4042, κτλ). (<https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-webui.html>)

Το Web UI παρέχει τα εξής tabs:

Jobs

Στην παρούσα καρτέλα εμφανίζεται η κατάσταση και τα στοιχεία όλων των διεργασιών (jobs) που ανήκουν σε μια εφαρμογή του Spark.



Spark Jobs (?)

User: jacek
 Total Uptime: 35 s
 Scheduling Mode: FIFO
 Active Jobs: 1
 Completed Jobs: 1
 Failed Jobs: 1
[Event Timeline](#)

Active Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	show at <console>:24	2016/09/29 14:01:20	5 s	0/1	0/1

Completed Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	show at <console>:24	2016/09/29 14:01:07	0.3 s	1/1	1/1

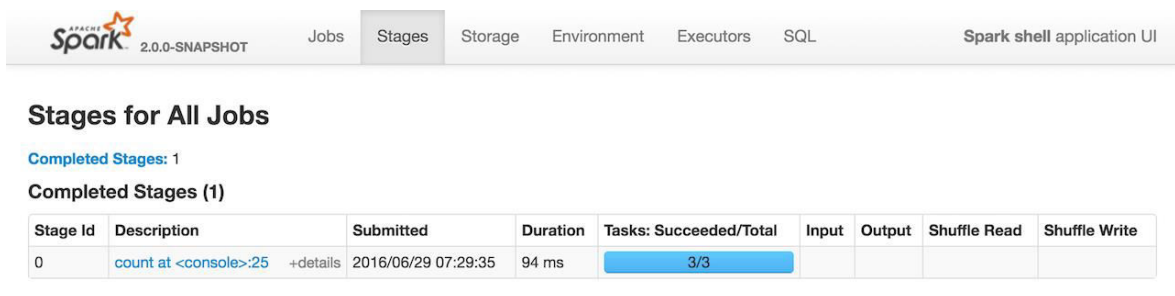
Failed Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1	show at <console>:24	2016/09/29 14:01:14	87 ms	0/1 (1 failed)	0/1 (1 failed)

Εικόνα 2.7 Καρτέλα Jobs

Stages

Στην καρτέλα «Stages» εμφανίζεται η τρέχουσα κατάσταση όλων των σταδίων, όλων των διεργασιών που ανήκουν σε μια εφαρμογή. Υπάρχει η δυνατότητα για την ύπαρξη δυο προαιρετικών σελίδων: μια που παρουσιάζει τις διεργασίες και κάποια στατιστικά στοιχεία για το κάθε βήμα και άλλη μια που παρουσιάζει τον πίνακα με τα scheduler pools, όταν υπάρχει εφαρμογή που βασίζεται σε FAIR scheduling mode.



Stages for All Jobs

Completed Stages: 1

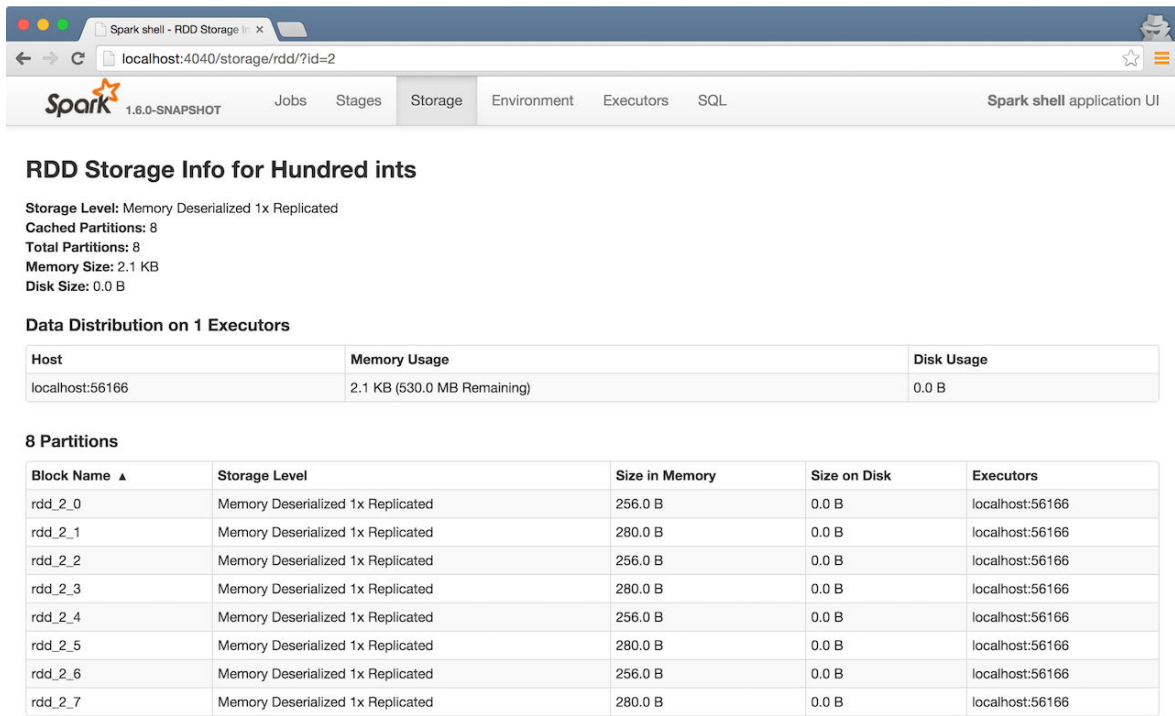
Completed Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
0	count at <console>:25 +details	2016/06/29 07:29:35	94 ms	3/3				

Εικόνα 2.8 Καρτέλα Stages

Storage

Στην παρούσα καρτέλα εμφανίζονται τα τμήματα του RDD, καθώς και τα μεγέθη που καταλαμβάνουν στην κύρια μνήμη και στον δίσκο.



RDD Storage Info for Hundred ints

Storage Level: Memory Deserialized 1x Replicated
 Cached Partitions: 8
 Total Partitions: 8
 Memory Size: 2.1 KB
 Disk Size: 0.0 B

Data Distribution on 1 Executors

Host	Memory Usage	Disk Usage
localhost:56166	2.1 KB (530.0 MB Remaining)	0.0 B

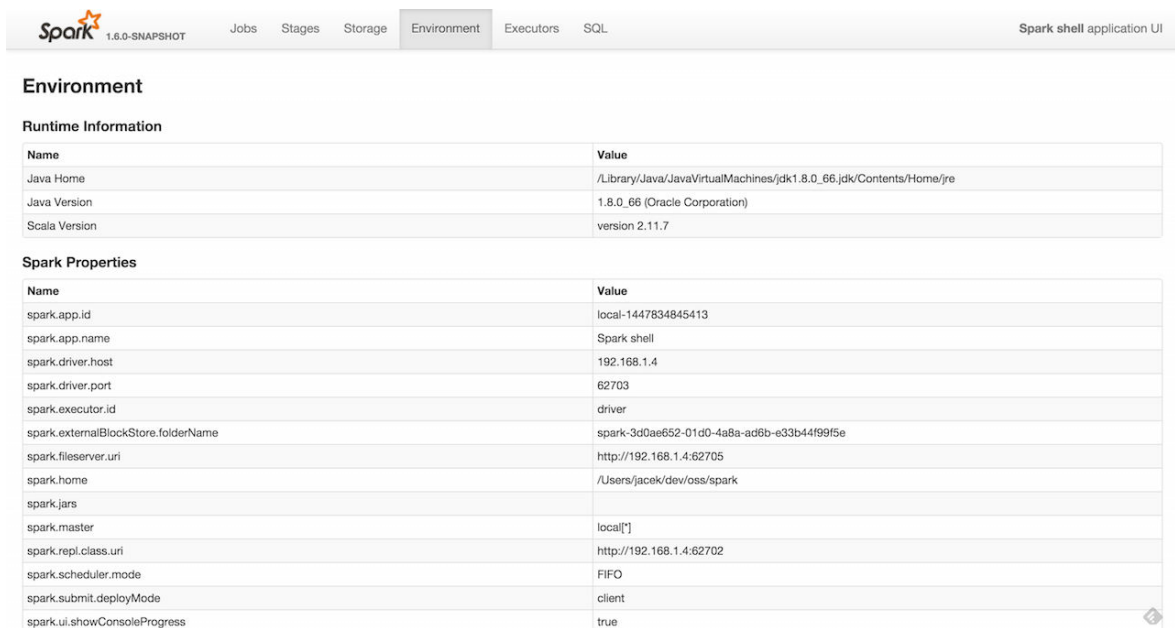
8 Partitions

Block Name	Storage Level	Size in Memory	Size on Disk	Executors
rdd_2_0	Memory Deserialized 1x Replicated	256.0 B	0.0 B	localhost:56166
rdd_2_1	Memory Deserialized 1x Replicated	280.0 B	0.0 B	localhost:56166
rdd_2_2	Memory Deserialized 1x Replicated	256.0 B	0.0 B	localhost:56166
rdd_2_3	Memory Deserialized 1x Replicated	280.0 B	0.0 B	localhost:56166
rdd_2_4	Memory Deserialized 1x Replicated	256.0 B	0.0 B	localhost:56166
rdd_2_5	Memory Deserialized 1x Replicated	280.0 B	0.0 B	localhost:56166
rdd_2_6	Memory Deserialized 1x Replicated	256.0 B	0.0 B	localhost:56166
rdd_2_7	Memory Deserialized 1x Replicated	280.0 B	0.0 B	localhost:56166

Εικόνα 2.9 Καρτέλα Storage

Environment

Στην παρούσα καρτέλα εμφανίζονται πληροφορίες για την έκδοση και την «διαδρομή» της Java, Scala και Python, καθώς και κάποιες ιδιότητες του Spark (πχ spark.app.name με τιμή «Spark shell») με τις αντίστοιχες τιμές τους.



Environment

Runtime Information

Name	Value
Java Home	/Library/Java/JavaVirtualMachines/jdk1.8.0_66.jdk/Contents/Home/jre
Java Version	1.8.0_66 (Oracle Corporation)
Scala Version	version 2.11.7

Spark Properties

Name	Value
spark.app.id	local-1447834845413
spark.app.name	Spark shell
spark.driver.host	192.168.1.4
spark.driver.port	62703
spark.executor.id	driver
spark.externalBlockStore.folderName	spark-3d0ae652-01d0-4a8a-ad6b-e33b44f99f5e
spark.fileserver.uri	http://192.168.1.4:62705
spark.home	/Users/jacek/dev/oss/spark
spark.jars	
spark.master	local[*]
spark.repl.class.uri	http://192.168.1.4:62702
spark.scheduler.mode	FIFO
spark.submit.deployMode	client
spark.ui.showConsoleProgress	true

Εικόνα 2.10 Καρτέλα Enviroment

Executors

Στην καρτέλα αυτήν εμφανίζονται οι ενεργοί και ολοκληρωμένοι Executors καθώς και πληροφορίες για αυτούς, όπως για παράδειγμα χρόνος εκτέλεσης. Παρέχεται και η δυνατότητα αναζήτησης τους, για την διευκόλυνση της εύρεσης τους αν είναι πολυπληθείς.

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(2)	10	40.4 KB / 1.9 GB	1.2 KB	2	0	0	4	4	1 s (48 ms)	0.0 B	0.0 B	0.0 B
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(2)	10	40.4 KB / 1.9 GB	1.2 KB	2	0	0	4	4	1 s (48 ms)	0.0 B	0.0 B	0.0 B

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	Thread Dump
driver	192.168.1.4:49478	Active	4	20.2 KB / 956.6 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump
0	192.168.1.4:49484	Active	6	20.2 KB / 956.6 MB	1.2 KB	2	0	0	4	4	1 s (48 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump

Showing 1 to 2 of 2 entries [Previous](#) [1](#) [Next](#)

Εικόνα 2.11 Καρτέλα Executors

SQL

Είναι η τελευταία καρτέλα του Spark Web UI, και παρέχει πληροφορίες για SQL ερωτήματα όπως για παράδειγμα περιγραφή του εκάστοτε ερωτήματος, ημερομηνίας και ώρα υποβολής κλπ. Τα ερωτήματα χωρίζονται σε τρεις κατηγορίες: υπο-εκτέλεση (Running Queries), ολοκληρωμένα (Completed Queries), και ανεπιτυχή (Failed Queries).

SQL

Running Queries

ID	Description	Submitted	Duration	Running Jobs	Succeeded Jobs	Failed Jobs
2	foreach at <console>:24 +details	2016/06/29 22:30:45	2 s	1		

Completed Queries

ID	Description	Submitted	Duration	Jobs
0	show at <console>:24 +details	2016/06/29 22:29:46	19 ms	

Failed Queries

ID	Description	Submitted	Duration	Succeeded Jobs	Failed Jobs
1	foreach at <console>:24 +details	2016/06/29 22:30:02	0.9 s		0

Εικόνα 2.12 Καρτέλα SQL

ΚΕΦΑΛΑΙΟ 3 : Υλοποίηση Εφαρμογών σε Scala και Python

ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο παρουσιάζονται οι δυο πιο δημοφιλείς γλώσσες προγραμματισμού που χρησιμοποιούνται για την δημιουργία εφαρμογών στο Spark, παράλληλα με την σύγκριση των πλεονεκτημάτων τους, για την διευκόλυνση της επιλογής. Τέλος παρουσιάζεται το προγραμματιστικό μοντέλο που ακολουθείται από το Spark.

ΥΠΟΚΕΦΑΛΑΙΟ 3.1 Η γλώσσα Scala

Η Scala αποτελεί μια γλώσσα προγραμματισμού γενικού σκοπού και παρέχει πλήρη υποστήριξη για συναρτησιακό προγραμματισμό (functional programming) και ανήκει στις γλώσσες που παρέχουν ισχυρό στατικό τύπο δεδομένων. Έχει σχεδιαστεί, ώστε να είναι συνοπτική και περιεκτική καθώς, οι επιλογές που πάρθηκαν κατά την διάρκεια του σχεδιασμού της είναι εμπνευσμένες από την κριτική που έχει ασκηθεί στην Java για τις ελλείψεις της.

Το όνομα Scala πηγάζει από τον συνδυασμό των λέξεων “scalable” και “language”, δηλώνοντας ότι είναι σχεδιασμένη να αναπτύσσεται με βάση τις ανάγκες των χρηστών της. Η Scala είναι μια μοντέρνα «multi-paradigm» γλώσσα προγραμματισμού, σχεδιασμένη να εκφράζει κοινά προγραμματιστικά πρότυπα με έναν συνοπτικό, κομψό και «type-safe» τρόπο. Η Scala δημιουργήθηκε από τον Martin Odersky ο οποίος διέθεσε την πρώτη έκδοσή της το 2003. Η Scala ενσωματώνει με εύκολο τρόπο τα χαρακτηριστικά και τις δυνατότητες των αντικειμενοστραφών και συναρτησιακών γλωσσών προγραμματισμού.

Ο πηγαίος κώδικας της Scala προορίζεται για να γίνει compile σε Java bytecode, με αποτέλεσμα ο παραγόμενος εκτελέσιμος κώδικας να τρέχει πάνω σε Java virtual machine. Οι βιβλιοθήκες της Java μπορούν να χρησιμοποιηθούν απευθείας σε κώδικα Scala όπως και το αντίστροφο (language interoperability). Τόσο η Java, όσο και η Scala είναι αντικειμενοστραφείς (object-oriented) και χρησιμοποιούν σύνταξη με αγκύλες (curly-brace) θυμίζοντας την σύνταξη της προγραμματιστικής γλώσσας C. Σε αντίθεση με την Java, η Scala έχει κληρονομήσει χαρακτηριστικά/δυνατότητες σχεσιακού προγραμματισμού από γλώσσες όπως: Scheme, Standard ML και Haskell, including currying, type inference, immutability, lazy evaluation, and pattern matching. Επίσης παρέχει ένα προχωρημένο type system supporting algebraic data types, covariance και contravariance, higher-order types (αλλά όχι higher-rank types), και anonymous types. Τέλος, κάποια άλλα χαρακτηριστικά της Scala τα οποία δεν εμφανίζονται στην Java είναι : operator overloading, optional parameters, named parameters, raw strings, and no checked exceptions. (<http://www.scala-lang.org/>)

ΥΠΟΚΕΦΑΛΑΙΟ 3.2 Η γλώσσα Python

Η Python είναι μια υψηλού επιπέδου γλώσσα προγραμματισμού, η οποία δημιουργήθηκε από τον Ολλανδό Γκβίντο βαν Ρόσσουμ (Guido van Rossum) το 1990. Κύριος στόχος της, είναι η αναγνωσιμότητα του κώδικά της και η ευκολία χρήσης της. Το συντακτικό της επιτρέπει στους προγραμματιστές να εκφράσουν έννοιες σε λιγότερες γραμμές κώδικα απ'ότι θα ήταν δυνατόν σε γλώσσες όπως η C++ ή η Java.

Παρατίθεται ένα απλό παράδειγμα σύγκρισης μεταξύ Java και Python, για το πόσες γραμμές κώδικα χρειάζεται για την ανάγνωση και εκτύπωση ενός αρχείου txt:

Κώδικας σε Java:

```
1 File dir = new File(".");// get current directory
2 File fin = new File(dir.getCanonicalPath() + File.separator
3               + "Code.txt");
4 FileInputStream fis = new FileInputStream(fin);
5 // //Construct the BufferedReader object
6 BufferedReader in = new BufferedReader(new InputStreamReader(fis));
7 String aLine = null;
8 while ((aLine = in.readLine()) != null) {
9     // //Process each line, here we count empty lines
10    if (aLine.trim().length() == 0) {
11    }
12 }
13
14 // do not forget to close the buffer reader
15 in.close();
16
```

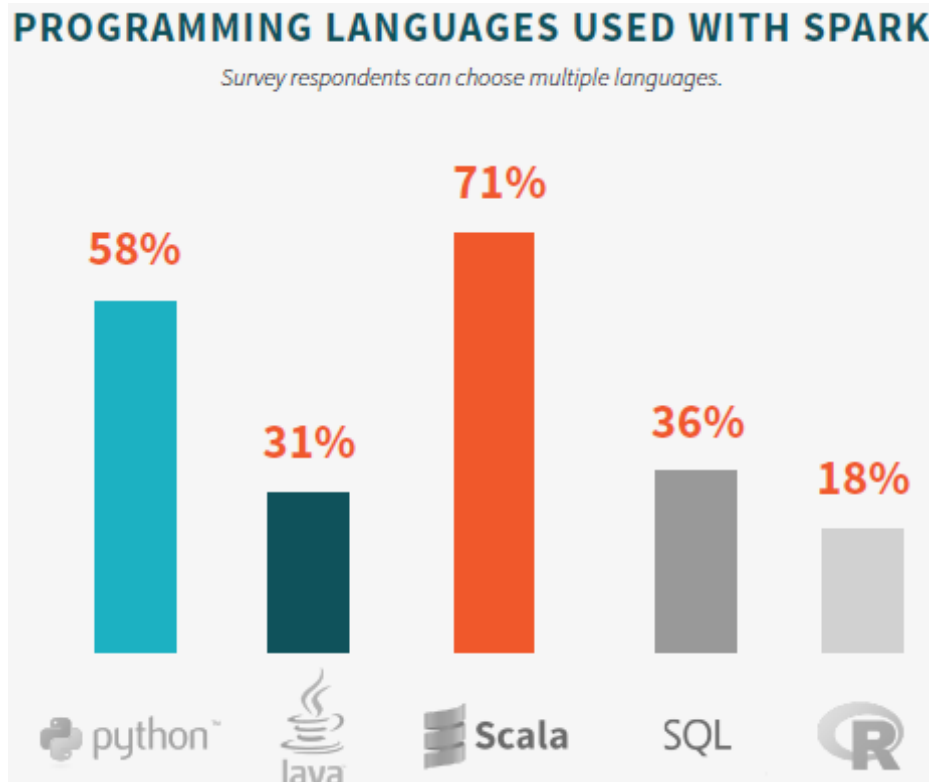
Κώδικας σε Python:

```
1 myFile = open("/home/xiaoran/Desktop/test.txt")
2 print myFile.read();
```

Τέλος η Python ξεχωρίζει, λόγω του ότι διαθέτει πολλές βιβλιοθήκες που διευκολύνουν ιδιαίτερα αρκετές εργασίες, καθώς ο χρήστης δεν αναλώνει τον χρόνο του σε θέματα που αφορούν το style και την στοίχιση του κώδικα, διότι στην Python δεν χρησιμοποιούνται άγκιστρα που περικλείουν κώδικα. (<http://www.programcreek.com/2012/04/java-vs-python-why-python-can-be-more-productive>)

ΥΠΟΚΕΦΑΛΑΙΟ 3.3 Scala ή Python

Όπως ήδη έχει αναφερθεί το Spark υποστηρίζει ένα μεγάλο εύρος από γλώσσες προγραμματισμού όπως: Java, Python, R, και Scala. Για την ανάπτυξη των εφαρμογών έχει επιλεχθεί η γλώσσα προγραμματισμού Scala. Παρατίθεται ένα συγκριτικό ανάμεσα στην Scala και την Python εφόσον είναι οι δημοφιλέστερες επιλογές για την ανάπτυξη εφαρμογών με την χρήση του Spark.



Εικόνα 3.1 Ποσοστά χρήσης της εκάστοτε γλώσσας προγραμματισμού με βάση την έρευνα του Databricks το 2015

Scala ή Python για το Apache Spark

Η Scala είναι γρήγορη, όσον αφορά το compile time, και είναι μέτριας δυσκολίας όσον αφορά την εκμάθηση και την χρήση της, ενώ η Python είναι αργή αλλά πάρα πολύ εύκολη στην χρήση. Η πλατφόρμα (framework) του Apache Spark είναι γραμμένη σε Scala. Γνωρίζοντας την προγραμματιστική γλώσσα Scala οι προγραμματιστές δεδομένων έχουν την δυνατότητα να εισέλθουν στον πηγαίο κώδικα του Spark με ευκολία, σε περίπτωση που κάτι δεν λειτουργεί όπως αναμενόταν προκειμένου να το διορθώσουν.

Χρησιμοποιώντας Python αυξάνεται η πιθανότητα να εμφανιστούν περισσότερα προβλήματα και bugs, καθώς η μετάφραση μεταξύ δυο διαφορετικών γλωσσών είναι δύσκολη. Χρησιμοποιώντας λοιπόν την Scala για το Spark παρέχεται πρόσβαση στα πιο πρόσφατα features της πλατφόρμας του Spark, διότι είναι διαθέσιμα πρώτα σε Scala και στην συνέχεια γίνονται διαθέσιμα (ported) στην Python.

Αποφασίζοντας για το ζήτημα Scala ή Python για το Spark, όπως είναι εμφανές τα περισσότερα εξαρτώνται από τα χαρακτηριστικά της κάθε γλώσσας και πως αυτά θα ταιριάξουν καλύτερα στο κάθε project, καθώς η κάθε μια έχει τα δικά της πλεονεκτήματα και μειονεκτήματα. Πριν επιλέξουμε ποιά προγραμματιστική γλώσσα θα χρησιμοποιήσουμε για το Apache Spark είναι απαραίτητο οι προγραμματιστές να μάθουν Scala και Python, ώστε να οικειοποιηθούν με τα χαρακτηριστικά της κάθε μιας. Όταν οι προγραμματιστές λοιπόν, φτάσουν στο σημείο ώστε να γνωρίζουν και Python και Scala, θα είναι πολύ εύκολο να πάρουν απόφαση για το πότε είναι κατάλληλη η επιλογή της κάθε γλώσσας για το Spark. Η επιλογή της γλώσσας προγραμματισμού για το Apache Spark εξαρτάται καθαρά στο τι προβλήματα καλείται ο καθένας να λύσει.

ΥΠΟΚΕΦΑΛΑΙΟ 3.3.1 Ευκολία εκμάθησης

Η προγραμματιστική γλώσσα Scala διαθέτει αρκετές συντακτικές ευκολίες (syntactic sugar) όταν χρησιμοποιείται για την δημιουργία προγραμμάτων για το Apache Spark, γεγονός το οποίο οι αναλυτές big data πρέπει να είναι εξαιρετικά προσεκτικοί. Οι προγραμματιστές πιθανότατα μπορεί να βρουν το συντακτικό της Scala για τον προγραμματισμό εφαρμογών ιδιαίτερα δύσκολο ορισμένες φορές. Λίγες είναι οι βιβλιοθήκες της Scala, οι οποίες δυσκολεύουν στον καθορισμό random symbolic operators και να είναι κατανοητές από προγραμματιστές με ελλιπή εμπειρία. Είναι σημαντικό οι προγραμματιστές να αφιερώνουν χρόνο στην αναγνωσιμότητα του κώδικα. Η Scala, λοιπόν είναι μια εκλεπτυσμένη γλώσσα με ευέλικτης σύνταξη σε σχέση με την Java ή με την Python. Υπάρχει αυξημένη ζήτηση για προγραμματιστές που γνωρίζουν Scala, επειδή οι εταιρίες big data δίνουν προτεραιότητα στους προγραμματιστές που κατέχουν εύκολα (master) μια παραγωγική και εύρωστη γλώσσα για ανάλυση δεδομένων και επεξεργασία στο Apache Spark.

Σε αντίθεση με τα παραπάνω, η Python είναι πιο κατανοητή και εύκολη στην μάθηση, ειδικά για τους προγραμματιστές που γνωρίζουν Java, λόγω σύνταξης και χρήσης πρότυπων βιβλιοθηκών. Παρόλα αυτά η Python δεν αποτελεί ιδανική επιλογή για μεγάλα παράλληλα και κλιμακωτά συστήματα όπως το SoundCloud ή το Twitter. (<https://www.dezyre.com/article/scala-vs-python-for-apache-spark/213>)

ΥΠΟΚΕΦΑΛΑΙΟ 3.3.2 Ευκολία χρήσης

Τόσο η Scala όσο και η Python είναι εξίσου εκφραστικές γλώσσες προγραμματισμού για την δημιουργία προγραμμάτων στο Spark. Για τον λόγο αυτό χρησιμοποιώντας μια από τις δυο αυτές γλώσσες μπορούμε να δημιουργήσουμε Spark content και να καλούμε functions, ώστε να πετύχουμε την θεμιτή λειτουργικότητα του κώδικά μας. Η Python, είναι πιο φιλική στον χρήστη από ότι είναι η Scala, καθώς έχει ένας πιο «λακωνικό τρόπο έκφρασης», καθιστώντας την έτσι πιο εύκολη στην δημιουργία scripts για το Spark. Αξίζει να σημειωθεί, πως η ευκολία χρήσης είναι ένας υποκειμενικός παράγοντας καθώς βασίζεται στην προσωπική προτίμηση του κάθε προγραμματιστή. (<https://www.dezyre.com/article/scala-vs-python-for-apache-spark/213>)

ΥΠΟΚΕΦΑΛΑΙΟ 3.3.3 Προηγμένες δυνατότητες

Η Scala έχει πολλαπλά existential types, macros και implicits. Ο ιδιαίτερος τρόπος σύνταξής της (arcane syntax), την καθιστά δύσκολη και πολλές φορές λιγότερο κατανοητή στους προγραμματιστές που θα προσπαθήσουν να πειραματιστούν, με τις προηγμένες δυνατότητες της (advanced features). Ωστόσο, το πλεονέκτημα της έγκειται στο γεγονός ότι οι δυνατότητες της είναι πολύ σημαντικές σε πολλά frameworks και libraries.

Επεκτείνοντας τα παραπάνω, η Scala δεν παρέχει επαρκή εργαλεία και βιβλιοθήκες για την επιστήμη δεδομένων, όπως είναι η μηχανική μάθηση και η επεξεργασία φυσικής γλώσσας (natural language processing), σε αντίθεση με την Python. Η SparkMLib (η βιβλιοθήκη του Spark για μηχανική μάθηση) παρέχει μόνο ελάχιστους ML algorithms οι οποίοι όμως είναι ιδανικοί για την ανάλυση και την επεξεργασία big data. Τέλος η Scala στερείται καλής οπτικοποίησης και μετασχηματισμού τοπικών δεδομένων. Συνοψίζοντας, η Scala είναι αναμφισβήτητα η καλύτερη επιλογή για το Spark Streaming, επειδή η αντίστοιχη επιλογή σε Python δεν είναι τόσο προηγμένη και ώριμη. (<https://www.dezyre.com/article/scala-vs-python-for-apache-spark/213>)

ΥΠΟΚΕΦΑΛΑΙΟ 3.3.4 Performance

Όσον αφορά την χρήση τους σε ανάλυση και επεξεργασία δεδομένων η Scala είναι 10 φορές ταχύτερη σε σχέση με την Python, λόγω του JVM (Java Virtual Machine), στο οποίο βασίζεται η Scala. Οι επιδόσεις και η ταχύτητα της Python είναι λιγότερο ικανοποιητικές, στην περίπτωση που στον κώδικα κάνουμε κλήσεις προς τις βιβλιοθήκες του Spark. Η κατάσταση δυσχεραίνει όσον αφορά στην Python, στην περίπτωση που υπάρχει η ανάγκη για επεξεργασία πολλών γραμμών κώδικα και δεδομένων, καθιστώντας την έτσι πολύ πιο αργή σε σχέση με τον αντίστοιχο/ισοδύναμο κώδικα σε Scala. Ο διερμηνέας (interpreter) PyPy της Python έχει in-built JIT (Just-In-Time) compiler, ο οποίος είναι πολύ γρήγορος μεν αλλά δεν παρέχει υποστήριξη για ποικίλες επεκτάσεις της Python C. Σε τέτοιες περιπτώσεις, ο CPython interpreter που παρέχει υποστήριξη C extensions για τις βιβλιοθήκες αποδίδει καλύτερα σε σχέση με τον PyPy interpreter.

Χρησιμοποιώντας την Python για το Apache Spark, ο προγραμματιστής πρέπει να αποδεχθεί το κόστος του overhead σε σχέση με την Scala αλλά το γεγονός αυτό το οποίο έχει τελικά σημασία είναι το τι ακριβώς θέλει να υλοποιήσει ο προγραμματιστής. Η Scala λοιπόν, είναι ταχύτερη από την Python στην περίπτωση που υπάρχουν λίγοι πυρήνες (cores). Όσο λοιπόν, ο αριθμός τους αυξάνεται το πλεονέκτημα απόδοσης της Scala, αρχίζει να φθίνει. Στην περίπτωση που υπάρχει η δυνατότητα για τους προγραμματιστές να δουλέψουν με περισσότερους πυρήνες, η «επίδοση» δεν αποτελεί σημαντικό παράγοντα για την επιλογή της γλώσσας προγραμματισμού για το Apache Spark. Η μόνη περίπτωση όπου η «επίδοση» είναι σημαντικός παράγοντας στο παραπάνω σενάριο, είναι όταν χρειαζόμαστε σπουδαία επεξεργαστική λογική, ευνοώντας έτσι την επιλογή της Scala. (<https://www.dezyre.com/article/scala-vs-python-for-apache-spark/213>)

ΥΠΟΚΕΦΑΛΑΙΟ 3.3.5 TypeSafety

Στον τομέα της πληροφορικής, με τον όρο TypeSafety εννοούμε, κατά το πόσο μια γλώσσα προγραμματισμού αποτρέπει σφάλματα τύπων. Ένα σφάλμα τύπων είναι μία λανθασμένη ή μη επιθυμητή συμπεριφορά προγράμματος που προκαλείται από την ασυμφωνία μεταξύ διαφορετικών τύπων δεδομένων.

Οι προγραμματιστές του Apache Spark, πρέπει να κάνουν συνεχώς re-factor τον κώδικα, με βάση τις νέες απαιτήσεις. Η Scala είναι μια στατικού τύπου γλώσσα (statically typed language), αν και φαινομενικά μοιάζει με δυναμικού τύπου γλώσσα (dynamically typed language) λόγω του «classy type inference mechanism», καθώς παρέχει στον compiler την δυνατότητα να βρει compile time errors. Κάνοντας refactoring τον κώδικα του προγράμματος μιας statically typed γλώσσας όπως η Scala, είναι τυπικά πιο εύκολο και δεν απαιτεί κόπο, σε αντίθεση με το να κάνεις refactoring των

κώδικα μιας dynamic language όπως η Python. Οι προγραμματιστές αντιμετωπίζουν συχνά δυσκολίες όταν τροποποιούν κώδικα γραμμένο σε Python, καθώς δημιουργούνται περισσότερα bugs κατά την επίλυση των ήδη υπαρχόντων. Γενικότερα, είναι προτιμότερο για κάποιον να γράφει με πιο αργό ρυθμό αλλά να είναι ασφαλής χρησιμοποιώντας Scala για το Spark, παρά να προγραμματίζει πιο γρήγορα αλλά να είναι πιο επιρρεπής στα λάθη χρησιμοποιώντας Python για το Spark.

Η Python είναι μια αποτελεσματική λύση για το Spark για μικρότερης κλίμακας προβλήματα, και η απόδοση της δυσχεραίνει όσο μεγαλώνει το πρόβλημα που καλείται να επιλύσει. Στον αντίποδα, η στατικού τύπου γλώσσα Scala, είναι κατάλληλη για την παραγωγή μεγάλων λογισμικών συστημάτων. (<https://www.dezyre.com/article/scala-vs-python-for-apache-spark/213>)

ΥΠΟΚΕΦΑΛΑΙΟ 3.4 The Spark Programming model

Το πρώτο βήμα για τον προγραμματισμό στο Spark ξεκινά με ένα σύνολο δεδομένων , ή μερικών δεδομένων, συνήθως αποθηκευμένων σε κάποια μορφή κατανεμημένης και μόνιμου τύπου χώρου αποθήκευσης (persistent storage, όπως για παράδειγμα Hard disk drives και solid-state drives) όπως το Hadoop Distributed File System (HDFS)

Η δημιουργία ενός προγράμματος για το Spark, συνοψίζεται στα παρακάτω βήματα:

- Καθορισμός ενός συνόλου μετασχηματισμών, που αφορά στα εισαγόμενα datasets.
- Την επίκληση ενεργειών , οι οποίες θα παράγουν τα μετασχηματισμένα δεδομένα και θα τα αποθηκεύσουν είτε σε persistent storage είτε στην τοπική μνήμη του driver προγράμματος.
- Εκτέλεση «τοπικών υπολογισμών», οι οποίοι βασίζονται σε αποτελέσματα που έχουν υπολογιστεί με «διαμοιρασμένο τρόπο». Με αυτόν τον τρόπο οι προγραμματιστές αποφασίζουν τι μετασχηματισμούς και ενέργειες θα αναθέσουν για τα επόμενα βήματα.

Κατανοώντας το Spark εννοούμε , ότι ο προγραμματιστής θα πρέπει να αντιλαμβάνεται το σημείο τομής μεταξύ των δυο αφαιρέσεων που προσφέρει το framework: αποθήκευση και εκτέλεση. Το Spark «ταιριάζει» με έναν κομψό τρόπο τις δυο αυτές αφαιρέσεις , επιτρέποντας ουσιαστικά την αποθήκευση στην μνήμη κάθε ενδιάμεσου τμήματος ενός pipeline, για μελλοντική χρήση.(Ryza, Laserson, Owen & Wills,2015:11)

ΥΠΟΚΕΦΑΛΑΙΟ 3.4.1 Resilient Distributed Datasets (RDDs) και DAG

RDD

Το Spark εμπεριέχει την ιδέα του *resilient distributed dataset* (RDD), το οποίο είναι μια συλλογή από στοιχεία ανθεκτικά στα σφάλματα, με την δυνατότητα η συλλογή να λειτουργήσει σε παραλληλία. Υπάρχουν λοιπόν δυο τρόποι για την δημιουργία RDDs: παραλληλίζοντας μια ήδη

υπάρχουσα συλλογή που βρίσκεται στο κυρίως πρόγραμμα του χρήστη (driver program) ή να γίνει αναφορά σε κάποιο dataset που βρίσκεται σε κάποιο εξωτερικό σύστημα αποθήκευσης, όπως τα HDFS, HBase ή οποιαδήποτε λύση που υποστηρίζει Hadoop InputFormat.

Το Resilient Distributed Dataset είναι μια έννοια βαθιά ριζωμένη στην καρδιά του Spark. Είναι σχεδιασμένο να υποστηρίζει αποθήκευση δεδομένων στην κύρια μνήμη, η οποία κατανέμεται σε όλη την συστάδα (cluster), με τέτοιο τρόπο ώστε να είναι αποδεδειγμένα ανεκτικός σε σφάλματα και ταυτόχρονα πιο αποδοτικός. Η ανεκτικότητα στα σφάλματα επιτυγχάνεται εν μέρη στην παρακολούθηση της πηγής των μετασχηματισμών που εφαρμόζονται στα δεδομένα. Τα δεδομένα έχουν την μορφή coarse-grained . Με τον όρο coarse-grained , περιγράφουμε ένα σύστημα το οποίο αποτελείται από μεγάλα υπο-συστήματα.

Η αποδοτικότητα επιτυγχάνεται μέσω της παραλληλισμού της επεξεργασίας σε πολλαπλούς κόμβους μιας συστάδας και της ελαχιστοποίησης της αντιγραφής των δεδομένων ανάμεσα σε αυτούς τους κόμβους.

Την στιγμή που τα δεδομένα φορτωθούν σε ένα RDD, είναι δυνατό να διεξαχθούν δυο μορφές εργασίας:

- Μετασχηματισμοί (Transformations), όπου δημιουργείται ένα νέο RDD μετασχηματίζοντας το πρωτότυπο, μέσα από διεργασίες και μεθόδους όπως mapping, filtering, και άλλα.
- Ενέργειες (Actions), στις οποίες επιστρέφεται μια τιμή στο driver πρόγραμμα μετά από κάποια πράξη ή υπολογισμό στο dataset χωρίς να γίνει αλλαγή σε αυτό. Ένα τέτοιο παράδειγμα είναι και οι μετρήσεις.

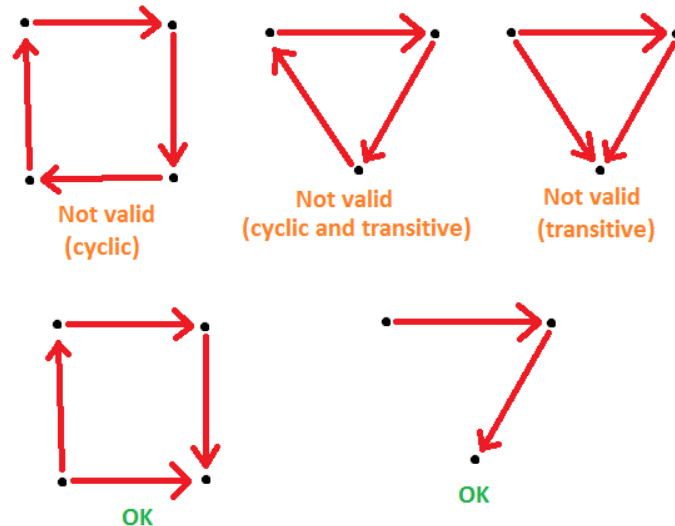
Το πρωτότυπο RDD παραμένει καθ' όλην την διάρκεια της διαδικασίας αμετάβλητο. Οι μετασχηματισμοί από το RDD1 μέχρι και το RDDn είναι καταγεγραμμένοι, και μπορούν να επαναληφθούν σε περίπτωση που υπάρξει απώλεια δεδομένων ή βλάβη σε κάποιο κόμβο της συστάδας.

Οι μετασχηματισμοί αυτοί είναι ως γνωστόν «lazily evaluated». Με τον όρο αυτό εννοείται ότι οι μετασχηματισμοί δεν θα εκτελεστούν μέχρι μια μεταγενέστερη ενέργεια, να έχει ανάγκη το αποτέλεσμα τους. Ο συγκεκριμένος τρόπος λειτουργίας έχει ως αποτέλεσμα την αύξηση των επιδόσεων, καθώς αποτρέπεται η ανάγκη της επεξεργασίας δεδομένων που δεν χρειάζονται την συγκεκριμένη χρονική στιγμή. Σε μερικές περιπτώσεις βέβαια, υπάρχει πιθανότητα να παρουσιαστεί bottleneck, αναγκάζοντας με αυτόν τον τρόπο τις εφαρμογές να αναβάλλονται, καθώς αναμένουν την επεξεργασία και το αποτέλεσμα μια πράξης να ολοκληρωθεί. Τέλος, όπου είναι δυνατόν τα RDDs παραμένουν στην μνήμη, αυξάνοντας σε μεγάλο βαθμό την συνολική επίδοση του cluster, ιδιαιτέρως σε περιπτώσεις χρήσης όπου υπάρχει η ανάγκη για επαναληπτικά ερωτήματα ή διεργασίες.(Scott,2015:18-19)

DAG

Αναφερόμενοι στο DAG (directed acyclic graph) εννοούμε ένα «κατευθυνόμενο άκυκλο γράφημα», το οποίο αποτελείται από ένα πεπερασμένο πλήθος κόμβων και ακμών, με την κάθε ακμή του γραφήματος να συνδέεται με τους υπόλοιπους κόμβους, με τέτοιο τρόπο ώστε να μην υπάρχει μονοπάτι, το οποίο να ξεκινά από τον κόμβο 'ν' , να διαπερνά μια αλληλουχία άλλων

κόμβων και να καταλήγει σε ξανά πίσω στον κόμβο 'ν'. Το DAG θεωρείται κατευθυνόμενο επειδή τηρεί τοπολογική σειρά, δηλαδή η ακολουθία των κόμβων έχει κατεύθυνση από τους νεώτερους προς τους παλαιότερους. (<http://mathworld.wolfram.com/AcyclicDigraph.html>)



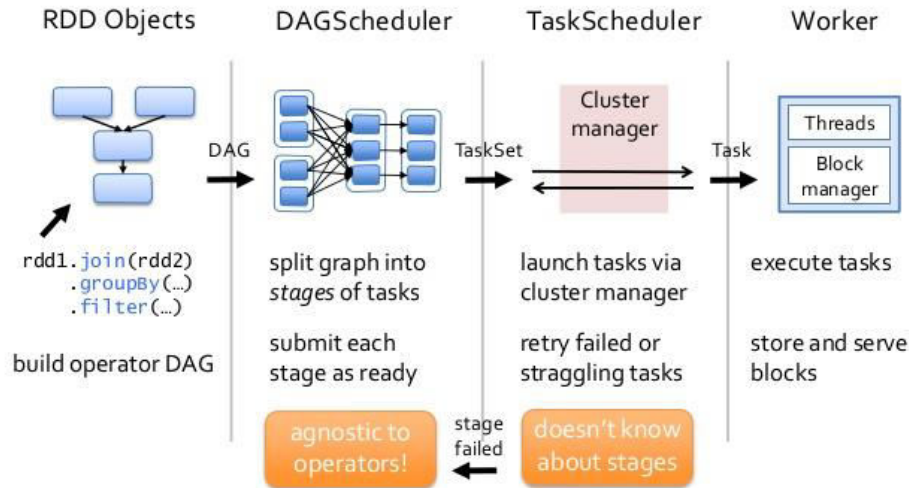
Εικόνα 3.2 Παραδείγματα για τι θεωρείται DAG

Το Spark έχει την δυνατότητα να δημιουργήσει ένα DAG με βάση δυο διαθέσιμους μετασχηματισμούς, τους narrow transformation and wide transformation, οι οποίοι εφαρμόζονται στα RDDs. Με τον narrow transformation, τα δεδομένα δεν χρειάζονται να μοιραστούν σε όλα τα partitions, όπως για παράδειγμα οι λειτουργίες Map, filter κτλ. . Αντίθετα, στον μετασχηματισμό wide transformation τα δεδομένα χρειάζεται να μοιραστούν, όπως για παράδειγμα στην λειτουργία reduceByKey.

Στο Spark, όταν στο υψηλότερο επίπεδο του μια ενέργεια καλείται για ένα RDD, τότε δημιουργείται ένα DAG το οποίο υποβάλλεται στον DAG scheduler. Ο DAG scheduler με την σειρά του διαχωρίζει το κατοχυρωμένο DAG σε στάδια αποτελούμενα από διεργασίες (operators), οι οποίες βασίζονται στα εισαγόμενα δεδομένα. Παραδείγματος χάριν, πολλές διεργασίες τύπου Map μπορούν να προγραμματιστούν σε ένα μόνο βήμα (stage). Το τελικό αποτέλεσμα του DAG scheduler είναι ένα σύνολο από βήματα. Στην συνέχεια αυτά τα βήματα, αποστέλλονται στον Task Scheduler, ο οποίος τα επεξεργάζεται μέσω ενός cluster manager (Spark Standalone, Yarn, Mesos). Ο Task Scheduler δεν έχει γνώση αν τα βήματα αυτά έχουν κάποια εξάρτηση (dependencies) μεταξύ τους. Τέλος ο Worker εκτελεί τα παραπάνω βήματα στο slave node. (<http://stackoverflow.com/questions/25836316/how-dag-works-under-the-covers-in-rdd>)

Spark Internals
<https://wiki.apache.org/confluence/display/SPARK/Spark+Internals>

Scheduling Process



Εικόνα 3.3 To Scheduling Process του Spark

ΥΠΟΚΕΦΑΛΑΙΟ 3.4.2 Parallel Operations

Οι παράλληλες εργασίες που μπορούν να εκτελεστούν πάνω στα RDDs είναι οι εξής:

- **Reduce:** Συνδυάζει τα στοιχεία ενός dataset, χρησιμοποιώντας μια associative function με σκοπό να παράγει ένα αποτέλεσμα στο driver program.
- **Collect:** Συλλέγει και στέλνει όλα τα στοιχεία ενός dataset στο driver program.
- **Foreach:** Τροποποιεί κάθε στοιχείο ενός dataset, με βάση μια συνάρτηση (function) δημιουργημένη από τον χρήστη.

Αξίζει να σημειωθεί ότι το Spark επί του παρόντος δεν υποστηρίζει μια ομαδοποιημένη εργασία τύπου reduce. Τα αποτελέσματα της reduce στο MapReduce συλλέγονται μόνο από το driver πρόγραμμα. Υπάρχουν σχέδια για το μέλλον ώστε να υπάρξει υποστήριξη για «grouped reductions», χρησιμοποιώντας μια “shuffle transformation” επί των διανεμόμενων datasets. Ωστόσο, η χρήση ενός και μόνο reducer είναι αρκετή για να υλοποιηθούν ποικίλοι αλγόριθμοι. (Zaharia, Chowdhury, Franklin, Shenker & Stoica, 2010:2)

ΥΠΟΚΕΦΑΛΑΙΟ 3.4.3 Shared Variables

Οι προγραμματιστές επικαλούνται λειτουργίες όπως το map, το filter και το reduce με το πέρασμα τους σε closures (functions) στο Spark. Γενικά είναι σύνηθες στον functional προγραμματισμό, αυτά τα closures να αναφέρονται σε μεταβλητές στο πεδίο όπου δημιουργούνται. Τυπικά, όταν το Spark τρέχει ένα closure σε ένα worker node, οι μεταβλητές αυτές αντιγράφονται σε αυτό. Ωστόσο, το Spark δίνει την δυνατότητα στους προγραμματιστές να δημιουργήσουν δυο restricted types των shared variables, για την υποστήριξη δυο απλών αλλά κοινών προτύπων χρήσης (usage patterns):

- **Broadcast variables:** Σε περιπτώσεις που, ένα μεγάλο τμήμα δεδομένων μόνο για ανάγνωση (read-only) , όπως για παράδειγμα το lookup table, χρησιμοποιείται σε πολλαπλές παράλληλες εργασίες, είναι προτιμότερο να διαμοιραστεί μια φορά στους workers σε σχέση με το να διανέμεται με κάθε διεργασία (task). Το Spark επιτρέπει στους προγραμματιστές να δημιουργήσουν μια «broadcast variable» ,η όποια αναλαμβάνει να μεταφέρει και να αντιγράφει τα δεδομένα στους workers με ένα πέρασμα. Τέλος, το Spark διαμοιράζει αυτόματα τα κοινά στοιχεία που είναι αναγκαία για τις διεργασίες, σε κάθε στάδιο.
- **Accumulators:** Οι μεταβλητές αυτές μπορούν να προστεθούν μόνο σε προσεταιριστικού τύπου ενέργεια (associative operation), επιτυγχάνοντας αποτελεσματική υποστήριξη παραλληλίας. Την δυνατότητα της προσθήκης έχουν μόνο οι workers, ενώ η ανάγνωση πραγματοποιείται από το driver πρόγραμμα. Συνήθως, χρησιμοποιούνται για την υλοποίηση «μετρητών», και στην παροχή ενός πιο αυστηρού συντακτικού για παράλληλα αθροίσματα. Τέλος ,το Spark υποστηρίζει εγγενώς accumulators αριθμητικού τύπου και οι προγραμματιστές έχουν την δυνατότητα να προσθέσουν και άλλους τύπους δεδομένων. Οι τύποι δεδομένων που δημιουργήθηκαν από τους προγραμματιστές εμφανίζονται στο UI του Spark, καθιστώντας το χρήσιμη λειτουργία για την κατανόηση της προόδου των ενεργών σταδίων. (Zaharia, Chowdhury, Franklin, Shenker & Stoica,2010:2)

ΚΕΦΑΛΑΙΟ 4 : Αλγόριθμοι μηχανικής μάθησης βασιζόμενοι στις βιβλιοθήκες ML και MLLIB

ΕΙΣΑΓΩΓΗ

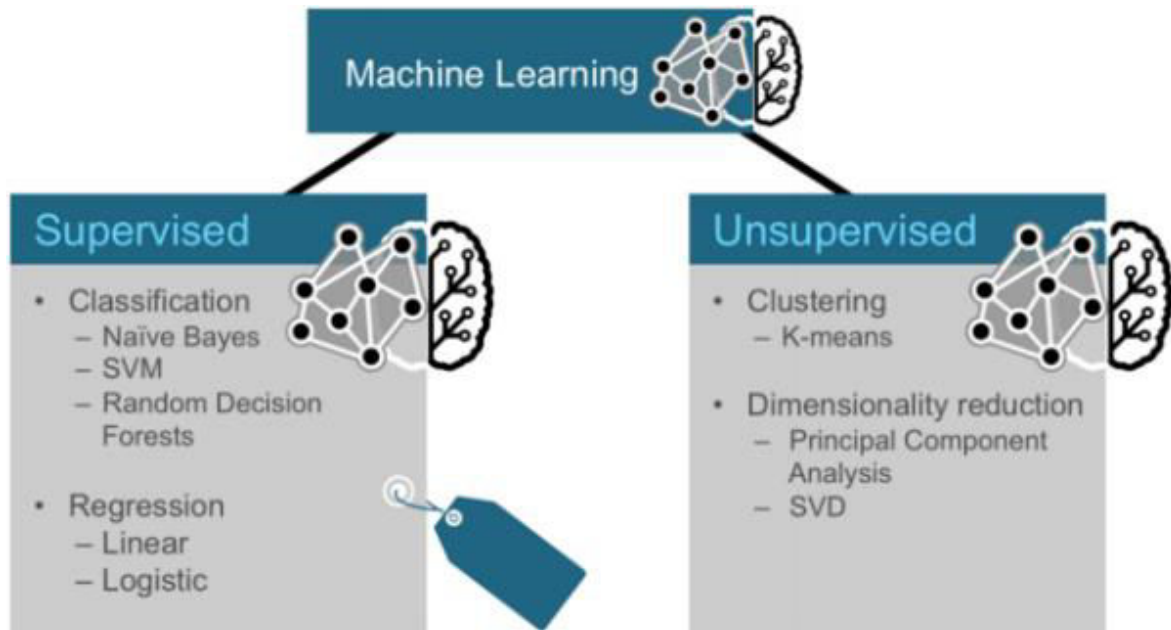
Το κεφάλαιο αυτό είναι αφιερωμένο στην παρουσίαση και περιγραφή των δυνατοτήτων της μηχανικής μάθησης, καθώς και ανάλυση των αλγορίθμων που χρησιμοποιήθηκαν για την δημιουργία των εφαρμογών.

ΥΠΟΚΕΦΑΛΑΙΟ 4.1 Μηχανική Μάθηση και ML lib

Η βιβλιοθήκη του Spark που αφορά την κλιμακωτή μηχανική μάθηση (scalable machine learning) ονομάζεται MLlib, και υλοποιεί μια σειρά από συχνά χρησιμοποιούμενους αλγορίθμους μηχανικής μάθησης καθώς και στατικούς αλγορίθμους. Τα παραπάνω, συμπεριλαμβάνουν correlations and hypothesis testing, classification and regression, clustering, and principal component analysis. Γενικά η μηχανική μάθηση μπορεί να διαχωριστεί σε δυο τύπους αλγορίθμων: με επίβλεψη και χωρίς επίβλεψη. Οι αλγόριθμοι με επίβλεψη χρησιμοποιούν τα πρότυπα εισόδου μαζί με την κλάση στην οποία ανήκουν, σε αντίθεση με τους αλγορίθμους χωρίς επίβλεψη όπου χρησιμοποιούν μόνο τα πρότυπα εισόδου χωρίς να γνωρίζουν σε ποια κλάση ανήκει το καθένα. Οι πιο κοινές κατηγορίες τεχνικών μηχανικής μάθησης είναι τρεις : Classification, Clustering και Regression και παρουσιάζονται αναλυτικά μαζί με τους αλγορίθμους όπου χρησιμοποιήθηκαν για την υλοποίηση των εφαρμογών στα παρακάτω κεφάλαια.

Η βιβλιοθήκη MLlib προσφέρει λειτουργικότητα και υποστήριξη στα παρακάτω:

- Ταξινόμηση σε κλάσεις (Classification)
- Αναδρομή (Regression)
- Collaborative Filtering (Τεχνική των συστημάτων που προτείνουν προϊόντα)
- Συστάδες υπολογιστών (Clustering)
- Dimensionality Reduction (Διαδικασία η οποία μειώνει τις τυχαίες μεταβλητές από ένα σύνολο μεταβλητών που λαμβάνουμε υπόψη)
- Εξαγωγή χαρακτηριστικών (Feature Extraction) (πχ από ένα πρόσωπο ή ένα αντικείμενο)
- Εξόρυξη προτύπων (Frequent Pattern Mining) (παιδί της εξόρυξης δεδομένων)
- Βελτιστοποίηση (Optimization) (Frampton,2015:3)



Εικόνα 4.1 Αλγόριθμοι της μηχανικής μάθησης

Διάφορες μεταξύ ML και MLlib

Αρκετοί νέοι προγραμματιστές έχουν την απορία, γιατί υπάρχουν δυο υλοποιήσεις μηχανικής μάθησης για το Spark (ML και MLlib) και ποιές είναι οι κύριες διαφορές τους. Με βάση το documentation του Spark η υποστήριξη για Machine Learning χωρίζεται σε δυο πακέτα:

- spark.mllib, το οποίο περιέχει το πρωτότυπο API “χτισμένο” με βάση τα RDDs.
- spark.ml, το οποίο παρέχει ένα υψηλού επιπέδου API “χτισμένο” με βάση τα DataFrames για την κατασκευή ML pipelines.

Γενικά είναι προτιμότερο να χρησιμοποιείται το spark.ml, γιατί με τα DataFrames το API είναι περισσότερο versatile και flexible. Βέβαια, θα συνεχίζει να υπάρχει υποστήριξη για το spark.mllib για το άμεσο μέλλον, παράλληλα με την ανάπτυξη του spark.ml, και οι χρήστες θα νιώθουν ασφάλεια με την χρήση του καθώς θα γίνονται διαθέσιμα νέες λειτουργίες. Γενικά το Spark ML παρέχει στους προγραμματιστές ένα σύνολο από εργαλεία για την δημιουργία pipelines διάφορων μετασχηματισμών μηχανικής μάθησης που θα πραγματοποιηθούν πάνω στα δεδομένα. Για παράδειγμα, καθιστά εύκολο το να ενωθούν η εξαγωγή χαρακτηριστικών (feature extraction), η μείωση του αριθμού των τυχαίων μεταβλητών που έχουμε υπόψη (dimensionality reduction) και η εκπαίδευση του ταξινομητή (classifier) σε ένα και μόνο μοντέλο, με την δυνατότητα να χρησιμοποιηθεί στην συνέχεια για classification. Παρόλα αυτά το MLlib είναι παλαιότερο και είναι υπό ανάπτυξη για περισσότερο διάστημα με επακόλουθο να έχει περισσότερα features.

Εν συντομία:

ML

- Νεότερο
- Pipelines
- Dataframes
- Ευκολότερο στην κατασκευή ενός πρακτικού pipeline για την χρήση στην μηχανική μάθηση.

MLlib

- Παλαιότερο
- RDD's
- Περισσότερες λειτουργίες και δυνατότητες

Στην πράξη, είναι χρησιμότερο για την ώρα να χρησιμοποιούνται και τα δυο πακέτα. Με το πέρασμα του χρόνου, οι επιπλέον λειτουργίες του MLlib θα μεταφερθούν στο ML, με αποτέλεσμα το MLlib να καταργηθεί.

ΥΠΟΚΕΦΑΛΑΙΟ 4.2 Kmeans (Clustering)

Πρόκειται για μία μέθοδο ομαδοποίησης μη εποπτευόμενης μάθησης. Η μέθοδος βρίσκει κέντρα για ομάδες προτύπων (cluster centers ή αλλιώς centroids). Κάθε πρότυπο ανήκει στην ομάδα (cluster) με το κοντινότερο κέντρο και κάθε κέντρο είναι η μέση τιμή των προτύπων που ανήκουν στην αντίστοιχη ομάδα.

Ο αλγόριθμος δέχεται ως εισόδους P πρότυπα $x^{(1)}, x^{(2)}, \dots, x^{(P)}$ και το πλήθος K των κλάσεων που ανήκουν αυτά τα πρότυπα, το οποίο ισούται με τον αριθμό των κέντρων. Στην ουσία, αναζητάει ένα διάνυσμα αντιπρόσωπο για κάθε κλάση. Έτσι ξεκινάει με τυχαία αρχικά διανύσματα κέντρων και αυτά διαμορφώνονται βάσει δύο κανόνων:

1. Για κάθε $i = 1, \dots, K$, η κλάση χ_i αποτελείται από τα πρότυπα $x^{(p)}$ τα οποία βρίσκονται πιο κοντά στο διάνυσμα c_i σε σχέση με όλα τα άλλα διανύσματα $c_j, i \neq j$, δηλαδή

$$\|x^{(p)} - c_i\| = \min \|x^{(p)} - c_j\| \quad (4.1)$$

2. Για κάθε $i = 1, \dots, K$, το διάνυσμα c_i είναι ο μέσος όρος των προτύπων που ανήκουν στην κλάση χ_i , δηλαδή

$$c_i = \frac{1}{|\chi_i|} \sum_{p \in \chi_i} x^{(p)} \quad (4.2)$$

Όπου $|\chi_i|$ είναι το πλήθος των προτύπων της κλάσης. Το διάνυσμα c_i , λέγεται κέντρο της κλάσης.

Ο αλγόριθμος αυτός είναι επαναληπτικός και σταματάει όταν δεν μεταβληθεί το κέντρο έστω και μίας οποιασδήποτε κλάσης. Ο αλγόριθμος δουλεύει με τον ίδιο τρόπο και στο Spark και στο Scikit.(Παπαδοπούλου,2016:32-33)

ΥΠΟΚΕΦΑΛΑΙΟ 4.3 SVM (Classification)

SPARK

Η πιο κοινή μέθοδος για την επίλυση μεγάλης κλίμακας προβλημάτων ταξινόμησης (classification) είναι η linear SVM. Είναι μια γραμμική μέθοδος και περιγράφεται από την εξίσωση:

$$f(\mathbf{w}) := \lambda R(\mathbf{w}) + \frac{1}{n} \sum_{i=1}^n L(\mathbf{w}; \mathbf{x}_i, y_i) . \quad (4.3)$$

Ο υπολογισμός της συνάρτησης «απωλειών», δίνεται με βάση το «hinge loss»:

$$L(\mathbf{w}; \mathbf{x}, y) := \max\{0, 1 - y\mathbf{w}^T \mathbf{x}\} . L(\mathbf{w}; \mathbf{x}, y) := \max\{0, 1 - y\mathbf{w}^T \mathbf{x}\} .$$

Τυπικά, η μέθοδος linear SVM εκπαιδεύεται με «L2 regularization» αλλά η υπάρχει η υποστήριξη για « L1 regularization.». Το αποτέλεσμα της μεθόδου linear SVM είναι ένα SVM μοντέλο.

SCIKIT

Οι μηχανές διανυσμάτων υποστήριξης της ταξινόμησης, SVC, NuSVC και LinearSVC λαμβάνουν ως είσοδο δύο σειρές: μια σειρά X μεγέθους [n_samples, n_features], που κατέχονται δείγματα εκπαίδευσης, καθώς και μια σειρά γετικετών κατηγορίας(χορδές ή ακέραιοι), μεγέθους[n_samples]. Τα δεδομένα αυτά μπορούν να είναι τόσο πυκνά όσο και αραιά.

Σε σύγκριση αυτών των μοντέλων, τα γραμμικά μοντέλα LinearSVC () και SVC (kernel = «γραμμική») αποδίδουν ελαφρώς διαφορετικά όρια απόφασης. Το LinearSVC ελαχιστοποιεί το τετραγωνικό σφάλμα, ενώ SVC ελαχιστοποιεί το τακτικό σφάλμα. Και τα δύο αυτά μοντέλα, όμως, έχουν γραμμικά όρια απόφασης σε αντίθεση με μη-γραμμικά μοντέλα πυρήνα (πολυωνυμική ή Gaussian RBF) τα οποία έχουν πιο ευέλικτα μη γραμμικά όρια απόφασης με σχήματα που εξαρτώνται από το είδος του πυρήνα και τις παραμέτρους του.

Από την άλλη το μοντέλο NuSVC είναι παρόμοιο με το μοντέλο SVC όμως διαφέρουν στο ότι δέχονται ελαφρώς διαφορετικά σύνολα παραμέτρων και έχουν διαφορετικά μαθηματικά σκευάσματα. Δεδομένου το ότι το LinearSVC δεν δέχεται πυρήνα λέξη-κλειδί, το NuSVC είναι μια άλλη εφαρμογή SVM της Linear SVC για την περίπτωση ενός γραμμικού πυρήνα.(Παπαδοπούλου,2016:41-42)

ΥΠΟΚΕΦΑΛΑΙΟ 4.4 Regression

SPARK

Ο συχνότερος τρόπος επίλυσης προβλημάτων παλινδρόμησης (regression) είναι με την χρήση της «linear least squares» φόρμουλας. Είναι μια γραμμική μέθοδος και περιγράφεται από την εξίσωση:

$$f(\mathbf{w}) := \lambda R(\mathbf{w}) + \frac{1}{n} \sum_{i=1}^n L(\mathbf{w}; \mathbf{x}_i, y_i) . \quad (4.4)$$

Υπάρχουν ποικίλες μέθοδοι παλινδρόμησης που προέρχονται με την χρήση διαφορετικών τύπων «regularization»:

- No Regularization, που χρησιμοποιεί η « linear least squares»
- L2 Regularization, που χρησιμοποιεί η «ridge regression» και σε αυτήν έχει βασιστεί η υλοποίησης της εφαρμογής με βάση την παλινδρόμηση (υποκεφάλαιο 5.5).
- L1 Regularization, που χρησιμοποιεί η «lasso»

Για όλα τα παραπάνω μοντέλα, το μέσο σφάλμα $\frac{1}{n} \sum_{i=1}^n (\mathbf{w}^T \mathbf{x}_i - y_i)^2$ (4.5) είναι γνωστό και ως «mean squared error»

SCIKIT

Στο Scikit χρησιμοποιήθηκε το μοντέλο «ridge regression», που αναφέραμε παραπάνω με την διάφορα, ότι δεν γίνεται η χρήση του «Stochastic gradient descent» όπως συμβαίνει στο Spark.

ΚΕΦΑΛΑΙΟ 5 : Δημιουργία Εφαρμογών

ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο γίνεται περιγραφή του τρόπου σκέψης των επιστημόνων της ανάλυσης δεδομένων, παράλληλα με την παρουσίαση των προκλήσεων που θα κλιθούν να λύσουν. Τέλος, υπάρχει αναλυτική παρουσίαση των εφαρμογών που δημιουργήθηκαν για τα πλαίσια της πτυχιακής εργασίας.

ΥΠΟΚΕΦΑΛΑΙΟ 5.1 Analyzing Big Data

Πριν από 5 ή 10 χρόνια ήταν αδύνατο να πραγματοποιηθούν εργασίες ανάλυσης big data όπως:

- Δημιουργία ενός μοντέλου, όπου θα ανακαλύπτει απάτες με πιστωτικές κάρτες, χρησιμοποιώντας χιλιάδες χαρακτηριστικά και δισεκατομμύρια συναλλαγών.
- Με έξυπνο τρόπο να προτείνονται, εκατομμύρια προϊόντα σε εκατομμύρια χρήστες .
- Εκτίμηση του χρηματοοικονομικού κινδύνου μέσω προσομοιώσεων.
- Να πραγματοποιείται με ευκολία η ανίχνευση ασθενειών μέσω της ανάλυσης χιλιάδων ανθρωπίνων γονιδιωμάτων.

Με αυτόν τον τρόπο, όταν οι άνθρωποι αναφέρονται στο ότι ζούμε στην εποχή των «big data», εννοούν ότι πλέον έχουμε τα εργαλεία για την συλλογή, αποθήκευση και επεξεργασία της πληροφορίας σε μεγέθη που προηγουμένως ήταν ανήκουστα. Κάπου εδώ εμφανίστηκαν εφαρμογές ανοικτού κώδικα, κατέχοντας όλες τις παραπάνω ικανότητες και λειτουργίες, όπως το Hadoop και ο απόγονος του το Spark, και έχουν δεχθεί ευρεία υποστήριξη από οργανισμούς σχεδόν σε κάθε τομέα. Υπάρχει βέβαια, ένα κενό όπου η Επιστήμη δεδομένων (data science) καλείται να το καλύψει. Το κενό συναντάται ανάμεσα στον χειρισμό όλων αυτών των εργαλείων και δεδομένων, και στην αξιοποίηση τους για την παραγωγή κάτι χρήσιμου. Η επιστήμη δεδομένων λοιπόν, είναι η πρακτική της μετατροπής των εργαλείων και των raw data σε πληροφορία, η οποία είναι αναγνώσιμη και σημαντική για τους μη-επιστήμονες. (Ryza, Laserson, Owen & Wills, 2015:1-2)

Οι προκλήσεις που αντιμετωπίζει η Επιστήμη Δεδομένων

Αρχικά, η συντριπτική πλειοψηφία των επιτυχημένων αναλύσεων έγκειται στην προ-επεξεργασία των δεδομένων, καθώς συνήθως αυτά παρουσιάζονται σε «ακατάστατη» μορφή. Ειδικότερα, μεγάλα data sets δεν επιδέχονται άμεση εξέταση από ανθρώπους, καθώς μπορεί να απαιτούνται υπολογιστές μέθοδοι που να μην έχουν ανακαλυφθεί ακόμα, για την διαπίστωση των βημάτων της προ-επεξεργασίας που απαιτείται. Επίσης, όταν έρθει η στιγμή της βελτιστοποίησης των επιδόσεων ενός μοντέλου, μια τυπική data pipeline έχει περισσότερο ανάγκη στην αφιέρωση χρόνου για «feature engineering και selection» σε σχέση με την επιλογή ή της δημιουργίας αλγορίθμων. Για παράδειγμα, όταν οι επιστήμονες θέλουν να δημιουργήσουν ένα μοντέλο το οποίο θα ανιχνεύει δόλιες αγορές σε μια ιστοσελίδα, θα πρέπει να επιλέξουν από μια μεγάλη ποικιλία πιθανών χαρακτηριστικών, όπως «IP location info», «login times» και «click logs».

Καθένα από αυτά έρχεται με τις δικές του προκλήσεις για την μετατροπή του σε διανύσματα (vectors) για να επεξεργαστεί από αλγορίθμους μηχανική μάθησης.

Δεύτερον, η επανάληψη (iteration) είναι ένα θεμελιώδες μέρος της επιστήμης δεδομένων, καθώς η μοντελοποίηση και η ανάλυση απαιτούν πολλά περάσματα πάω από τα ίδια δεδομένα. Μια πτυχή αυτού, βρίσκεται μέσα στους αλγορίθμους μηχανικής μάθησης και στις στατιστικές διαδικασίες. Δημοφιλής διαδικασίες βελτιστοποίησης όπως η «stochastic gradient descent» και η «expectation maximization» περιλαμβάνουν επαναλαμβανόμενες σαρώσεις στα εισαγόμενα δεδομένα για την επίτευξη σύγκλισης. Επίσης, η επανάληψη έχει σημασία για την ροή εργασίας του εκάστοτε επιστήμονα δεδομένων. Όταν οι επιστήμονες δεδομένων, αρχικά διερευνούν και προσπαθούν να πάρουν μια ιδέα για ένα σύνολο δεδομένων, συνήθως τα αποτελέσματα ενός ερωτήματος δίνουν αρκετές πληροφορίες για το πιο θα πρέπει να είναι το επόμενο ερώτημα. Κατά την κατασκευή μοντέλων, οι επιστήμονες δεδομένων δεν προσπαθούν να το ολοκληρώσουν σε μια μόνο προσπάθεια ή δοκιμή. Επιλέγοντας τα σωστά χαρακτηριστικά, τους σωστούς αλγορίθμους και μεταβλητές, και εκτελώντας σημαντικά τεστ απαιτείται πειραματισμός. Αν ένα framework έχει ανάγκη να διαβάζει τα ίδια δεδομένα από τον δίσκο όταν τα χρειαζόμαστε, προστίθεται καθυστέρηση που μπορεί να επιβραδύνει την διαδικασία της εξερεύνησης και να περιορίσει τον αριθμό των πραγμάτων που μπορούν να δοκιμαστούν.

Τέλος, το έργο των επιστημόνων δεδομένων δεν έχει ολοκληρωθεί όταν έχει δημιουργηθεί ένα αποδοτικό μοντέλο. Εάν ο σκοπός της επιστήμης δεδομένων ήταν μόνο η παροχή πληροφορίας σε μη-επιστήμονες, και η αποθήκευση του μοντέλου ως μια λίστας «regression weights» σε ένα αρχείο τότε ο στόχος δεν έχει επιτευχθεί πραγματικά. Οι χρήσεις των μηχανών σύστασης προϊόντων και των συστημάτων που ανακαλύπτουν απάτες σε πραγματικό χρόνο, κορυφώνεται όταν χρησιμοποιούνται σε εφαρμογές. Σε αυτές τις εφαρμογές, τα μοντέλα γίνονται μέρος μιας υπηρεσίας, με την πιθανότητα να χρειαστεί να ξαναχτιστεί περιοδικά ή ακόμη και σε πραγματικό χρόνο. Σε αυτές τις περιπτώσεις, είναι χρήσιμο να γίνει διάκριση μεταξύ των analytics των εργαστηρίων και των analytics στην αγορά. Στο εργαστήριο, οι επιστήμονες δεδομένων ενασχολούνται με εξερευνητικά analytics, προσπαθούν να κατανοήσουν την φύση των δεδομένων και να τα οπτικοποιήσουν, καθώς και να πειραματίζονται με ποικίλες κλάσεις χαρακτηριστικών. Στην αγορά, για την δημιουργία μιας εφαρμογής, οι επιστήμονες δεδομένων ενασχολούνται με «operational analytics». Μετατρέπουν τα μοντέλα σε υπηρεσίες, οι οποίες δίνουν πληροφορίες για την λήψη αποφάσεων στον πραγματικό κόσμο. Επίσης, οι επιστήμονες δεδομένων παρακολουθούν την απόδοση των μοντέλων σε βάθος χρόνου, και σκέφτονται για το πώς να τα κάνουν πιο αποτελεσματικά και ακριβή. (Ryza, Laserson, Owen & Wills, 2015:2-3)

ΥΠΟΚΕΦΑΛΑΙΟ 5.2 Scikit

Το Scikit-learn είναι ένα module της Python, το οποίο ενσωματώνει ένα μεγάλο εύρος από τους πιο πρόσφατους (state-of-the-art) αλγορίθμους της μηχανικής μάθησης, με σκοπό την χρήση τους σε μεσαίου μεγέθους προβλημάτων είτε με επίβλεψη είτε χωρίς επίβλεψη (supervised και unsupervised problems). Το Scikit-learn εστιάζει στο να φέρει κοντά στους μη-εξειδικευμένους προγραμματιστές στην μηχανική μάθηση, μέσω της χρήσης μιας υψηλού επιπέδου και γενικού σκοπού γλώσσας προγραμματισμού. Δίνεται έμφαση στην ευκολία χρήσης, την επίδοση, την τεκμηρίωση, και στην συνοχή του API. Τέλος, υπάρχουν ελάχιστα dependencies και διανέμεται κάτω από την «simplified BSD» άδεια, ενθαρρύνοντας έτσι την χρήση τόσο σε ακαδημαϊκό όσο και σε εμπορικό επίπεδο.

ΥΠΟΚΕΦΑΛΑΙΟ 5.3 Spark vs Scikit

Το Scikit-Learn έχει καλύτερες υλοποιήσεις των πιο «ώριμων» αλγορίθμων, καθώς παρέχει μεγαλύτερη ευκολία χρήσης και ανάπτυξης κώδικα σε αυτό. Το ML Lib του Spark, με την σειρά του έχει επαρκείς αλγορίθμους για να μπορούν να δουλέψουν οι προγραμματιστές, αλλά αποδίδουν καλύτερα όταν βασίζονται πάνω σε κάποιο cluster (distributed setting). Όταν έρθει η στιγμή όπου ένας προγραμματιστής πρέπει να επιλέξει ανάμεσα σε Spark ή Scikit, η απόφαση του πρέπει να βασιστεί στο μέγεθος των datasets που επιθυμεί να αναλύσει. Αν τα datasets έχουν μεγέθη της τάξης των Gigabytes ή Terabyte, το Spark ML lib είναι καλύτερο για την επεξεργασία των δεδομένων. Αντίθετα αν τα datasets μπορούν να φορτωθούν στην κύρια μνήμη ενός μηχανήματος το Scikit είναι καλύτερη επιλογή γιατί αποφεύγονται κάποιες δυσκολίες που έρχονται μαζί με το ML Lib όπως της δημιουργίας cluster ή της αδυναμίας να οπτικοποιηθούν τα αποτελέσματα με ευκολία (<https://www.quora.com/Which-is-the-best-tool-for-machine-learning-MLlib-Apache-Spark-or-scikit-learn>)

Στα επόμενα υπο-κεφάλαια παρουσιάζονται οι αλγόριθμοι, πληροφορίες για τα dataset καθώς και οι κώδικες των εφαρμογών που υλοποιήθηκαν για την πτυχιακή αυτή. Τέλος, σε κάθε υπο-κεφάλαιο βρίσκονται αναλυτικοί πίνακες με τους χρόνους εκτέλεσης όλης της εφαρμογής (Total Time) και με τους χρόνους για την μάθηση και πρόβλεψη (Run/Predict Time).

ΥΠΟΚΕΦΑΛΑΙΟ 5.4 Εφαρμογές Kmeans

Το επιλεγμένο dataset με όνομα «KDDCUP04Bio» χρησιμοποιήθηκε για πρώτη φορά το 2004 σε έναν ετήσιο διαγωνισμό εξόρυξης δεδομένων και βασίζεται σε πειράματα σύγκρουσης υψηλής ενέργειας, και έχει 145751 εγγραφές με 74 χαρακτηριστικά η κάθε μια. Τέλος, αξίζει να αναφερθεί πως χρησιμοποιήθηκαν τα ίδια 20 κέντρα και στους δυο κώδικες, για την δίκαιη σύγκρισή τους (στον κώδικα του Spark εμφανίζονται μόνο τρία, με σκοπό τον περιορισμό του μεγέθους του).

Πίνακας 1 Συγκριτικό για τον αλγόριθμο Kmeans

	Max-iterations	Clusters	Run/Predict Time	Total Time
SPARK	1000	20	5,437 sec	8,487 sec
SCIKIT	1000	20	12,379 sec	19,507 sec

Κώδικας Spark

```
import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}
import org.apache.spark.mllib.linalg.Vectors

val starttime = System.currentTimeMillis

// Load and parse the data
val data = sc.textFile("CupBio.txt")
```

```
val parsedData = data.map(s => Vectors.dense(s.split(' ').map(_toDouble))).cache()

val loadcenters = Array(
  Vectors.dense (52,32.69,0.3,2.5,20,1256.8,-0.89,0.33,11,-55,267.2,0.52,0.05,-
2.36,49.6,252,0.43,1.16,-2.06,-33,-123.2,1.6,-0.49,-6.06,65,296.1,-0.28,-0.26,-3.83,-22.6,-
170,3.06,-1.05,-3.29,22.9,286.3,0.12,2.58,4.08,-33,-178.9,1.88,0.53,-7,-44,1987,-5.41,0.95,-4,-
57,722.9,-3.26,-0.55,-7.5,125.5,1547.2,-0.36,1.12,9,-37,72.5,0.47,0.74,-11,-8,1595.1,-1.64,2.83,-
2,-50,445.2,-0.35,0.26,0.76),
  Vectors.dense(58,33.33,0,16.5,9.5,608.1,0.5,0.07,20.5,-52.5,521.6,-1.08,0.58,-0.02,-
3.2,103.6,-0.95,0.23,-2.87,-25.9,-52.2,-0.21,0.87,-1.81,10.4,62,-0.28,-0.04,1.48,-17.6,-
198.3,3.43,2.84,5.87,-16.9,72.6,-0.31,2.79,2.71,-33.5,-11.6,-1.11,4.01,5,-57,666.3,1.13,4.38,5,-
64,39.3,1.07,-0.16,32.5,100,1893.7,-2.8,-0.22,2.5,-28.5,45,0.58,0.41,-19,-6,762.9,0.29,0.82,-3,-
35,140.3,1.16,0.39,0.73),
  .....
  Vectors.dense(88,25.26,1.72,52,-51,453.7,0.89,0.46,3.5,-65.5,419.5,-0.49,-0.27,-1.65,-
4.4,43.4,0.41,-0.5,-0.24,-15.1,-107.5,0.92,1.27,4.35,2.1,77.1,0.01,-0.42,-0.69,-11.2,-
138.2,2.48,0.8,3.7,4.7,100.8,-0.43,3.05,1.14,-12.8,13.9,-1.69,0.4,-2,-5,480.3,-1.24,3.64,0,-22,-
6.2,0.71,2.08,43.5,-20.5,522.6,0.93,0.08,-1.5,-38.5,272.6,-0.29,1.59,19,-28,393.4,0.55,0.93,11,-
41,315.6,-0.5,0,-0.7)
)

val defaultcenters = new KMeansModel(clusterCenters = loadcenters)

val starttime2 = System.currentTimeMillis
val clusters = new KMeans()
clusters.setK(20)
clusters.setInitialModel(defaultcenters)
clusters.setInitializationSteps(1000)
val model = clusters.run(parsedData)

model.predict(parsedData).coalesce(1,true).saveAsTextFile("Sparkoutput")
val partialtime = System.currentTimeMillis - starttime2
println("Elapsed run/predict time: %1d ms".format(partialtime))

val totalTime = System.currentTimeMillis - starttime
println("Elapsed time: %1d ms".format(totalTime))
```

Κώδικας Scikit

```
import time
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics import mean_squared_error
from numpy import *
```

```
start_time=time.time()
np.set_printoptions(threshold=np.inf)

X = loadtxt("CupBio.txt")
centers = loadtxt("CupBioCenters.txt")

start_time2=time.time()
kmeans = KMeans(n_clusters = 20, n_init = 1000 , init = centers)
kmeans.fit(X)
predictions = kmeans.predict(X)
elapsed_time2=time.time() - start_time2

print(predictions)

np.savetxt('scikit.txt',predictions)

elapsed_time = time.time() - start_time
print(elapsed_time)
print(elapsed_time2)
```

ΥΠΟΚΕΦΑΛΑΙΟ 5.5 Εφαρμογές SVM

Ο συγκεκριμένος αλγόριθμος βασίζεται στο «Susy» dataset το οποίο σημαίνει «SuperSymmetry» και αποτελείται από 100,000 εγγραφές με την κάθε μια να κατέχει 18 χαρακτηριστικά (το dataset είναι σε LIBSVM format). Οι εγγραφές είναι υπο-ατομικά σωματίδια που θεωρούνται είτε μποζόνια (bosons) είτε φερμιόνια (fermions) .Ο αλγόριθμος λοιπόν, καλείται να διαχωρίσει με βάση τα χαρακτηριστικά σε ποια κλάση ανήκει η κάθε εγγραφή. Το 60% dataset αφιερώνεται για την εκπαίδευση του αλγορίθμου ενώ το υπόλοιπο 40% για τον έλεγχο του . Στο τέλος των δυο υλοποιήσεων εμφανίζεται η μετρική Area under ROC , η οποία όσο πιο κοντά είναι στο «1» τόσο πιο αποδοτικός και αξιόπιστος είναι ο αλγόριθμος. Αν η τιμή ήταν κοντά στο «0,5» αυτό θα σήμαινε ότι ο αλγόριθμος επιλέγει την κλάση μιας εγγραφής σχεδόν στην τύχη. (Pentreath,2015:38)

Πίνακας 2 Συγκριτικό για τον αλγόριθμο SVM

	Number of iterations	Run/Predict time	Total Time	Area under ROC
SPARK	2000	5,381 sec	10,859 sec	0,8387
SCIKIT	-	208,359 sec	210,404 sec	0,7926

Κώδικας Spark

```
import org.apache.spark.mllib.classification.{SVMModel, SVMWithSGD}
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
```



```
import org.apache.spark.mllib.util.MLUtils

val starttime = System.currentTimeMillis

// Load training data in LIBSVM format.
val data = MLUtils.loadLibSVMFile(sc, "SUSYCo100k")

// Split data into training (60%) and test (40%).
val splits = data.randomSplit(Array(0.6, 0.4), seed = 11L)
val training = splits(0).cache()
val test = splits(1)

// Run training algorithm to build the model
val numIterations = 2000
val trainpredicttime = System.currentTimeMillis
val model = SVMWithSGD.train(training, numIterations)

// Clear the default threshold.
model.clearThreshold()

// Compute raw scores on the test set.
val scoreAndLabels = test.map { point =>
  val score = model.predict(point.features)
  (score, point.label)
}
val trainpredictstoptime = System.currentTimeMillis - trainpredicttime
// Get evaluation metrics.
val metrics = new BinaryClassificationMetrics(scoreAndLabels)
val auROC = metrics.areaUnderROC()

println("Area under ROC = " + auROC)

val totalTime = System.currentTimeMillis - starttime
println("Elapsed time: %1d ms".format(totalTime))
println("Elapsed Train/Predict time: %1d ms".format(trainpredictstoptime))
```

Κώδικας Scikit

```
import numpy as np
import time

from sklearn import svm
from sklearn.cross_validation import train_test_split
from sklearn.metrics import roc_auc_score

start_time=time.time()

A = np.loadtxt(fname = "Susy100k", delimiter = ",")

X_train, X_test, y_train, y_test = train_test_split(A[:,1:18], A[:,0], test_size=0.4, random_state=0)
```

```

start_time2=time.time()
y_train= y_train.astype(int)
y_test= y_test.astype(int)

svc1 = svm.SVC().fit(X_train,y_train)
svc1.score(X_test,y_test)

auROC = roc_auc_score(y_test, svc1.predict(X_test))

print(auROC)
elapsed_time2 = time.time() - start_time2
elapsed_time = time.time() - start_time
print(elapsed_time)
print(elapsed_time2)

```

ΥΠΟΚΕΦΑΛΑΙΟ 5.6 Εφαρμογές Regression

Ο συγκεκριμένος κώδικας βασίζεται σε ένα τεχνητό dataset που δημιουργήθηκε από τον J. Friedman, και έχει 40768 εγγραφές με 10 χαρακτηριστικά στην κάθε εγγραφή. Οι εγγραφές δημιουργήθηκαν με βάση την εξής εξίσωση:

$$Y = 10\sin(\pi X_1 X_2) + 20(X_3 - 0.5)^2 + 10X_4 + 5X_5 + \sigma(0.1) \quad (5.1)$$

Πίνακας 3 Συγκριτικό για τον αλγόριθμο Regression

	Number of iterations	Run/Predict time	Total Time	MSE
SPARK	1000	1,824 sec	5,013 sec	0,33351
SCIKIT	1000	0,067 sec	0,5245 sec	-

Κώδικας Spark

```

import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LinearRegressionModel
import org.apache.spark.mllib.regression.LinearRegressionWithSGD
import org.apache.spark.mllib.linalg.Vectors

val starttime = System.currentTimeMillis

// Load and parse the data
val data = sc.textFile("fried_delve.data")
val parsedData = data.map { line =>
  val parts = line.split(',')
  LabeledPoint(parts(0).toDouble, Vectors.dense(parts(1).split(' ').map(_toDouble)))
}

```

```
}.cache()
```

```
// Building the model
val numIterations = 1000
val step = 0.0000001
val algorithm = new LinearRegressionWithSGD()
algorithm.setIntercept(true)
algorithm.optimizer.setNumIterations(numIterations)
algorithm.optimizer.setStepSize(step)
val runandpredicttime = System.currentTimeMillis
val model = algorithm.run(parsedData)

// Evaluate model on training examples and compute training error
val valuesAndPreds = parsedData.map { point =>
  val prediction = model.predict(point.features)
  (point.label, prediction)
}
val runandpredicttotaltime = System.currentTimeMillis - runandpredicttime

val MSE = valuesAndPreds.map{case(v, p) => math.pow((v - p), 2)}.mean()
println("training Mean Squared Error = " + MSE)

val totalTime = System.currentTimeMillis - starttime
println("Elapsed time: %1d ms".format(totalTime))
println("Run And Predict time: %1d ms".format(runandpredicttotaltime))
```

Κώδικας Scikit

```
from sklearn.linear_model import Ridge
import numpy as np
from numpy import *
import time

start_time=time.time()

y=loadtxt("fried_delvefirstcol")
X=loadtxt("fried_delverestcol",delimiter=',')

start_fit_time=time.time()
ridge = Ridge(alpha = 1.0,max_iter = 1000)
ridge.fit(X, y)
elapsed_fit_time = time.time() - start_fit_time

elapsed_time = time.time() - start_time
print(elapsed_time)
print(elapsed_fit_time)
```

ΣΥΜΠΕΡΑΣΜΑΤΑ

Με την παρούσα πτυχιακή εργασία έγινε μια ολοκληρωμένη και αντικειμενική παρουσίαση του Spark και των δυνατοτήτων του. Η πλατφόρμα του Spark είναι σε θέση να προσφέρει πολλές δυνατότητες για την επεξεργασία "big data" και η ιδέα της ανοικτής διακίνησης λογισμικού που υποστηρίζει προσφέρει ελευθερία και το απαραίτητο κίνητρο προς τους προγραμματιστές να ασχοληθούν με αυτό. Επίσης, διαπιστώθηκε η ταχύτητα που επεξεργάζεται τα δεδομένα σε σχέση με τον άμεσο ανταγωνιστή του στην μηχανική μάθηση, όπου το Spark ήταν με διαφορά το ταχύτερο. Με το πέρασμα του χρόνου, ολοένα και περισσότεροι αλγόριθμοι θα είναι διαθέσιμοι, καθιστώντας το πιθανά στην κορυφή των διαθέσιμων πλατφόρμων για την επεξεργασία "big data". Τέλος, η συγγραφή του κώδικα έγινε με γνώμονα την απλότητα και την πληρότητα, με σκοπό την παρουσίαση κάποιων βασικών αλγορίθμων της μηχανικής μάθησης.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλία

Frampton, M. (2015). Mastering Apache Spark. United Kingdom: Packt Publishing Ltd.

Garreta, R. & Moncecchi, G. (2013). Learning scikit-learn: Machine learning in Python. United Kingdom: Packt Publishing Ltd.

Karau, H., Konwinski, A., Wendell, P. & Zaharia, M. (2015). Learning Spark. United States of America: O'Reilly.

Pentreath, N. (2015). Machine learning with Spark. United Kingdom: Packt Publishing Ltd.

Ryza, S., Laserson, U., Owen, S. & Wills, J. (April 2015). Advanced Analytics with Spark. United States of America: O'Reilly.

Scott, J. (2015). Getting started with Apache Spark. United States of America: MapR Technologies.

White, T. (May 2012). Hadoop the definitive guide. United States of America: O'Reilly.

Άρθρα

Caruana, R., Joachims, T. & Backstrom, L. (2004). KDD-Cup 2004: Results and Analysis. United States of America.

Hindman, B., Konwinski, A. & Zaharia, M. (2010). Mesos: A platform for fine-grained resource sharing in the data center. United States of America.

Zaharia, M., Chowdhury, M., Franklin, M., Shenker, S. & Stoica, I. (2010). Spark: Cluster Computing with Working Sets. United States of America: University of California.

Zaharia, M., Mosharaf, C., Tathagata, D. & Ankur, D. (2012). Resilient Distributed Datasets: A fault tolerant abstraction of in-memory cluster computing. United States of America.

Zaharia, M., Tathagata, D., Haoyuan, L., Hunter, T., Shenker, S. & Stoica, I. (2013). Discretized Streams: Fault-Tolerant streaming computation at Scale. United States of America.

Shindler, M., Wong, A. & Meyerson, A. (2012). Fast and Accurate k-means For large datasets. United States of America.

Περιοδικά

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V. & Thirion, B. (2011, 11 Οκτωβρίου). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 12

Πτυχιακή Εργασία

Παπαδοπούλου, Ε. (2016). Scikit: Βιβλιοθήκη αλγορίθμων μηχανικής μάθησης στην γλώσσα Python. Αλεξάνδρειο Τεχνολογικό Εκπαιδευτικό Ίδρυμα Θεσσαλονίκης, Θεσσαλονίκη.

Ιστοσελίδες

(2016). Dezyre. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <https://www.dezyre.com/article/scala-vs-python-for-apache-spark/213>.

(2016). Regression DataSets. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html>.

(2013). Measuring elapsed time with the Time module. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <http://stackoverflow.com/questions/3620943/measuring-elapsed-time-with-the-time-module>.

(2017). Spark Documentation. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <http://spark.apache.org/documentation.html>.

(2016). Apache Spark Cluster overview. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <http://spark.apache.org/docs/latest/cluster-overview.html>.

(2013). How to Save Time by Using Snapshots in VirtualBox. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <http://www.howtogeek.com/150258/how-to-save-time-by-using-snapshots-in-virtualbox/>.

(2014). SUSY Data Set. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <https://archive.ics.uci.edu/ml/datasets/SUSY>.

(2017). Clustering - RDD-based API. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <http://spark.apache.org/docs/latest/mllib-clustering.html>.

(2017). Clustering datasets. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <http://cs.joensuu.fi/sipu/datasets/>.

(2017). Scikit-learn: Machine Learning in Python. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <http://scikit-learn.org/stable/>.

(2016). How is scikit-learn compared with Apache Spark's MLlib?. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <https://www.quora.com/How-is-scikit-learn-compared-with-Apache-Sparks-MLlib>.

(2016). Which is the best tool for machine learning: MLlib (Apache-Spark) or scikit-learn?. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <https://www.quora.com/Which-is-the-best-tool-for-machine-learning-MLlib-Apache-Spark-or-scikit-learn>.

(2017). Journal of Machine Learning Research. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <http://www.jmlr.org/>.

(2016). Apache Spark Future. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <https://0x0fff.com/apache-spark-future/>.

(2016). Linear Methods - spark.mllib. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <https://spark.apache.org/docs/1.6.1/mllib-linear-methods.html>.

(2017). LIBSVM FAQ. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από http://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html#Q3: Data_preparation.

(2016). Import an array in python. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <http://stackoverflow.com/questions/1796597/import-an-array-in-python>.

(2015). Diving into Apache Spark Streaming's Execution Model. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <https://databricks.com/blog/2015/07/30/diving-into-apache-spark-streamings-execution-model.html>.

(2016). Spark SQL - Introduction. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από https://www.tutorialspoint.com/spark_sql/spark_sql_introduction.htm.

(2016). GraphX Programming Guide. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <https://spark.apache.org/docs/0.9.0/graphx-programming-guide.html>.

(2017). SparkContext — Entry Point to Spark (Core). Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-sparkcontext.html>.

(2017). Monitoring and Instrumentation. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <https://spark.apache.org/docs/1.6.1/monitoring.html>.

(2017). Web UI — Spark Application's Web Console. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-webui.html>.

(2013). Java vs. Python (1): Simple Code Examples. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <http://www.programcreek.com/2012/04/java-vs-python-why-python-can-be-more-productive/>.

(2015). Six reasons why I recommend scikit-learn. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <https://www.oreilly.com/ideas/six-reasons-why-i-recommend-scikit-learn>.

(2017). Welcome to Apache™ Hadoop®!. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <http://hadoop.apache.org/>.

(2017). Object-Oriented Meets Functional. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <http://www.scala-lang.org/>.

(2017). Acyclic Digraph. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <http://mathworld.wolfram.com/AcyclicDigraph.html>.

(2016). How DAG works under the covers in RDD?. Ανακτήθηκε 5 Φεβρουαρίου, 2017, από <http://stackoverflow.com/questions/25836316/how-dag-works-under-the-covers-in-rdd>.

ΠΑΡΑΡΤΗΜΑΤΑ

Το παράδειγμα SparkPi

```
1  import sys
2  from random import random
3  from operator import add
4
5  from pyspark import SparkContext
6
7  if __name__ == "__main__":
8      """
9          Usage: pi [partitions]
10         """
11         sc = SparkContext(appName="PythonPi")
12         partitions = int(sys.argv[1]) if len(sys.argv) > 1 else 2
13         n = 100000 * partitions
14
15         def f(_):
16             x = random() * 2 - 1
17             y = random() * 2 - 1
18             return 1 if x ** 2 + y ** 2 < 1 else 0
19
20         count = sc.parallelize(xrange(1, n + 1), partitions).map(f).reduce(add)
21         print "Pi is roughly %f" % (4.0 * count / n)
22
23     sc.stop()
24
```

Για να εκτελέσουμε τον κώδικα, τρέξαμε την εντολή:

/Desktop/spark-1.6.1-bin-hadoop2.6/bin\$./run-example SparkPi 10

Όπως βλέπουμε, ο αριθμός 10 αντιπροσωπεύει τον αριθμό των τμημάτων που έχει δημιουργήσει το πρόγραμμα του Spark. Η δουλειά για τον υπολογισμό του Πι έχει διαιρεθεί σε 10 διεργασίες. Το Πι υπολογίζεται μέσω ενός επαναληπτικού αλγορίθμου.

Έξοδος:

Pi is roughly 3.141216

ΟΔΗΓΟΣ ΧΡΗΣΗΣ ΛΟΓΙΣΜΙΚΟΥ

Εγκατάσταση του VirtualBox της Oracle και δημιουργία Virtual Machine



Από το επίσημο site της oracle <https://www.virtualbox.org/> κατεβάζουμε το VirtualBox 5.1 . Για την εγκατάσταση του δεν χρειάζονται ιδιαίτερες οδηγίες.

Για την δημιουργία Virtual Machine ακολουθούμε τα εξής βήματα:

1. Πατάμε στο κουμπί New και ορίζουμε ένα όνομα για το μηχάνημά μας και πατάμε Next.
2. Σε αυτό το βήμα ορίζουμε το μέγεθος της RAM. Είναι συνετό να ορίσουμε το μέγεθος στο μέγιστο δυνατό “πράσινο” και πατάμε Next.
3. Επιλέγουμε την επιλογή “Create a virtual hard disk now” και πατάμε Create.
4. Επιλέγουμε την επιλογή “VDI (VirtualBox Disk Image)” και πατάμε Next.
5. Επιλέγουμε την επιλογή “Dynamically allocated” η οποία έχει σαν αποτέλεσμα το μηχάνημα που θα δημιουργήσουμε να αυξάνει δυναμικά τον χώρο που καταλαμβάνει στον δίσκο. Πατάμε Next.
6. Σε αυτό το βήμα ορίζουμε το αρχικό μέγεθος του δίσκου. Ένα ικανοποιητικό μέγεθος είναι 20-30 GB. Πατάμε Create.
7. Επιλέγουμε το μηχάνημα που δημιουργήσαμε και πατάμε Start.
8. Στο πλαίσιο που εμφανίζεται διαλέγουμε το ISO αρχείο του λειτουργικού συστήματος που θέλουμε να κάνουμε εγκατάσταση. Στα πλαίσια της πτυχιακής χρησιμοποιήθηκε Ubuntu 15 και το ISO μπορεί να βρεθεί εδώ: <https://www.ubuntu.com/download/desktop>
9. Ακολουθούμε τα βήματα του λειτουργικού που διαλέξαμε για την ολοκλήρωση της εγκατάστασης.
10. Τέλος μετά την ολοκλήρωση της εγκατάστασης του λειτουργικού θα πρέπει να προσθέσουμε τα λεγόμενα “Guest additions”. Στο Ubuntu γίνονται με την εντολή : `sudo apt-get install virtualbox-guest-dkms`

Εγκατάσταση Python 2.7 (με anaconda)

Από το site της Continuum Analytics <https://www.continuum.io/downloads> κατεβάζουμε τον installer “Python 2.7 version”. Όταν κατέβει το αρχείο τρέχουμε την εντολή `bash Anaconda-latest-Linux-x86_64.sh`

Εγκατάσταση Scikit

Σε ένα terminal τρέχουμε την εντολή: `conda install scikit-learn`

Εγκατάσταση Spark

Από το επίσημο site του Apache Spark <http://spark.apache.org/downloads.html> επιλέγουμε : 1) 1.6.3 2) Pre-built for Hadoop 2.6 και στην συνέχεια κάνουμε κλικ στο link με όνομα `spark-1.6.3-bin-hadoop2.6.tgz`

Οδηγίες για το setup των cluster manager στο Spark

Ρύθμιση του Mesos για το Spark

Για την εγκατάσταση του Apache Mesos χρειάζεται να γίνει Download μιας έκδοσης τους από το <http://mesosphere.io/downloads/>. Για οποιοδήποτε άλλο σύστημα υπάρχουν λεπτομερείς οδηγίες εγκατάστασης εδώ : <http://mesos.apache.org/gettingstarted/>

Επιπρόσθετα, για να μπορέσει το Mesos να δουλέψει στο Spark, το binary package του Spark θα πρέπει να είναι αποθηκευμένο σε έναν χώρο προσβάσιμο από το Mesos, καθώς και να γίνουν οι απαραίτητες ρυθμίσεις στο driver πρόγραμμα.

Για την χρήση του Mesos σε «cluster mode» (υπάρχει διαθέσιμο και σαν client mode), θα πρέπει να γίνει εκκίνηση στο cluster το «MesosClusterDispatcher». Για να πραγματοποιηθεί αυτό πρέπει να εκτελεστεί το `sbin/start-mesos-dispatcher.sh` script. Υπάρχει η δυνατότητα προσθήκης παραμέτρων για τον ορισμό των Mesos master URL και της προεπιλεγμένης πύλης. Όταν εκτελεστεί το script , έχει ξεκινήσει το MesosClusterDispatcher σαν «daemon» στο host μηχανήμα.

Από ένα client μηχανήμα, μπορούμε να ορίσουμε μια διεργασία (job) για το cluster του Mesos εκτελώντας `spark-submit` και προσδιορίζοντας το master URL για το MesosClusterDispatcher (πχ: `mesos://dispatcher:7077`).

Για παράδειγμα:

```
./bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master mesos://207.184.161.138:7077 \
--deploy-mode cluster \
```

```
--supervise \  
--executor-memory 20G \  
--total-executor-cores 100 \  
http://path/to/examples.jar \  
1000
```

Τέλος υπάρχει η δυνατότητα προβολής της κατάστασης των driver μέσω του Spark cluster Web UI. Περισσότερες πληροφορίες: <http://spark.apache.org/docs/latest/running-on-mesos.html>

Πως να ρυθμίσεις το Mesos να δουλεύει με το YARN

Αρχικά, θα πρέπει να εξασφαλιστεί ότι το HADOOP_CONF_DIR ή το YARN_CONF_DIR έχει πρόσβαση στον φάκελο που περιέχει τα configuration files για το Hadoop cluster. Αυτά τα configuration files χρησιμοποιούνται για να γίνεται εγγραφή δεδομένων στο HDFS, καθώς και για την σύνδεση με τον YARN ResourceManager. Το configuration αρχείο, θα διανεμηθεί στον YARN cluster με σκοπό όλα τα υποσυστήματα της εφαρμογής να βασίζονται στο ίδιο αρχείο. Στην περίπτωση που το αρχείο αυτό περιέχει «Java system properties» ή «environment variables», τα οποία δεν διαχειρίζονται από το YARN, θα πρέπει να οριστούν μέσα στο Spark application's configuration (driver, executors, καθώς και στο AM όταν εργαζόμαστε σε client mode).

Υπάρχουν δυο τρόποι για την ανάπτυξη εφαρμογών Spark που να βασίζονται στο Yarn: το cluster mode και το client mode.

Για την εκτέλεση μιας εφαρμογής σε cluster mode:

```
$ ./bin/spark-submit --class path.to.your.Class --master yarn --deploy-mode cluster [options] <app jar> [app options]
```

Για παράδειγμα:

```
$ ./bin/spark-submit --class org.apache.spark.examples.SparkPi \  
--master yarn \  
--deploy-mode cluster \  
--driver-memory 4g \  
--executor-memory 2g \  
--executor-cores 1 \  
--queue thequeue \  
lib/spark-examples*.jar \  
10
```

Για την εκτέλεση εφαρμογών σε client mode, κάνουμε το ίδιο με τα παραπάνω, με την διαφορά πως πρέπει να αντικαταστήσουμε το «cluster» με το «client». Αν θέλουμε να εκτελέσουμε το «spark-shell in client mode», τότε εκτελούμε το παρακάτω:

```
$ ./bin/spark-shell --master yarn --deploy-mode client
```

Περισσότερες πληροφορίες: <http://spark.apache.org/docs/latest/running-on-yarn.html>