Πανεπιστήμιο Πελοποννήσου

Σχολή Μηχανικών

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Π.Μ.Σ: «Τεχνολογίες και Συστήματα Ευρυζωνικών Εφαρμογών και Υπηρεσιών»

Διπλωματική Εργασία

Αναλυτική Μεγάλων Δεδομένων. Διερεύνηση Τεχνολογιών και Μεθοδολογιών

Ιωάννης Μπάλλας
ΑΜ: 67052201631

Επιβλέπων Καθηγητής:

Δρ. Βασίλειος Ταμπακάς

Πάτρα 2020

University of Peloponnese

School of Engineering

Department of Electrical and Computer Engineering

Master Program in Technologies and Infrastructures for Broadband Applications and Services

Master Thesis

Big Data Analytics. Investigation of Technologies and Methodologies

Ioannis Ballas
AM: 67052201631

Supervisor:

Dr. Vasileios Tampakas

Patras, Greece, 2020

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή
Πάτρα, 21 Μαΐου 2020

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Ταμπακάς Βασίλειος, ……
2. Μιχαήλ Παρασκευάς, ……
3. Ιωάννης Τζήμας, ..……

Υπεύθυνη Δήλωση Φοιτητή

Βεβαιώνω ότι είμαι συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τη συγκεκριμένη εργασία. Η έγκριση της διπλωματικής εργασίας από το Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Πελοποννήσου δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα εκ μέρους του Τμήματος.

Αφιέρωση

Αφιερώνεται στην κόρη μου και στα αγαπημένα μου πρόσωπα που έφυγαν νωρίς από τη ζωή.

Μπάλλας Ιωάννης

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή κ. Βασίλειο Ταμπακά για την άριστη συνεργασία που είχαμε όλα αυτά τα χρόνια και για την ευκαιρία που μου έδωσε να ασχοληθώ με το θέμα αυτό καθώς και για την υποστήριξη και καθοδήγηση που μου παρείχε κατά την εκπόνηση της διπλωματικής μου εργασίας.

Περίληψη

Στην παρούσα διπλωματική εργασία αρχικά γίνεται μια εισαγωγή στις βασικές αρχές και έννοιες των δεδομένων ευρείας κλίμακας (μεγάλα δεδομένα) και στην αναλυτική τους.

Στη συνέχεια γίνεται μια συγκριτική παρουσίαση των Apache Hadoop και Apache Spark και στους τρόπους που αντιμετωπίζουν τις προκλήσεις που εισάγουν τα μεγάλα δεδομένα.

Ακολουθεί μια εκτενής αναφορά στις δυο μεθόδους με τις οποίες μπορεί να διαμορφωθεί και να εκτελεστεί ένα πρόγραμμα Spark και στις δομές δεδομένων RDD και DataFrame του Apache Spark.

Τέλος η παρούσα διπλωματική εργασία επικεντρώνεται στην διεξοδική ανάλυση της διαδικασίας εγκατάστασης του Spark τόσο σε συστάδα με διαχειριστή πόρων το Hadoop/Yarn όσο και σε συστάδα με τον εγγενή διαχειριστή συστάδας Spark (Spark Standalone Cluster Manager). Επίσης γίνεται μια εκτενής παρουσίαση του Διακομιστή Ιστορικού Spark (Διαδικτυακή Διεπαφή Χρήστη / Web UI) και των τεχνικών δυναμικής κατανομής πόρων καθώς και στους τρόπους διαμόρφωσής του προκειμένου να επιτευχθεί η ορθή υλοποίησή τους.

Abstract

In the present dissertation, an introduction to the concepts of Big Data and Big Data Analytics is initially given. Then, it is made a comparative presentation of Apache Hadoop and Apache Spark frameworks and their techniques facing the big data challenges.

After that, an extensive report is given, concerning the two methods by which a Spark program can be configured/executed as well as the RDDs and DataFrames data structures of Apache Spark.

Finally, the dissertation focuses on the process of Spark installation considering, in depth, two cases: (a) installation in the Yarn resource manager Hadoop cluster and (b) installation in the Spark Standalone resource manager cluster. There is also an extensive presentation of the Spark history server (Web UI) and the dynamic allocation of Spark resources as well as Spark's configuration in order to achieve their implementation and ways of testing their proper implementation.

Περιεχόμενα

Κατάλογος Πινάκων

Κατάλογος Σχημάτων

Κεφάλαιο 1
Εισαγωγή

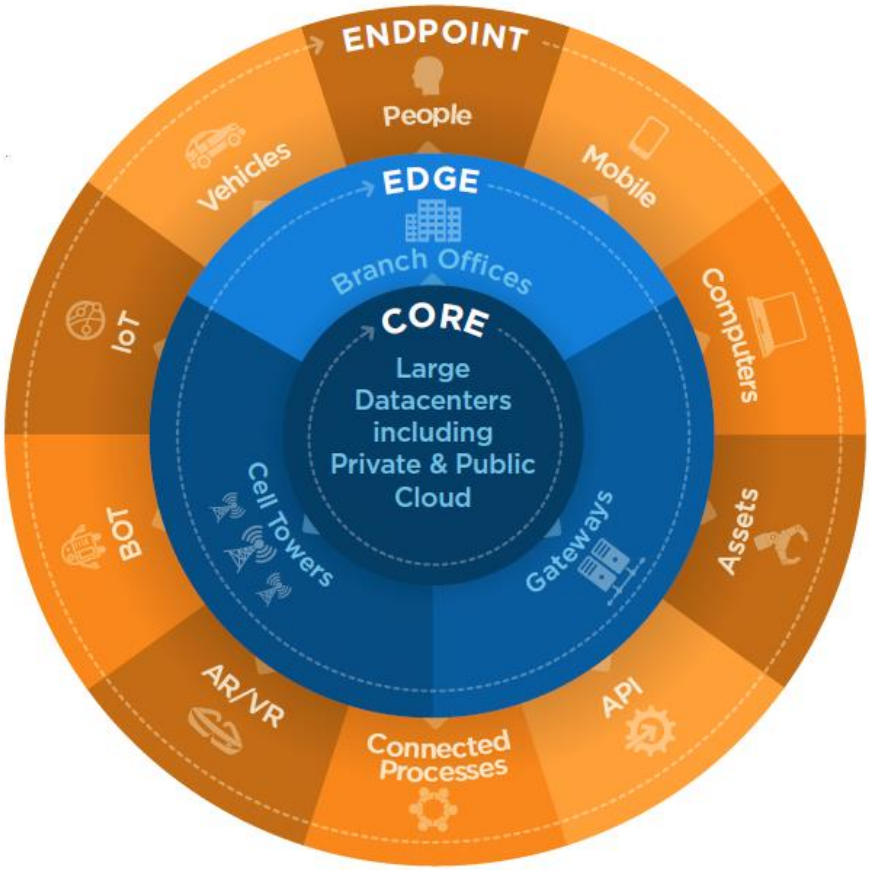
Η ψηφιοποίηση του Κόσμου

Η πρόοδος της τεχνολογίας τα τελευταία έτη κατέστησε προσιτό το κόστος αποθήκευσης μεγάλου όγκου δεδομένων. Το γεγονός αυτό είχε ως αποτέλεσμα την χρήση νέων τρόπων μετασχηματισμού, επεξεργασίας και ανάλυσης δεδομένων για την εξαγωγή πολύτιμης γνώσης απ' αυτά. Τα περισσότερα από τα δεδομένα αυτά δημιουργήθηκαν από αισθητήρες ενσωματωμένους σε συσκευές και μηχανές και η ανάλυσή τους βοηθά τις επιχειρήσεις στη βελτίωση της απόδοσης και στην αύξηση της παραγωγικότητάς τους και τους καταναλωτές στη βελτίωση της ποιότητας ζωής τους.

Σύμφωνα με άρθρο του World Economic Forum [12] το ψηφιακό σύμπαν αναμένεται να φτάσει τα 44 ZB (1 zettabyte = 10^{21} bytes) μέχρι το 2020. Ενώ η IDC (International Data Corporation) [13] προβλέπει :

- αύξηση του ψηφιακού σύμπαντος από 33 ZB το 2018 στα 175 ZB το 2025.
- Το 2025 το 49% των παγκόσμια αποθηκευμένων δεδομένων (η IDC χρησιμοποιεί τον όρο DataSphere) - θα βρίσκονται σε περιβάλλοντα δημόσιου υπολογιστικού νέφους (public cloud environments).
- Τα δισεκατομμύρια συσκευών IoT που θα είναι συνδεδεμένες σε όλη την υδρόγειο, αναμένεται να δημιουργήσουν πάνω από 90 ZB δεδομένων το 2025.
- Σχεδόν το 30% των δεδομένων που παράγονται θα καταναλώνονται σε πραγματικό χρόνο έως το 2025.

Η IDC ορίζει το DataSphere ως αποτελούμενο από τρεις τοποθεσίες. Πρώτον, υπάρχει ο "πυρήνας" (core), ο οποίος περιλαμβάνει τα παραδοσιακά και τα υπολογιστικού νέφους κέντρα δεδομένων. Στη συνέχεια, υπάρχει η «Παρυφή» (Edge), η οποία περιλαμβάνει την υποδομή για επιχειρήσεις, όπως πύργους κυψελών, αίθουσες διακομιστών (dataserver rooms), κλπ. Τέλος, υπάρχουν τα "τελικά σημεία" (End-points), τα οποία περιλαμβάνουν υπολογιστές, έξυπνα τηλέφωνα και συσκευές IoT, AR/VR (Augmented Reality/Virtual Reality), κ.λπ. (βλέπε Σχήμα 1.1).



Πηγή: [13]
Σχήμα 1-1: Το DataSphere (IDC) αποτελούμενη από τον Πυρήνα (core), την Παρυφή (Edge) και το Τελικό Σημείο (Endpoint) - Η διάδοση των δεδομένων γίνεται από τα τελικά σημεία προς τον πυρήνα και αντίστροφα

Ο όρος **μεγάλα δεδομένα (Big Data)** άρχισε να πρωτοεμφανίζεται στα τέλη της δεκαετίας του 1990, η επικράτησή του και η σημασία του αυξήθηκε εκθετικά με την πάροδο των χρόνων. Σήμερα τα μεγάλα δεδομένα θεωρούνται συχνά αναπόσπαστο στοιχείο της στρατηγικής δεδομένων μιας επιχείρησης, η οποία διασφαλίζει ότι τα δεδομένα διαχειρίζονται και χρησιμοποιούνται ως περιουσιακά στοιχεία με αποκλειστικό σκοπό την ανάλυσή τους και την εξαγωγή πολύτιμης γνώσης που θα συμβάλλει στη λήψη βέλτιστων αποφάσεων.

Ο όρος Μεγάλα Δεδομένα (Big Data) αναφέρεται σε σύνολα δεδομένων που χαρακτηρίζονται από τεράστιο όγκο (volume), υψηλή ταχύτητα (velocity) και μεγάλη ποικιλία (ετερογένεια) (variety). Εν ολίγοις, το μέγεθος των δεδομένων και / ή η ταχύτητά τους καθιστά αδύνατη την αποθήκευση ή την αποτελεσματική επεξεργασία τους από τα παραδοσιακά εργαλεία διαχείρισης δεδομένων.

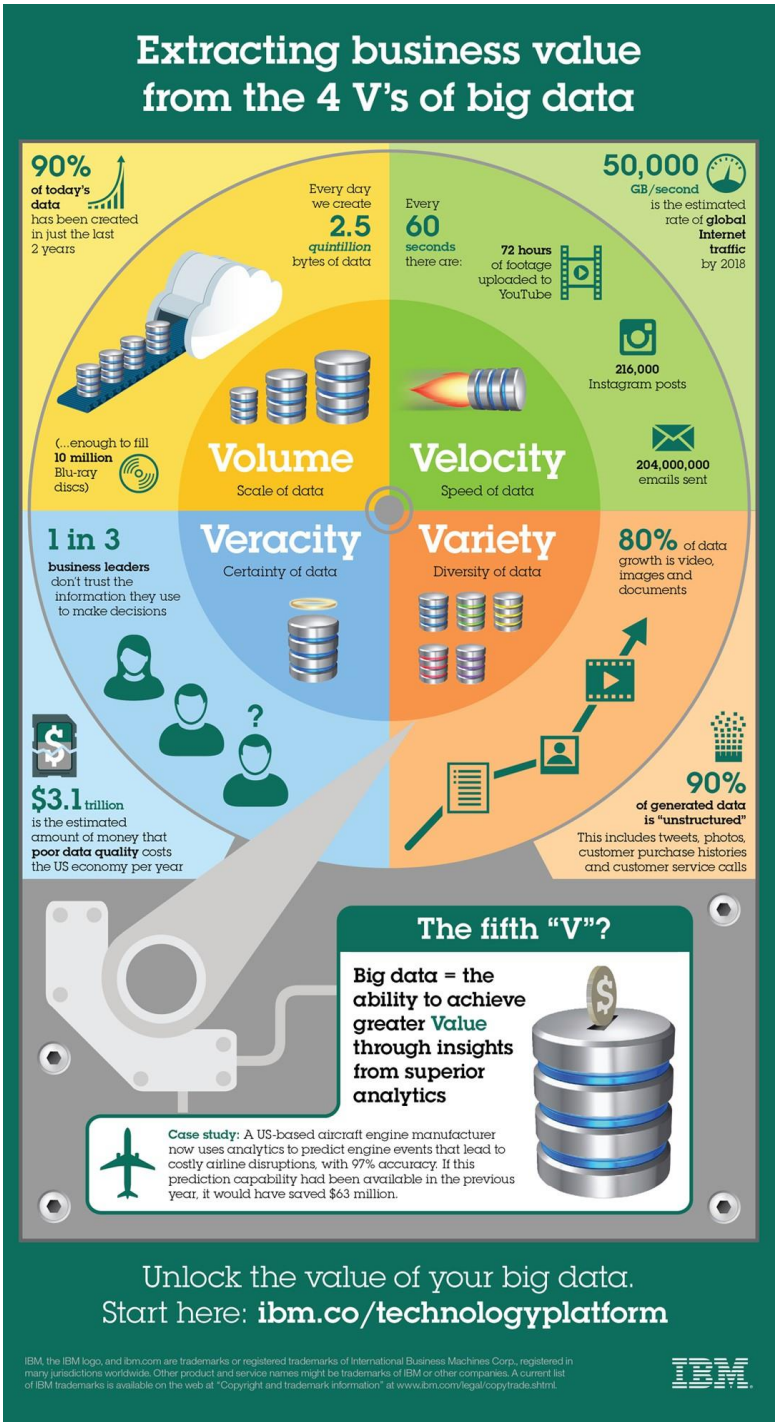
Ο ***όγκος (volume)*** δεδομένων υποδεικνύει τον συνολικό όγκο των δεδομένων που υφίστανται επεξεργασία και προκειμένου τα δεδομένα να χαρακτηριστούν ως "μεγάλα", ο όρος αυτός θα πρέπει να είναι της τάξης αρκετών terabyte και άνω.

Η ***ταχύτητα (velocity)*** υποδηλώνει τη ταχύτητα με την οποία παράγονται, συλλέγονται και αναλύονται τα δεδομένα.

Η ***ποικιλία (variety)*** δείχνει το εύρος των τύπων δεδομένων που υφίστανται επεξεργασία και ενσωμάτωση. Όταν πρόκειται για μεγάλα δεδομένα, δεν έχουμε μόνο να χειριστούμε δομημένα δεδομένα, αλλά και ημιδομημένα και κυρίως μη δομημένα δεδομένα. Είναι βέβαιο ότι με την πάροδο του χρόνου, τα τρία αυτά χαρακτηριστικά θα εξακολουθήσουν να μεγαλώνουν διαρκώς σε μέγεθος.

Τα τρία V – Volume,Velocity, Variety – αποτελούν τα καθιερωμένα χαρακτηριστικά για τον ορισμό των big data. *Κατά καιρούς έχει γίνει προσπάθεια από προμηθευτές λύσεων Big Data να προστεθούν στα χαρακτηριστικά των Big Data* ορισμένα ακόμα που *αρχίζουν με το γράμμα V, επεκτείνοντας το μοντέλο των 3 V. Σύμφωνα με την IBM (Veracity of data, 2013, [14]) ένα επιπλέον χαρακτηριστικό είναι* ***η αξιοπιστία (veracity)*** τους. Προέβλεπε ότι έως το 2015, το 80% των παγκόσμιων δεδομένων θα ήταν αβέβαια. Οι τρόποι που προτείνει για την μείωση της αβεβαιότητας είναι:

- Η ενσωμάτωση δεδομένων που σχετίζονται μεταξύ τους.
- Η χρήση εργαλείων όπως η επεξεργασία της φυσικής γλώσσας (NLP), η προγνωστική αναλυτική πελατών και η ανάλυση συναισθήματος των δεδομένων από μέσα κοινωνικής δικτύωσης συντελεί στην ανάλυση και εξαγωγή χρήσιμων πληροφοριών από όλα τα διαθέσιμα δεδομένα.



Πηγή: https://www.ibmbigdatahub.com/infographic/extracting-business-value-4-vs-big-data

Σχήμα 1-2: Τα 4V (ή 5V;) των Big Data κατά την IBM

- Επίσης μαθηματικές προσεγγίσεις όπως η στοχαστική συλλογιστική ή οι προσομοιώσεις Monte Carlo λαμβάνουν υπόψη την αβεβαιότητα για να δώσουν προγνωστικές συστάσεις.

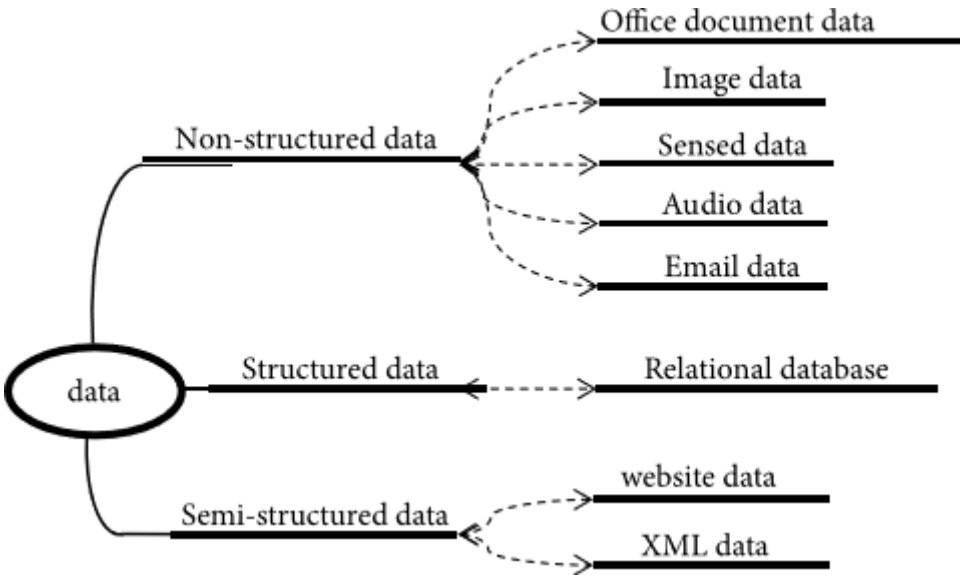
Αυτό που είναι ζωτικής σημασίας για την κατανόηση των μεγάλων δεδομένων είναι η βρώμικη και θορυβώδης φύση τους και το ποσό της εργασίας που απαιτείται για την μετατροπή τους

σε ακριβή δεδομένα πριν αρχίσει η ανάλυσή τους. Αποτελεί πρόκληση η απελευθέρωση του δυναμικού που εμπεριέχουν τα μεγάλα δεδομένα. Η αφαίρεση στοιχείων όπως μεροληψία (bias), ανωμαλιών ή ασυνεπειών και διπλοτύπων είναι μερικές από τις τεχνικές που χρησιμοποιούνται για τη βελτίωση της ακρίβειας των μεγάλων δεδομένων.

Άλλα προτεινόμενα χαρακτηριστικά είναι ***η μεταβλητότητα*** (***Variability***) των δεδομένων λόγω του αριθμού των ακραίων τιμών (outliers) που βρίσκονται στα δεδομένα, των πολλαπλών διαφορετικών τύπων δεδομένων και πηγών καθώς και της ασυνέπειας στην ταχύτητα φόρτωσης των δεδομένων, **η Εγκυρότητα** (***Validity***), **η *πτητικότητα (Volatility)*** *η οποία αναφέρεται στον ρυθμό μεταβολής των δεδομένων, το χρονικό διάστημα που τα δεδομένα είναι επίκαιρα/χρήσιμα καθώς και στο χρονικό διάστημα που χρειάζεται να παραμείνουν αποθηκευμένα,* ***η οπτικοποίηση (Visualization)*** η οποία αναφέρεται στην παρουσίαση των μεγάλων δεδομένων κατά τρόπο εύκολα κατανοητό *και τέλος* ***η αξία (Value****) η οποία αναφέρεται στην αποτίμηση του κέρδους που μπορούν να επιφέρουν τα δεδομένα στις επιχειρήσεις, όμως δεν έχουν γίνει ευρέως αποδεκτά.*

Διάκριση των Μεγάλων Δεδομένων σε Δομημένα – Ημιδομημένα – Μη δομημένα

Τα **δομημένα δεδομένα** (structured data) είναι δεδομένα με προκαθορισμένο μοντέλο δεδομένων που καθιστά εύκολη την ανάλυσή τους. Τα δεδομένα που περιέχονται σε σχεσιακές βάσεις δεδομένων και υπολογιστικά φύλλα είναι δομημένα. Μπορούν να αποθηκευτούν σε βάση δεδομένων SQL σε πίνακα με γραμμές και στήλες. Περιλαμβάνουν ένα σχεσιακό κλειδί, το οποίο έχει αντιστοιχιστεί σε προκαθορισμένα πεδία. (Κάθε εγγραφή μπορεί να έχει ένα πεδίο με το αναγνωριστικό εγγραφής (RID – UI unique identifier) το οποίο μπορεί να χρησιμοποιηθεί για την μοναδική αναγνώριση μιας εγγραφής σε πολλαπλές εκδόσεις ενός συνόλου δεδομένων. Λόγω της αυστηρής οργάνωσης και της εύκολης προσβασιμότητας, τα δομημένα δεδομένα είναι χρήσιμα και αποτελεσματικά στην επεξεργασία μεγάλου όγκου πληροφοριών. Ωστόσο, τα δομημένα δεδομένα αντιπροσωπεύουν ένα μικρό ποσοστό των συνολικών διαθέσιμων δεδομένων (5 έως 10%).



Πηγή: https://www.hindawi.com/journals/mpe/2018/8058670/

Σχήμα 1-3: Διάκριση των Μεγάλων Δεδομένων σε Δομημένα, Ημιδομημένα και Μη Δομημένα

Τα **ημιδομημένα δεδομένα** (semi-structured data), είναι πληροφορίες που δεν διαθέτουν τη δομή που απαιτείται για να ανήκουν σε μια σχεσιακή βάση δεδομένων αλλά έχουν ορισμένες οργανωτικές ιδιότητες που διευκολύνουν την ανάλυσή τους. Μπορεί να έχουν σχετικές πληροφορίες, όπως ετικέτες μεταδεδομένων (metadata tagging), που επιτρέπουν την προσπέλαση των στοιχείων που περιέχονται. Παραδείγματα ημιδομημένων δεδομένων είναι τα αρχεία: XML, JSON, csv, tsv (κειμένου οριοθετημένα με στηλοθέτη - tab), καθώς και οι μη σχεσιακές βάσεις δεδομένων (NoSQL). Τα ημιδομημένα δεδομένα αποτελούν και αυτά 5 έως 10% των συνολικών διαθέσιμων δεδομένων .

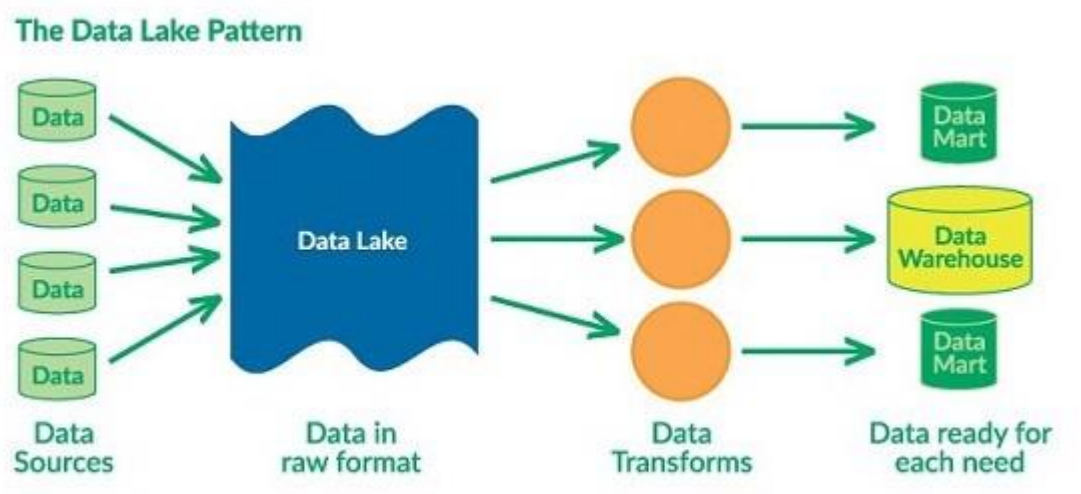
Τα **μη δομημένα δεδομένα** (non-structured data), αποτελούν περίπου το 80% των συνολικών διαθέσιμων δεδομένων στον κόσμο, δεν έχουν προκαθορισμένο σχήμα (predefined schema) - μοντέλο δεδομένων ή είναι οργανωμένα με οποιονδήποτε τρόπο. Είναι συνήθως μηνύματα ηλεκτρονικού ταχυδρομείου, έγγραφα επεξεργασίας κειμένου, υπολογιστικά φύλλα, βίντεο, φωτογραφίες, αρχεία ήχου, παρουσιάσεις, ιστοσελίδες, αρχεία καταγραφής (log files), δεδομένα clickstream, δεδομένα αισθητήρων από συσκευές, μηχανές και οχήματα, κ.λπ.

Μέθοδοι συλλογής δεδομένων για αναλυτική

Λίμνες Δεδομένων, ΛΔ / Αποθήκες Δεδομένων, ΑΔ

Οι **λίμνες δεδομένων** (Data Lakes - DL) και οι **αποθήκες δεδομένων** (Data WareHouses - DW) χρησιμοποιούνται ευρέως για την αποθήκευση μεγάλων δεδομένων, αλλά διαφέρουν αισθητά μεταξύ τους. Μια λίμνη δεδομένων είναι μια κεντρική τοποθεσία για την αποθήκευση μεγάλου όγκου δεδομένων ανεξάρτητα από την πηγή τους και τη μορφή τους. Μια αποθήκη

δεδομένων είναι ένα αποθετήριο (repository) για δομημένα, φιλτραρισμένα δεδομένα που έχουν ήδη υποβληθεί σε επεξεργασία για συγκεκριμένο σκοπό.



Πηγή: https://medium.com/datadriveninvestor/data-lakes-market-2018-2025-top-key-players-like-microsoft-informatica-teradata-capgemini-b6f6a86e2fc8

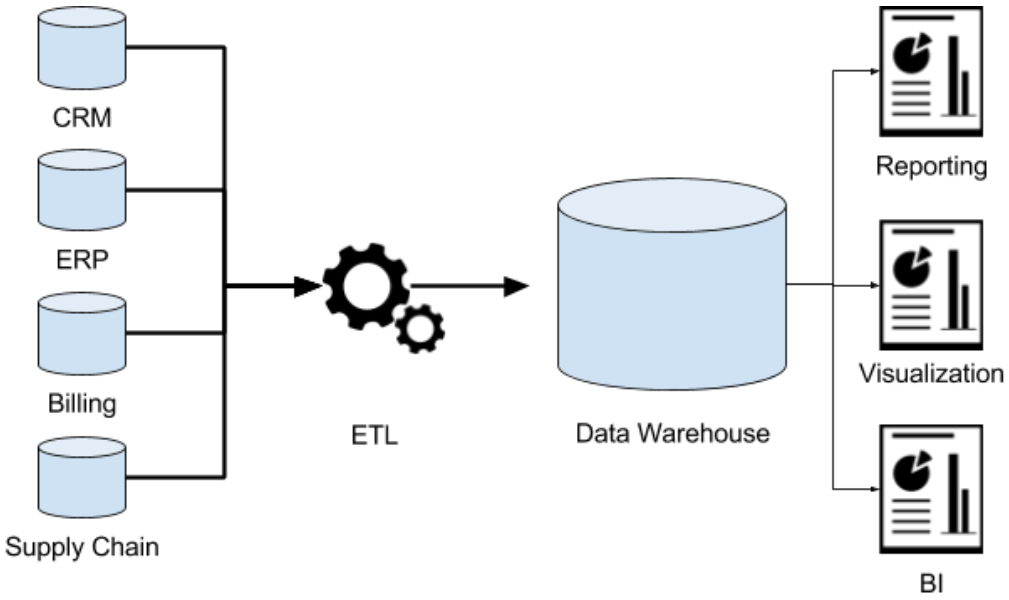
Σχήμα 1-4: Η Λίμνη Δεδομένων

Η λίμνη δεδομένων είναι μια νεότερη τεχνολογία αποθήκευσης δεδομένων με εξαιρετικές δυνατότητες κλιμάκωσης που λειτουργεί ως ένα κεντρικό αποθετήριο αποθήκευσης τόσο ακατέργαστων δεδομένων (raw data), όπου δεν έχει προσδιοριστεί ο τρόπος με τον οποίο θα αξιοποιηθούν, όσο και επεξεργασμένων δεδομένων. Η λίμνη δεδομένων υποστηρίζει την πρόσληψη (ingestion) τεράστιων ποσοτήτων δεδομένων από πολλαπλές πηγές δεδομένων.

Η λίμνη δεδομένων έχει επίπεδη (flat) αρχιτεκτονική. Κάθε στοιχείο δεδομένων σε μια Λίμνη Δεδομένων επισημαίνεται με ένα μοναδικό αναγνωριστικό και ένα σύνολο πληροφοριών μεταδεδομένων. Η δυνατότητα υλοποίησης διαφόρων τύπων αναλυτικής, σε τεράστιες ποσότητες δεδομένων συνεισφέρει στην αύξηση της αναλυτικής απόδοσης και την εγγενή ενσωμάτωση.

Λόγω της αυξανόμενης ποικιλίας και όγκου των μεγάλων δεδομένων, οι λίμνες δεδομένων αποτελούν μια ισχυρή αρχιτεκτονική προσέγγιση που η παγκόσμια αγορά της εκτιμήθηκε σε 3,24 δισεκατομμύρια δολάρια ΗΠΑ το 2017 και αναμένεται να φθάσει το ποσό των 14,01 δισεκατομμυρίων δολαρίων ΗΠΑ το 2026 με σύνθετο ετήσιο ρυθμό ανάπτυξης (compound annual growth rate - CAGR) + 27,4%, κατά την περίοδο πρόβλεψης 2018-2026 (Tejshri, A, 2019,[15]).

Μια **αποθήκη δεδομένων** είναι ένα αποθετήριο που συγκεντρώνει δομημένα δεδομένα από διάφορες πηγές για σκοπούς σύγκρισης και ανάλυσης στον τομέα της επιχειρηματικής ευφυΐας. Η απαίτηση για δομημένα δεδομένα έχει κόστος καθώς θα πρέπει να σχεδιάσει ο επίδοξος χρήστης ένα σχήμα για να φορτώσει τα δεδομένα του. Όπως αναφέρθηκε προηγουμένως η ποικιλία (πηγές και μορφές δεδομένων) των μεγάλων δεδομένων αυξάνει με την πάροδο του χρόνου και καθώς θα προσθέτει ο χρήστης νέες πηγές και μορφές δεδομένων, θα πρέπει να κάνει περισσότερη μοντελοποίηση και αναδιαμόρφωση που θα μπορούσε επίσης να περιορίσει την ικανότητά του να επαναχρησιμοποιήσει τα ίδια δεδομένα για νέες περιπτώσεις χρήσης (use cases) στο μέλλον.



Πηγή: https://www.dremio.com/what-is-a-data-warehouse

Σχήμα 1-5: Αποθήκη Δεδομένων

Σχήμα κατά την ανάγνωση εναντίον σχήματος κατά την εγγραφή

Μια άλλη διαφορά μεταξύ μιας λίμνης δεδομένων και μιας αποθήκης δεδομένων είναι ο τρόπος ανάγνωσης των δεδομένων. Μια λίμνη δεδομένων φιλοξενεί κυρίως ακατέργαστα δεδομένα, χωρίς κάποιο προκαθορισμένο σχήμα. Το σχήμα δημιουργείται και εφαρμόζεται μόνο όταν τα δεδομένα διαβάζονται από τη λίμνη δεδομένων. Αυτή η διαδικασία επιβολής σχήματος ονομάζεται "σχήμα κατά την ανάγνωση" (schema on read). Το "σχήμα κατά την ανάγνωση" επιτρέπει τον χρήστη να παρουσιάσει τα δεδομένα του σε ένα σχήμα που προσαρμόζεται καλύτερα στα ερωτήματα που υποβάλλονται.

Το "σχήμα κατά την ανάγνωση" παρέχει μεγαλύτερη ευελιξία και επεκτασιμότητα διότι [26]:

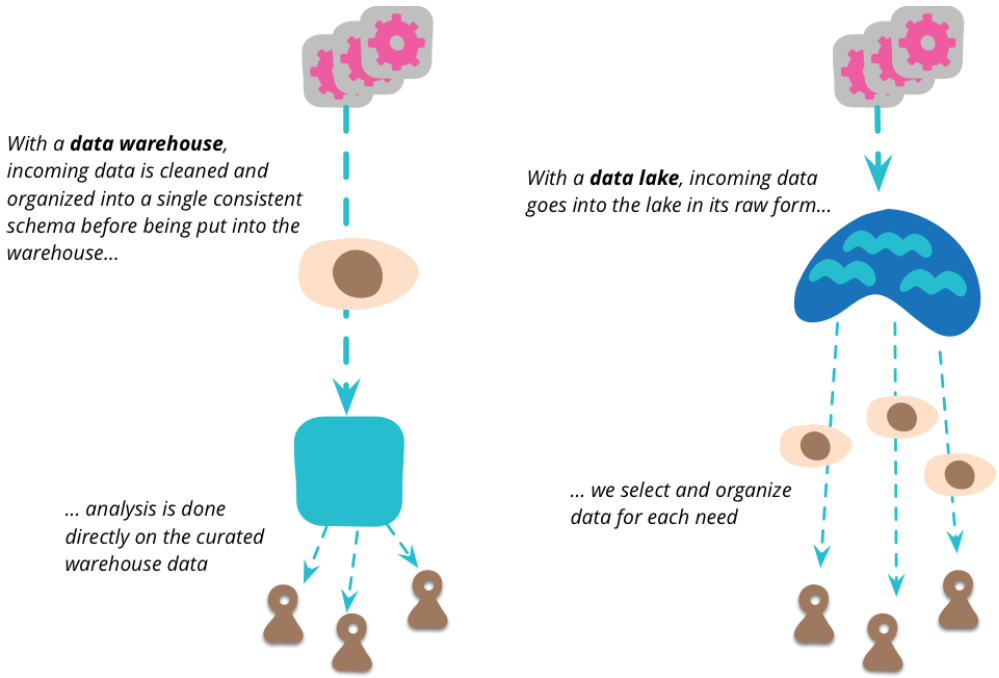
• Διαφορετικές εφαρμογές μπορούν να εφαρμόσουν διαφορετικά σχήματα παράλληλα στα δεδομένα.

• Η τροποποίηση του σχήματος μιας εφαρμογής δεν επηρεάζει τα ανεπεξέργαστα δεδομένα ή τις άλλες εφαρμογές.

• Κατά την επιβολή ενός σχήματος από μια εφαρμογή, δεν χάνονται πληροφορίες εφόσον διατηρηθούν τα ανεπεξέργαστα δεδομένα.

• Το προκύπτον κόστος πειραματισμού με τα δεδομένα είναι χαμηλό.

Αντίθετα το σχήμα στην αποθήκη δεδομένων καθορίζεται εκ των προτέρων. Ένα σχήμα αποτελείται από πολλούς πίνακες και τα πεδία τους, με τους τύπους δεδομένων, τους περιορισμούς και τις σχέσεις τους. Τα περισσότερα από τα δεδομένα που υπάρχουν είναι μη δομημένα και θα πρέπει να δομηθούν με βάση αυτό το σχήμα καθώς εισέρχονται στην Αποθήκη Δεδομένων. Η τεχνική αυτή επιβολής του σχήματος ονομάζεται "σχήμα κατά την εγγραφή" (schema on write). Όταν διαβάζει ο χρήστης δεδομένα από την αποθήκη δεδομένων, είναι στη μορφή που έχει προκαθορισθεί από αυτό το σχήμα. Καθώς προκύπτουν συνεχώς νέες μορφές και πηγές δεδομένων είναι αδύνατη η δημιουργία ενός καθολικού σχήματος που να μπορεί να τις προδιαγράψει εκ των προτέρων. Οι χρήσεις των δεδομένων ενδέχεται να αλλάξουν με την πάροδο του χρόνου. Αυτό σημαίνει ότι το ενεργό σχήμα θα πρέπει να τροποποιηθεί πράγμα διόλου εύκολο αφού μπορεί να μην είναι σε θέση ο χρήστης να ανακτήσει τις πληροφορίες που έχουν χαθεί από τα ανεπεξέργαστα δεδομένα που έχουν ήδη δομηθεί με το προγενέστερο σχήμα.



Πηγή: https://martinfowler.com/bliki/DataLake.html

Σχήμα 1-6: (α) Αποθήκη Δεδομένων: Στα δεδομένα επιβάλλεται ένα μοναδικό σχήμα (σχήμα κατά την εγγραφή) πριν την εγγραφή τους στην αποθήκη δεδομένων. (β) Λίμνη Δεδομένων: Τα δεδομένα αποθηκεύονται ανεπεξέργαστα στη λίμνη δεδομένων και εφαρμόζονται διαφορετικά σχήματα κατά την ανάγνωση που να ανταποκρίνονται καλύτερα στα ερωτήματα που υποβάλλονται.

Για παράδειγμα, ας πούμε ότι μια λίμνη δεδομένων έχει μια συλλογή πολλών χιλιάδων αρχείων JSON. Αυτά τα αρχεία μπορεί να μην ακολουθούν κάποιο συγκεκριμένο σχήμα, μπορεί να είναι πολλά επίπεδα βαθιά, αλλά μπορεί επίσης να έχουν μερικά κοινά πεδία. Ένας επιστήμονας δεδομένων μπορεί να εξάγει μόνο αυτά τα κοινά πεδία από κάθε αρχείο και να συμπληρώσει έναν πίνακα. Σε αυτή την περίπτωση, ο επιστήμονας δεδομένων γράφει ένα ερώτημα για να αναλύσει τα αρχεία JSON, να εξάγει τις τιμές των πεδίων και να συμπληρώσει έναν πίνακα. Με άλλα λόγια, εφαρμόζεται ένα σχήμα στα αρχεία προέλευσης όταν διαβάζονται τα δεδομένα.

ELT εναντίον ETL

Ο μετασχηματισμός δεδομένων δεν είναι τόσο πολύ μια προτεραιότητα στις ΛΔ (λίμνες δεδομένων) όσο η φόρτωση δεδομένων. Συνήθως, οι σωληνώσεις δεδομένων (data pipelines) στις λίμνες δεδομένων εξάγουν δεδομένα από τα πηγαία συστήματα και τα φορτώνουν στη ΛΔ στόχο όσο το δυνατόν γρηγορότερα. Επομένως τα εργαλεία ΕΦΜ, Εξαγωγής, Φόρτωσης-Μετασχηματισμού (ELT, Extraction-Loading-Transformation) είναι ιδανικά για αυτό. Πολλά εργαλεία ΕΦΜ μπορούν να συνδεθούν με τα συστήματα αποθήκευσης ΛΔ εγγενώς. Το τμήμα μετασχηματισμού γενικά διαφέρει μεταξύ των καταναλωτών δεδομένων, έτσι μπορεί να υπάρχουν πολλά διαφορετικά είδη περιπτώσεων χρήσης μετασχηματισμού. Σε γενικές γραμμές, ο μετασχηματισμός γίνεται με εφαρμογές τελικού χρήστη που συνδέονται στη ΛΔ. Χρησιμοποιώντας μόνο τα «Ε» και «Φ» του ΕΦΜ σημαίνει επίσης ότι οι σωληνώσεις δεδομένων στις ΛΔ είναι απλές και φθηνές.

Οι αποθήκες Δεδομένων, ΑΔ, συνήθως εξαρτώνται από διαδικασίες ΕΜΦ, Εξαγωγής – Μετασχηματισμού - Φόρτωσης (ETL, Extract-Transform-Load), όπου ένα ή περισσότερα πακέτα ΕΜΦ τρέχει μέσα σε έναν ή περισσότερους αφιερωμένους υπολογιστικούς πόρους. Αυτό το υπολογιστικό στρώμα εξάγει δεδομένα, τα μετασχηματίζει και στη συνέχεια τα φορτώνει στην ΑΔ. Οι διαδικασίες μετασχηματισμού για την αποθήκευση των δεδομένων είναι σαφώς καθορισμένες, αποτελούν αυστηρούς κανόνες των επιχειρήσεων, και είναι επαναλαμβανόμενες.

Μια άλλη διαφορά μεταξύ της ΕΦΜ της ΛΔ και της ΕΜΦ της ΑΔ είναι ο τρόπος που έχουν προγραμματιστεί. Στις ΛΔ μπορεί (ή δεν μπορεί) να υπάρχουν προγραμματισμένες διαδικασίες φόρτωσης και μετασχηματισμού. Μερικοί μετασχηματισμοί μπορεί να είναι εφάπαξ ή ad-hoc. Οι μετασχηματισμοί στις ΑΔ είναι σχεδόν πάντα προγραμματισμένοι.

Αναλυτική Δεδομένων

Η αναλυτική δεδομένων (data analytics) είναι ένα επιστημονικό πεδίο που χρησιμοποιεί μαθηματικά, στατιστική, τεχνικές πρόβλεψης και μηχανική μάθηση με σκοπό την ανακάλυψη, ερμηνεία και επικοινωνία σημαντικών προτύπων και συσχετισμών στα δεδομένα και τη διαδικασία εφαρμογής αυτών των προτύπων για την αποτελεσματική λήψη αποφάσεων.

Η ανάλυση δεδομένων μπορεί να βοηθήσει τον ενδιαφερόμενο να απαντήσει στους ακόλουθους τύπους ερωτήσεων: Τι συνέβη; Πώς ή γιατί συνέβη αυτό; Τι συμβαίνει τώρα; Τι θα συμβεί στη συνέχεια; Τι πρέπει να κάνουμε;.

Υπάρχουν τέσσερις κύριοι τύποι αναλυτικής που χρησιμοποιούνται σήμερα.

- Η **περιγραφική αναλυτική** (descriptive analytics) εστιάζει στην συνάθροιση και την εξόρυξη δεδομένων ώστε να παρέχει πληροφορίες για το παρελθόν και να απαντήσει στο ερώτημα: "Τι έχει συμβεί;" . Η κατανόηση των αλλαγών που έχουν συμβεί σε μια επιχείρηση είναι πρωταρχικής σημασίας για την λήψη τελέσφορων αποφάσεων στις οποίες θα βασιστεί η επιχειρησιακή στρατηγική της. Στοιχεία όπως καταμετρήσεις, αθροίσματα, μέσοι όροι, κατανομές, κ.λπ. συνοψίζονται και συγκρίνονται σε χρονικές περιόδους και οι πληροφορίες αυτές μπορεί να πάρουν τη μορφή απεικονίσεων δεδομένων όπως γραφήματα, διαγράμματα, αναφορές και πίνακες ελέγχου.

  Η περιγραφική αναλυτική για παράδειγμα μπορεί να χρησιμοποιηθεί για τη συγκέντρωση συναλλακτικών πληροφοριών, όπως το πόσα κινητά πωλήθηκαν. Ποια μοντέλα κινητών πωλήθηκαν (υπό μορφή κατάταξης ή ποσοστού) και κατά πόσο πωλήθηκαν περισσότερα ή λιγότερα κινητά από την προηγούμενη ημέρα. Η περιγραφική αναλυτική είναι η κύρια προσέγγιση για την κατανόηση των δεδομένων και των υφιστάμενων προτύπων.

  Η αγορά λογισμικού περιγραφικής αναλυτικής θα φτάσει τα 1,88 δισεκατομμύρια δολάρια μέχρι το 2022 με 20,6% σύνθετο ετήσιο ρυθμό ανάπτυξης από το 2017 (Woodward A., et al., 2019[17]).
- Η **διαγνωστική αναλυτική** (diagnostic analytics) αναζητά τη βασική αιτία που συνέβη ένα συμβάν χρησιμοποιώντας τεχνικές όπως ανίχνευση δεδομένων, ανακάλυψη δεδομένων, εξόρυξη δεδομένων και συσχετική στατιστική. Απαντά στο ερώτημα: "Γιατί συνέβη αυτό;"

Η προηγμένη αναλυτική (ή προηγμένες τεχνικές ανάλυσης - advanced analytics) ασχολείται με την αυτόματη ανακάλυψη και επικοινωνία ουσιαστικών πρότυπων τόσο σε δομημένα όσο

και σε μη δομημένα δεδομένα αλλά και την πρόβλεψη μελλοντικών γεγονότων. Η Προηγμένη Αναλυτική επεκτείνει τους ορίζοντες της Παραδοσιακής Αναλυτικής (traditional analytics). Με την παραδοσιακή αναλυτική οι αναλυτές απαντάνε στο ερώτημα "Τι συνέβη;" Αντίθετα με την προηγμένη αναλυτική εκμαιεύουν το "Τι θα συμβεί;" αλλά και το "Τι πρέπει να κάνουμε;".

Η χρήση Μεθοδολογιών και τεχνολογιών από την στατιστική και την επιστήμη των υπολογιστών έπαιξε σημαντικό ρόλο στην ανάπτυξη και καθιέρωση του κλάδου της προηγμένης αναλυτικής. Οι κύριες συμβολές προέρχονται από την μηχανική μάθηση και την εξόρυξη δεδομένων.

Η **μηχανική μάθηση** ασχολείται με τη μεγάλης κλίμακας και αυτοματοποιημένη δημιουργία στατιστικών μοντέλων σε μεγάλα σύνολα δεδομένων, χωρίς την ανάγκη για δοκιμή υποθέσεων (hypothesis testing).

Η **Εξόρυξη Δεδομένων** έχει εμπλουτίσει την μηχανική μάθηση καλύπτοντας επίσης τα απαραίτητα βήματα για την ενσωμάτωση δεδομένων και την προεπεξεργασία τους (ανίχνευση ανωμαλιών ή ακραίων τιμών, προτύπων, αλληλεξαρτήσεων ανάμεσα σε μεταβλητές, κ.λπ.), προκειμένου να δημιουργηθούν καλύτερα μοντέλα. Η μηχανική μάθηση επικεντρώνεται στην κατασκευή μοντέλων. Σήμερα, ο όρος εξόρυξη δεδομένων χρησιμοποιείται λιγότερο συχνά. Ανάμεσα σε αναλυτές, ο όρος αυτός έχει αντικατασταθεί κυρίως από τούς όρους προβλεπτική αναλυτική, περιγραφική μοντελοποίηση ή άλλων συγγενικών πεδίων όπως αναλυτική κείμενου ή αναλυτική πολυμέσων.

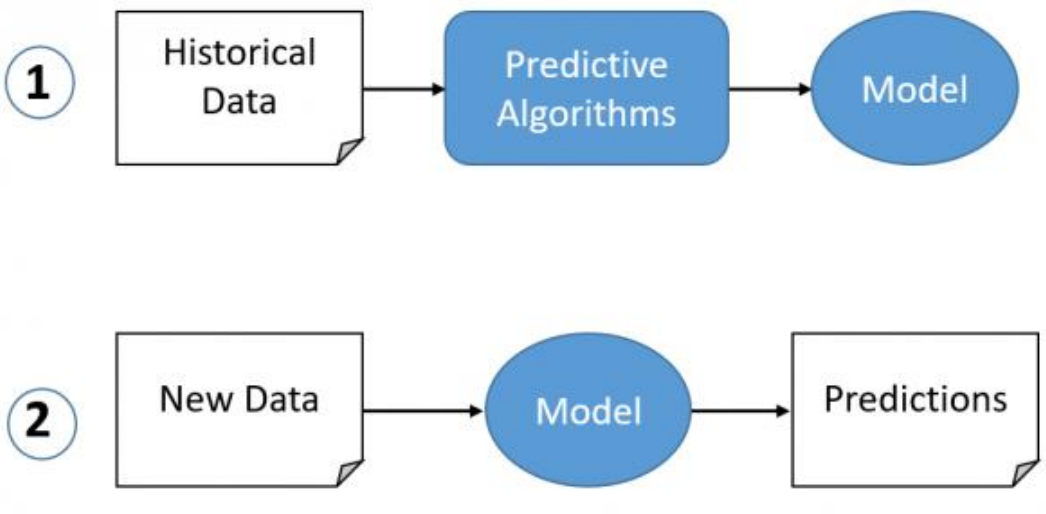
Τα Μεγάλα Δεδομένα αποτελούν περιουσιακό στοιχείο και όπως κάθε άλλο εταιρικό περιουσιακό στοιχείο, έχουν οικονομική αξία. Η εξόρυξη δεδομένων είναι το μέσο για την εξαγωγή “αξίας” ή “πολύτιμης” πληροφορίας από τα δεδομένα.

- Η **προβλεπτική αναλυτική** (predictive analytics), είναι ένας κλάδος του επιστημονικού πεδίου της στατιστικής advanced Analytics που χρησιμοποιεί στατιστικά μοντέλα και μηχανική μάθηση για να κατανοηθεί το μέλλον και να απαντηθεί το ερώτημα: "Τι θα μπορούσε να συμβεί;"

  Η προβλεπτική αναλυτική χρησιμοποιεί στατιστικά μοντέλα και μηχανική μάθηση για να προβλέψει τι θα μπορούσε να συμβεί στο μέλλον, με βάση προηγούμενα πρότυπα. Συνήθως, αυτό γίνεται τροφοδοτώντας ιστορικά δεδομένα (καθαρισμένα και / ή μετασχηματισμένα) σε ένα προβλεπτικό μοντέλο μηχανικής μάθησης που εξετάζει τις βασικές τάσεις και πρότυπα στα δεδομένα. Το μοντέλο εφαρμόζεται στη συνέχεια στα

τρέχοντα δεδομένα για να προβλέψει ο ενδιαφερόμενος τι θα συμβεί στη συνέχεια (βλέπε Σχήμα 1.7).

Εκτός από δομημένα δεδομένα μπορεί ο ενδιαφερόμενος να έχει και μη δομημένα δεδομένα τα οποία όμως απαιτούν προεπεξεργασία πριν την χρήση τους από το προβλεπτικό μοντέλο. Η επιλογή των σχετικών δεδομένων, ο καθαρισμός τους και οι επακόλουθοι μετασχηματισμοί μπορεί να είναι μακριά και επίπονη διαδικασία.



Πηγή: https://www.logianalytics.com/predictive-analytics/what-is-predictive-analytics

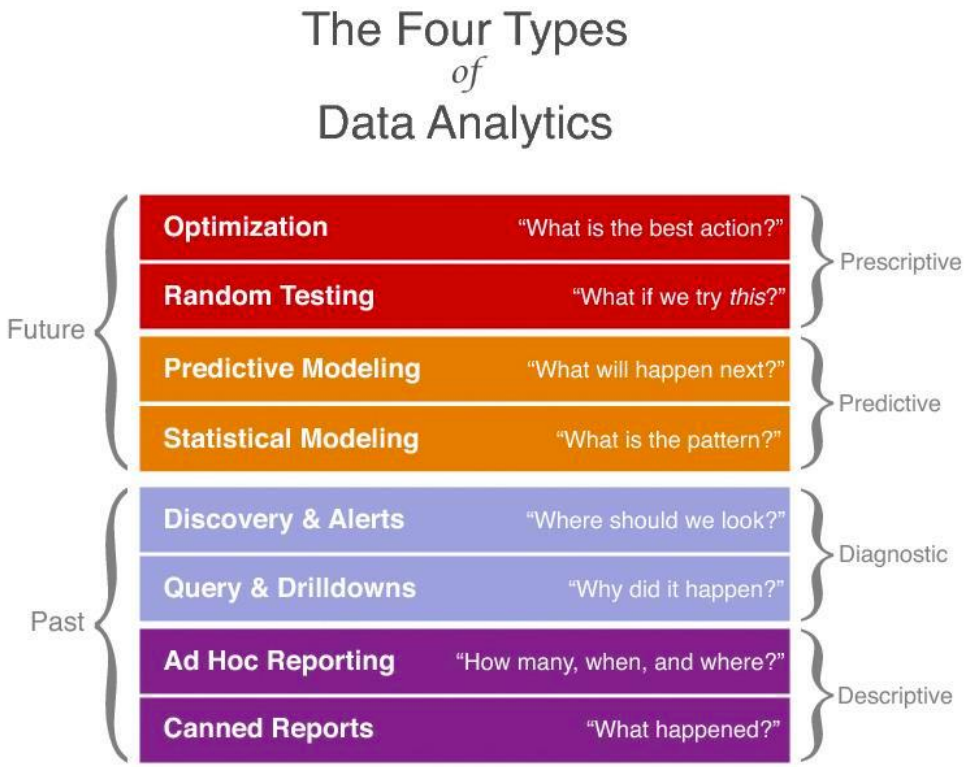
Σχήμα 1-7: Προβλεπτική αναλυτική

Η προβλεπτική αναλυτική αφορά την εποπτευόμενη μάθηση. Αυτό σημαίνει ότι θέλει ο ενδιαφερόμενος να μάθει από προηγούμενα δεδομένα για να προβλέψει τις μελλοντικές τάσεις. Η προβλεπτική ανάλυση βασίζεται κυρίως στην στατιστική πιθανότητα. Μπορεί να ταξινομηθεί η προβλεπτική ανάλυση σε δύο βασικές προσεγγίσεις:

- Παλινδρόμηση
- Ταξινόμηση

Σε προσεγγίσεις παλινδρόμησης, γίνεται πρόβλεψη μιας αξίας. Για παράδειγμα, ενδέχεται να προβλέψει κανείς τις πωλήσεις κινητών για τους επόμενους δυο μήνες.

Ενώ σε προσεγγίσεις ταξινόμησης γίνεται πρόβλεψη της κατηγορίας / κλάσης στην οποία θα περιλαμβάνονται νέα δεδομένα. Για παράδειγμα, στο πρόβλημα της απώλειας πελατών (customer churn) , μπορεί ο ενδιαφερόμενος να προβλέψει για ένα νέο πελάτη αν ανήκει σε μια ομάδα που μπορεί να εγκαταλείψει ή να παραμείνει στην πελατειακή του βάση.



Πηγή: https://datafloq.com/read/the-four-types-of-data-analytics/3903

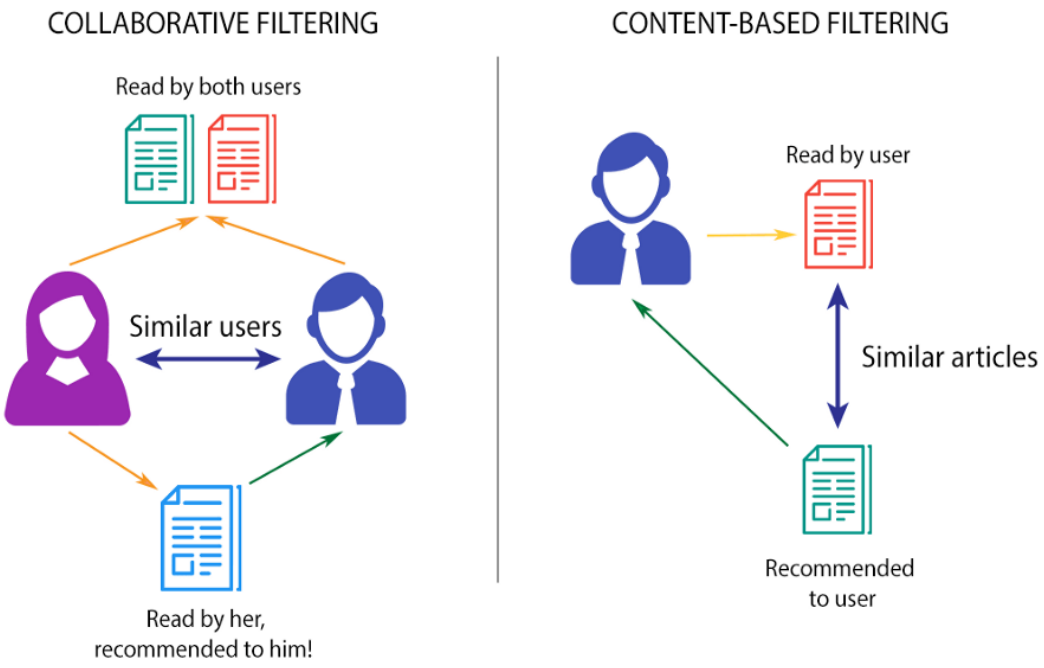
Σχήμα 1-8: Οι τέσσερις τύποι της Αναλυτικής Δεδομένων – Παρελθόν: Περιγραφική (Descriptive), και Διαγνωστική (Diagnostic) αναλυτική. Μέλλον: προβλεπτική (Predictive) και καθοδηγητική (Prescriptive) αναλυτική

- Η **καθοδηγητική αναλυτική** (prescriptive analytics), που αποτελεί τον έτερο κλάδο της προηγμένης αναλυτικής, χρησιμοποιεί αλγορίθμους βελτιστοποίησης και σύστασης για την παροχή συμβουλών σχετικά με τις πιθανές συνέπειες και απαντούν στο ερώτημα: "Τι πρέπει να κάνουμε;". Αυτός ο τύπος αναλυτικής απαιτεί εξωγενή πληροφόρηση εκτός από ιστορικά δεδομένα εξαιτίας της φύσης των αλγορίθμων που χρησιμοποιεί.

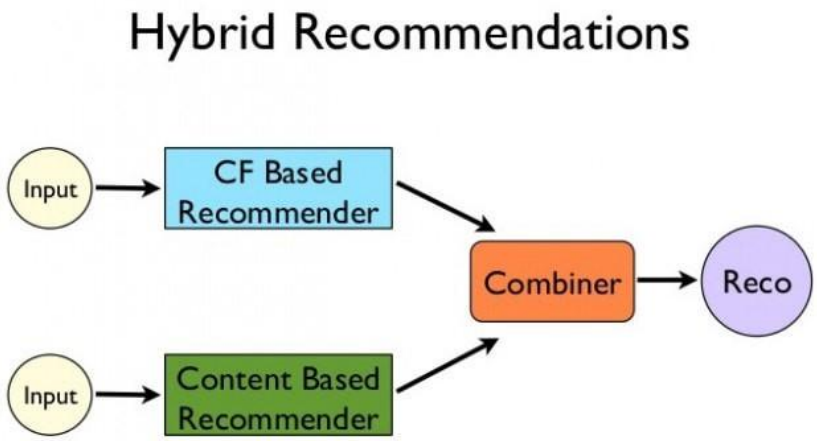
  Αφορά κυρίως συστήματα σύστασης (Recommendation or Recommender systems), τα οποία γίνονται όλο και πιο δημοφιλή τα τελευταία χρόνια και χρησιμοποιούνται σε διάφορους τομείς, όπως ερευνητικά άρθρα, ερωτήματα αναζήτησης, κοινωνικές ετικέτες και προϊόντα γενικά. Υπάρχουν βασικά τρεις σημαντικοί τύποι μηχανισμών συστάσεων:

  - Συνεργατικό φιλτράρισμα (Collaborative Filtering – CF). Το συνεργατικό φιλτράρισμα χρησιμοποιείται ευρέως στα μέσα κοινωνικής δικτύωσης, το λιανικό εμπόριο και τις υπηρεσίες ροής. Βασίζεται στην ιδέα ότι άνθρωποι με κοινό ενδιαφέρον για ορισμένα πράγματα θα έχουν πιθανώς παρόμοια προτίμηση και σε άλλα πράγματα.

- Φιλτράρισμα βάσει περιεχομένου (Content-Based Filtering – CBF) . Είναι η εύρεση της ομοιότητας μεταξύ αντικειμένων. Η σύσταση βάσει περιεχομένου αναλύει τις περιγραφές αντικειμένων και συνιστά σ΄ έναν χρήστη ένα αντικείμενο που είναι παρόμοιο με εκείνα που του έχουν αρέσει στο παρελθόν. Καθώς ο χρήστης παρέχει περισσότερες πληροφορίες ή λαμβάνει μέτρα στις συστάσεις, η μηχανή σύστασης γίνεται όλο και πιο ακριβής. Συστήματα σύστασης βάσει περιεχομένου είναι για παράδειγμα τα συστήματα που προτείνουν μια νέα ταινία ή ένα νέο τραγούδι σε ένα άτομο, με βάσει την ομοιότητά τους με άλλες ταινίες ή τραγούδια που έχει ήδη επιλέξει ο χρήστης κατά το παρελθόν.
- Υβριδικά συστήματα συστάσεων (Hybrid Recommendation Systems). Είναι τα συστήματα που συνδυάζουν συνεργατικό φιλτράρισμα και φιλτράρισμα βάσει περιεχομένου. Τα συστήματα αυτά έχει αποδειχθεί ότι είναι συνήθως πολύ πιο αποτελεσματικά από τις ανωτέρω προσεγγίσεις.



Σχήμα 1-9: Συστήματα Σύστασης (Recommendation or Recommender systems), (α) συνεργατικό φιλτράρισμα (collaborative filtering), (β) φιλτράρισμα βάσει περιεχομένου (content based filtering)



Πηγή: https://towardsdatascience.com/what-are-product-recommendation-engines-and-the-various-versions-of-them-9dcab4ee26d5

Σχήμα 1-10: Συστήματα Σύστασης, Υβριδικά Συστήματα Συστάσεων (Hybrid Recommendation Systems)

Κεφάλαιο 2
Εισαγωγή στο Apache Hadoop

Οι προκλήσεις που εισάγουν τα μεγάλα δεδομένα

Τα μεγάλα δεδομένα δημιουργούν μεγάλες ευκαιρίες. Για να μετουσιωθούν όμως οι ευκαιρίες αυτές σε κέρδη θα πρέπει μια επιχείρηση να μπορεί να επεξεργαστεί και να αναλύσει τα δεδομένα αυτά ούτως ώστε να εξάγει πολύτιμη γνώση που θα της προσδώσει ένα ανταγωνιστικό πλεονέκτημα έναντι του ανταγωνισμού. Η διαχείριση των μεγάλων δεδομένων εισάγει τρεις μεγάλες προκλήσεις.

- Η πρώτη πρόκληση, αφορά το πρόβλημα της αποθήκευσης των μεγάλων δεδομένων. Είναι προφανές ότι δεν είναι δυνατή η αποθήκευση τεράστιου όγκου δεδομένων σε ένα παραδοσιακό σύστημα.
- Η δεύτερη πρόκληση αφορά το πρόβλημα της αποθήκευσης ετερογενών δεδομένων. Όπως έχει ήδη αναφερθεί υπάρχουν τρεις μορφές μεγάλων δεδομένων, τα μη δομημένα, τα ημι-δομημένα και τα δομημένα δεδομένα. Θα πρέπει λοιπόν το αποθηκευτικό σύστημα μιας επιχείρησης / οργανισμού να υποστηρίζει την αποθήκευση διαφορετικών τύπων δεδομένων που δημιουργούνται από διάφορες πηγές.
- Η τρίτη πρόκληση, είναι η ταχύτητα επεξεργασίας. Ο απαιτούμενος χρόνος για τη επεξεργασία αυτού του τεράστιου όγκου δεδομένων είναι αρκετά υψηλός.

Μια λύση στις παραπάνω προκλήσεις αποτελεί το Hadoop, όμως πριν εξεταστεί πως τις επιλύει θα γίνει μια παρουσίαση αναφορικά με το τι είναι το Hadoop και ποια είναι η αρχιτεκτονική του.

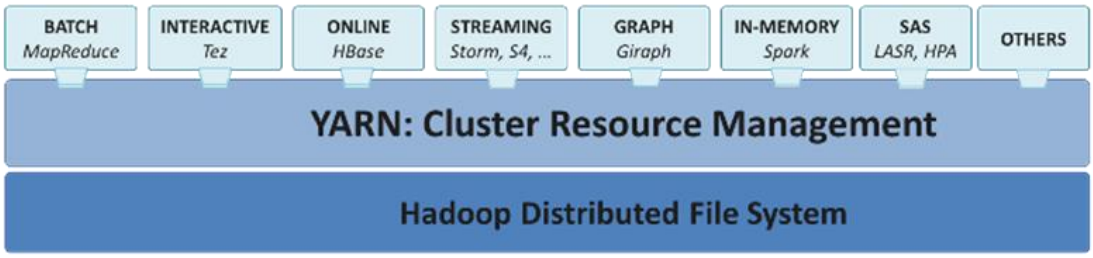
Hadoop 2.x

Το Hadoop είναι ένα πλαίσιο λογισμικού ανοικτού κώδικα γραμμένο σε Java που χρησιμοποιείται για την κατανεμημένη αποθήκευση και επεξεργασία μεγάλων δεδομένων σε μεγάλες συστάδες κοινών κυρίως υπολογιστών. Το Hadoop παρέχει υψηλή υπολογιστική ισχύ, μεγάλη αποθηκευτική ικανότητα αλλά και ταχύτητα ενώ ταυτόχρονα είναι κλιμακώσιμο (scalable), ευέλικτο (flexible), αξιόπιστο και ανεκτικό σε σφάλματα (fault tolerant) όπως

φαίνεται στο Σχήμα 2.1. Η έκδοση 1.0 του Hadoop επέτρεπε την επεξεργασία δεδομένων χρησιμοποιώντας την υπολογιστική μηχανή MapReduce, η οποία χρησιμοποιούσε ένα προγραμματιστικό μοντέλο για επεξεργασία δεδομένων σε δέσμες – κατά παρτίδες - (batch processing) ενώ η έκδοση 2.x επιτρέπει την χρήση μιας πλειάδας υπολογιστικών μηχανών πέραν της ενσωματωμένης υπολογιστικής μηχανής MapReduce. Στη συνέχεια θα επικεντρωθούμε στην αρχιτεκτονική και τα χαρακτηριστικά του Hadoop 2.x.

Τα τρία βασικά συστατικά του hadoop είναι :

1. Το HDFS (*Hadoop Distributed File System*) - είναι ένα εξαιρετικά ανεκτικό σε σφάλματα κατανεμημένο σύστημα αρχείων που χρησιμοποιείται από μια μεγάλη ποικιλία από ταυτόχρονες εφαρμογές πρόσβασης δεδομένων (Σχήμα 2.1), που συντονίζονται από το YARN. Το HDFS αντί να αποθηκεύει ένα πλήρες αρχείο, το χωρίζει σε μικρά μπλοκ δεδομένων με προκαθορισμένο μέγεθος (συνήθως 128 MB), δημιουργεί πολλαπλά αντίγραφα των μπλοκ δεδομένων και τα διανέμει στους υπολογιστικούς κόμβους μιας συστάδας. Αυτή η κατανομή εξασφαλίζει ότι τα δεδομένα παραμένουν διαθέσιμα, παρά τις αναπόφευκτες αστοχίες κάποιων από τους κόμβους που τα φιλοξενούν.
2. Το YARN (Yet Another Resource Negotiator) - είναι ένα σύστημα για τον συντονισμό και την διαχείριση των πόρων συστάδας, το οποίο ενεργώντας ως διεπαφή με τους πόρους συστάδας επιτρέπει κατανεμημένες εφαρμογές MapReduce αλλά και από άλλες πολλές μηχανές επεξεργασίας δεδομένων (για διαδραστικά ερωτήματα SQL, ροή σε πραγματικό χρόνο, επεξεργασίας κατά παρτίδες, κ.λπ.) να επεξεργαστούν δεδομένα αποθηκευμένα σε μια συστάδα Hadoop.
3. Το MapReduce - είναι η εγγενής μηχανή του Hadoop για επεξεργασία σε δέσμες (batches).



Πηγή: https://data-flair.training/blogs/hadoop-yarn-tutorial/

Σχήμα 2-1:Η αρχιτεκτονική του Hadoop 2.x

HDFS

Η Αρχιτεκτονική του Apache Hadoop HDFS ακολουθεί μια τοπολογία Master / Slave όπου μία συστάδα αποτελείται από ένα μόνο NameNode (NN-Master Node or Daemon /master κόμβο ή δαίμονα) κόμβο και όλοι οι άλλοι κόμβοι είναι DataNodes (DN-Slave κόμβοι ή δαίμονες). Το HDFS μπορεί να λειτουργήσει με:

- Απενεργοποιημένη την Υψηλή Διαθεσιμότητα (non-HA-enabled HDFS cluster – HA: High Availability) (NameNode Secondary, NameNode)
- Ενεργοποιημένη Υψηλή Διαθεσιμότητα (HA-enabled HDFS cluster) (βλέπε Σχήμα 2-3) (NameNode, Standby NameNode)

Παρακάτω δίδεται μια σύντομη περιγραφή των συστατικών στοιχείων του HDFS :

- Ο **NameNode** διαχειρίζεται το χώρο ονομάτων (namespace) του συστήματος αρχείων, επιτρέποντας στους υπολογιστές-πελάτες να δουλεύουν με αρχεία και καταλόγους. Ο **Namenode** αποθηκεύει μόνο τα μεταδεδομένα για τα αρχεία (όπως ονόματα αρχείων συμπεριλαμβανομένου του μονοπατιού αποθήκευσής τους, δικαιώματα (permissions) πρόσβασης αρχείων και καταλόγων, χρόνος πρόσβασης αρχείου, χρόνος τροποποίησης αρχείων και καταλόγου, πληροφορίες για το μέγεθος του block, καθώς και τα blocks από τα οποία αποτελείται το κάθε αρχείο και την θέση τους στους DataNodes που είναι αποθηκευμένα (Block Address Table), ιδιοκτήτες αρχείου ή καταλόγου, όνομα ομάδας (groupname) αρχείου ή καταλόγου κλπ.) και παρακολουθεί τα αρχεία σε ολόκληρη τη συστάδα, ενώ τα πραγματικά δεδομένα αποθηκεύονται στον DataNode. Όταν χρησιμοποιείται, ο NameNode αποθηκεύει όλα τα μεταδεδομένα στην κύρια μνήμη, αλλά τα αποθηκεύει και σε μόνιμο φυσικό αποθηκευτικό μέσο. Συγκεκριμένα τα μεταδεδομένα αποθηκεύονται σε δύο αρχεία:
  - Στο αρχείο fsimage που είναι η αποθήκη μεταδεδομένων (metadata store) που κρατά ένα στιγμιότυπο (snapshot) του συστήματος αρχείων. Ο μορφότυπος (format) αυτού του αρχείου είναι αποδοτικός μόνο για διάβασμα αντίθετα είναι ακατάλληλος για μικρές διαδοχικές ενημερώσεις.
  - Στο αρχείο καταγραφής επεξεργασίας edits (edits log file) που κρατά όλες τις αλλαγές στο σύστημα αρχείων HDFS. Εννοιολογικά, το αρχείο καταγραφής επεξεργασίας edits είναι μια ενιαία οντότητα, αλλά αντιπροσωπεύεται από έναν αριθμό αρχείων στο δίσκο.

Ο NameNode λαμβάνει περιοδικά τις αναφορές Heartbeat[1] (heartbeat report) και block[2] (block report) από όλους τους DataNodes της συστάδας. Η αναφορά heartbeat επιβεβαιώνει τον NameNode ότι ο κόμβος είναι ενεργός (ζωντανός) ενώ παράλληλα περιλαμβάνονται πληροφορίες όπως ο συνολικός χώρος στο δίσκο, ο συνολικός χώρος που χρησιμοποιείται καθώς και ο αριθμός των μεταφερόμενων δεδομένων που βοηθούν τον NameNode στην διενέργεια εξισορρόπησης φορτίου. Εάν ο NameNode δεν λάβει αναφορά heartbeat από ένα DataNode για διάστημα μεγαλύτερο των 10 λεπτών θεωρεί τον Datanode ως νεκρό και αρχίζει την αντιγραφή των block[3] που είναι αποθηκευμένα στον νεκρό DataNode σε άλλους ενεργούς κόμβους για να διασφαλιστεί η διαθεσιμότητα. Η αναφορά block (block report) είναι μια λίστα όλων των αντιγράφων (replicas) HDFS block δεδομένων που αντιστοιχούν σε κάθε ένα από τα τοπικά αρχεία του DataNode (περιέχει το block ID, το generation stamp, και το μήκος κάθε αντιγράφου block (block replica) που φιλοξενεί). Είναι υπεύθυνος για την αποδοχή εργασίας από τους πελάτες (clients). Σκοπός του είναι να εξασφαλίσει ότι τα δεδομένα που απαιτούνται για τη λειτουργία φορτώνονται και διαχωρίζονται σε κομμάτια block δεδομένων. Ο αριθμός των αντιγράφων ενός αρχείου που επιθυμεί ο χρήστης να αποθηκευτούν στο HDFS ονομάζεται συντελεστής αναπαραγωγής (replication factor) αυτού του αρχείου. Οι πληροφορίες αυτές αποθηκεύονται από το NameNode.

- Πριν την έλευση του Hadoop 2.0 [4], ο Namenode αποτελούσε το μοναδικό σημείο αποτυχίας (single point of failure -SPOF) σε μια συστάδα HDFS. Έτσι αποτυχία του NameNode σήμαινε αποτυχία ολόκληρης της συστάδας και απαιτούνταν χειρωνακτική παρέμβαση για την επαναφορά της συστάδας Hadoop είτε με επανεκκίνηση του NameNode είτε με εκκίνηση ενός κόμβου που διατηρούσε ένα αντίγραφο ασφαλείας του NameNode γεγονός που είχε αρνητικό αντίκτυπο στο συνολικό χρόνο διακοπής της συστάδας. Η ιδιότητα της υψηλής διαθεσιμότητας (HDFS High Availability ή HA) του Hadoop 2.0 αντιμετωπίζει προβλήματα όπως απώλεια συστάδας λόγω απρόσμενου γεγονότος ή συντήρησης του NameNode, παρέχοντας την επιλογή εκτέλεσης δύο

[1] *Το προεπιλεγμένο διάστημα που αποστέλλεται μια αναφορά heartbeat είναι 3 δευτερόλεπτα (dfs.heartbeat.interval=3). Η αναφορά αυτή περιλαμβάνει κα (dfs.blockreport.intervalMec=21600000) σε ms.*

[2] *Το προεπιλεγμένο διάστημα που αποστέλλεται μια αναφορά block (dfs.blockreport.intervalMec=21600000) σε ms.Το Datanode μαζί με τον κτύπο της καρδιάς στέλνει επίσης την αναφορά μπλοκ στο Namenode, η αναφορά μπλοκ τυπικά περιέχει τη λίστα όλων των μπλοκ σε ένα datanode.*

[3] *Ο NameNode γνωρίζει όπως ήδη αναφέραμε όλους τους ενεργούς κόμβους δεδομένων της συστάδας και τα block που περιέχει ο καθένας από αυτούς έτσι όσα από αυτά τα block έχουν συντελεστή αναπαραγωγής (replication factor, default=3) μεγαλύτερο του 1 μπορούν να αναπαραχθούν (άρα αντίστοιχα και τα αρχεία στα οποία ανήκουν).*

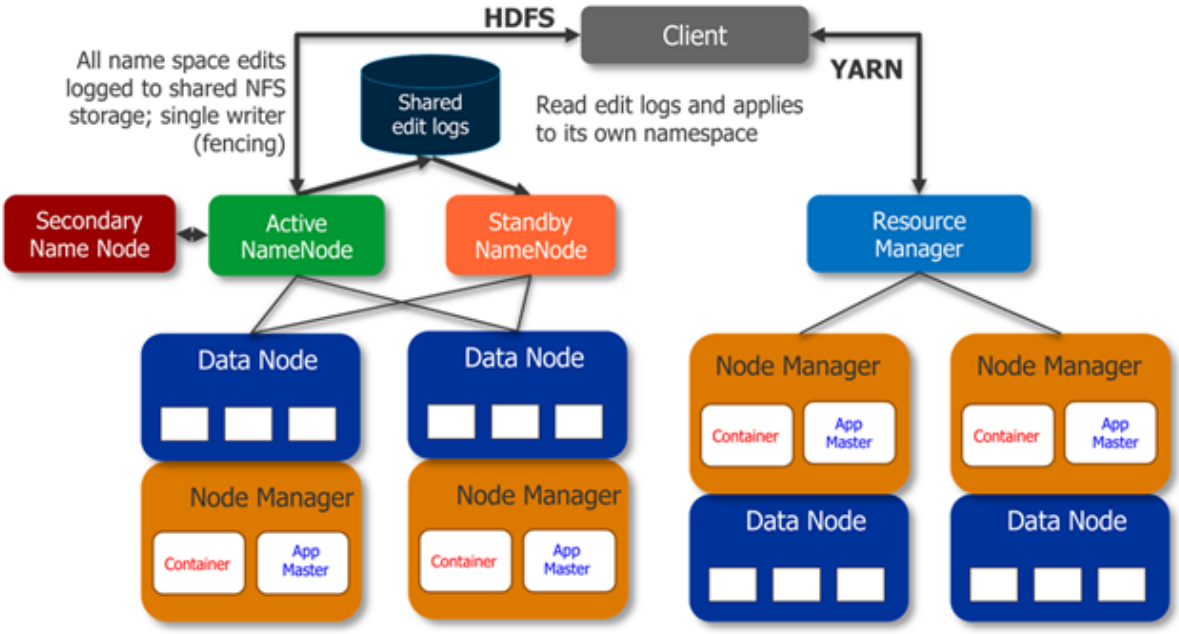
[4] *Οκτώβριος του 2013*

NameNodes [5] (βλέπε Σχήμα 2-3) στην ίδια συστάδα σε ενεργητική / παθητική (active/passive) λειτουργία με καυτή κατάσταση αναμονής (hot standby). Ο **Standby NameNode** είναι συγχρονισμένος με τον active NameNode δηλαδή έχει τα ίδια μεταδεδομένα με αυτόν (βλέπε Σχήμα 2-3) γεγονός που του επιτρέπει την γρήγορη επαναφορά της συστάδας στην προτέρα κατάστασή της, δηλαδή στην κατάστασή της πριν την αποτυχία του NameNode. Το Hadoop 3.0 υποστηρίζει πλέον πολλαπλούς κόμβους αναμονής, γεγονός που αυξάνει περαιτέρω τη διαθεσιμότητα της συστάδας.

- Ο **Δευτερεύον Namenode** [6] (SecondaryNameNode, βλέπε Σχήμα 2-2) είναι βοηθητικός κόμβος του NameNode που εκτελεί εργασίες για λογαριασμό του NameNode και χρησιμοποιείται όταν είναι απενεργοποιημένη η ιδιότητα Υψηλής διαθεσιμότητας στο HDFS. Ο Δευτερεύον NameNode συνενώνει κάθε τόσο το αρχείο καταγραφής edits με το fsimage γεγονός που βοηθά από τη μια μεριά στη μείωση του μεγέθους του αρχείου καταγραφής edits και από την άλλη στο να διατηρείται ενημερωμένο το αρχείο fsimage εξασφαλίζοντας έτσι ότι σε περίπτωση αποτυχίας του NameNode δεν θα χαθεί μεγάλο μέρος μεταδεδομένων. Στη συνέχεια αντιγράφει το ενημερωμένο αρχείο fsimage πίσω στον NameNode ο οποίος θα το χρησιμοποιήσει στην επόμενη επανεκκίνηση, γεγονός που θα οδηγήσει στη μείωση του χρόνου εκκίνησης.
- Οι **DataNodes** αποθηκεύουν και διατηρούν τα μπλοκ δεδομένων. Όταν ένας DataNode ξεκινά αιτείται να εγγραφεί στον NameNode (handshaking process with NN) αποστέλλοντάς του τα Cluster ID, Namespace ID και Blockpool ID. Ο NameNode αφού επιβεβαιώσει την ορθότητα αυτών των IDs στέλνει στον DataNode την επιβεβαίωση εγγραφής ο οποίος του αποστέλλει ως απάντηση μια αναφορά block (block report). Μια Hadoop 2.x συστάδα μπορεί να φιλοξενήσει έως και 10.000 DataNodes. Ο διακομιστής NameNode παρέχει τα μπλοκ δεδομένων βάσει του τύπου εργασίας που υποβάλλει ο πελάτης.
- Η αλληλεπίδραση με το HDFS γίνεται με το πρόγραμμα-πελάτη (client) του συστήματος αρχείων HDFS. Οι υπολογιστές πελάτες hdfs επικοινωνούν πρώτα με τον NameNode για τα μεταδεδομένα και στη συνέχεια επικοινωνούν με τους κόμβους δεδομένων (DN) για την ανάγνωση / εγγραφή δεδομένων (βλέπε Σχήμα 2-4).

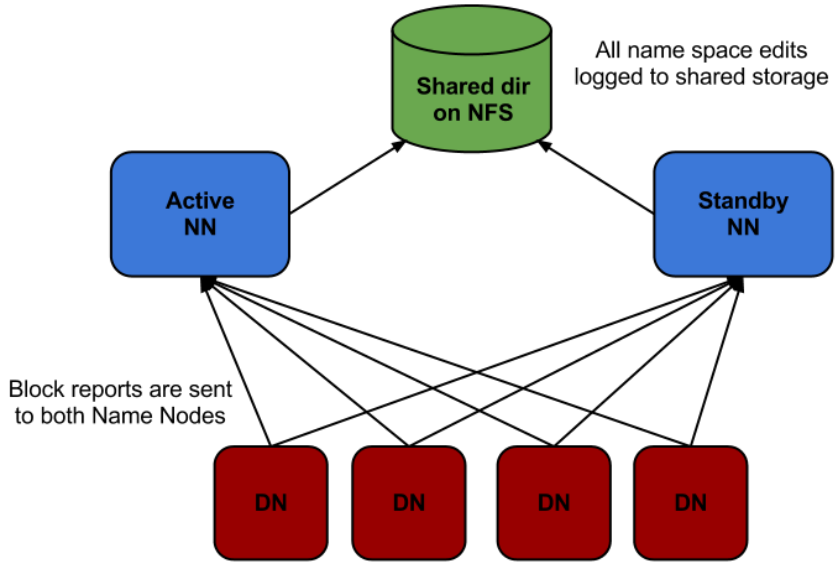
[5] *Ο Standby NameNode είναι συγχρονισμένος με τον active NameNode δηλαδή έχει τα ίδια μεταδεδομένα με αυτόν γεγονός που του επιτρέπει την γρήγορη επαναφορά της συστάδας στην προτέρα κατάστασή της, δηλαδή στην κατάστασή της πριν την αποτυχία του NameNode.*

[6] *Δεν θα πρέπει να συγχέεται με τον Standby NameNode που εξασφαλίζει υψηλή διαθεσιμότητα (HA – High Availability).*



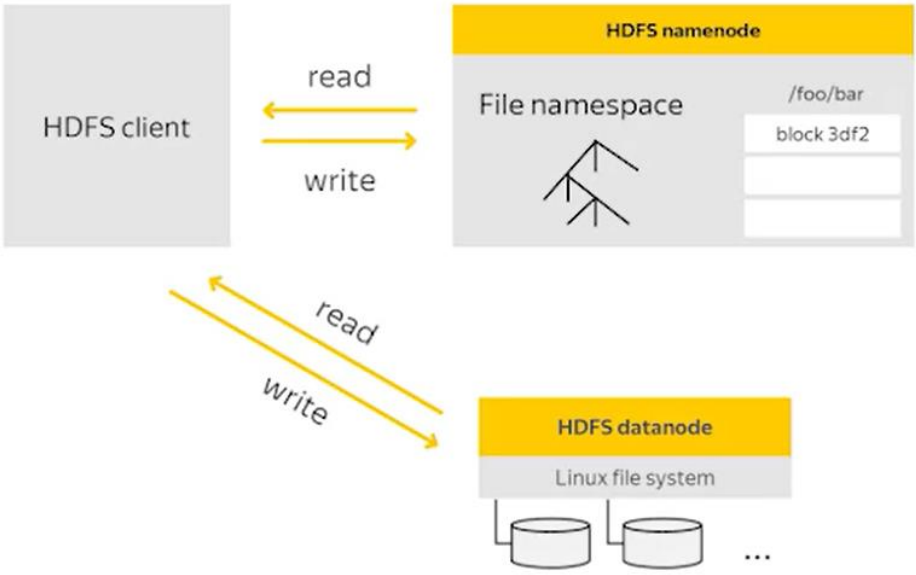
Πηγή: https://www.dezyre.com/article/hadoop-2-0-yarn-framework-the-gateway-to-easier-programming-for-hadoop-users/84

Σχήμα 2-2: Η Αρχιτεκτονική του Hadoop 2.0 (Yarn)



Πηγή: https://www.dezyre.com/article/what-is-hadoop-2-0-high-availability/87

Σχήμα 2-3: Το Το Hadoop με ενεργοποιημένη (enabled) την Υψηλή Διαθεσιμότητα (High Availability – HA).



Πηγή: https://www.coursera.org/lecture/big-data-essentials/hdfs-client-xqbJQ

Σχήμα 2-4: Οι υπολογιστές πελάτες hdfs επικοινωνούν πρώτα με τον NameNode για τα μεταδεδομένα και στη συνέχεια επικοινωνούν με τους κόμβους δεδομένων (DN) για την ανάγνωση / εγγραφή δεδομένων.

YARN

Όπως φαίνεται στο Σχήμα 2.1 , το Apache Hadoop YARN βρίσκεται μεταξύ του HDFS και των μηχανών επεξεργασίας που χρησιμοποιούνται για την εκτέλεση εφαρμογών. Το YARN ακολουθεί την αρχιτεκτονική master-slave (βλέπε Σχήμα 2.2). Η master διεργασία (δαίμονας - daemon) ονομάζεται Διαχειριστής Πόρων (ResourceManager, RM) και η διεργασία (δαίμονας) slave ονομάζεται Διαχειριστής Κόμβου (NodeManager, NM).

Η αρχιτεκτονική του Apache Hadoop 2.x (Yarn) αποτελείται από τα ακόλουθα στοιχεία:

- Τον **Διαχειριστή Πόρων - RM** (βλέπε Σχήμα 2.5α) που αποτελείται από δύο στοιχεία τον Χρονοδρομολογητή (Scheduler) και τον Διαχειριστή Εφαρμογών (Applications Manager, AsM). Ο **Χρονοδρομολογητής** κατανέμει πόρους στις διάφορες εφαρμογές που τρέχουν. Διατηρεί μια κατάσταση με τις εφαρμογές που εκτελούνται στη συστάδα και μια με τους διαθέσιμους πόρους σε κάθε ζωντανό NodeManager (βλέπε Σχήμα 2-5β). Για να επιτρέπονται διαφορετικοί περιορισμοί πολιτικής, το YARN χρησιμοποιεί pluggable Χρονοδρομολογητές που υποστηρίζουν τη χρήση διαφορετικών αλγορίθμων, όπως η χρονοδρομολόγηση χωρητικότητας (capacity scheduling) και η

χρονοδρομολόγηση δίκαιης κατανομής (fair scheduling [7]). Ο **Διαχειριστής Εφαρμογών** (AsM) είναι υπεύθυνος για την αποδοχή των υποβαλλόμενων από τον πελάτη εργασιών (jobs), την αρχικοποίηση του Application Master (AM) για την συγκεκριμένη εργασία και την παρακολούθηση της κατάστασης των Application Masters και σε περίπτωση αποτυχίας ενός Application Master επανεκκινεί το container στο οποίο έτρεχε.

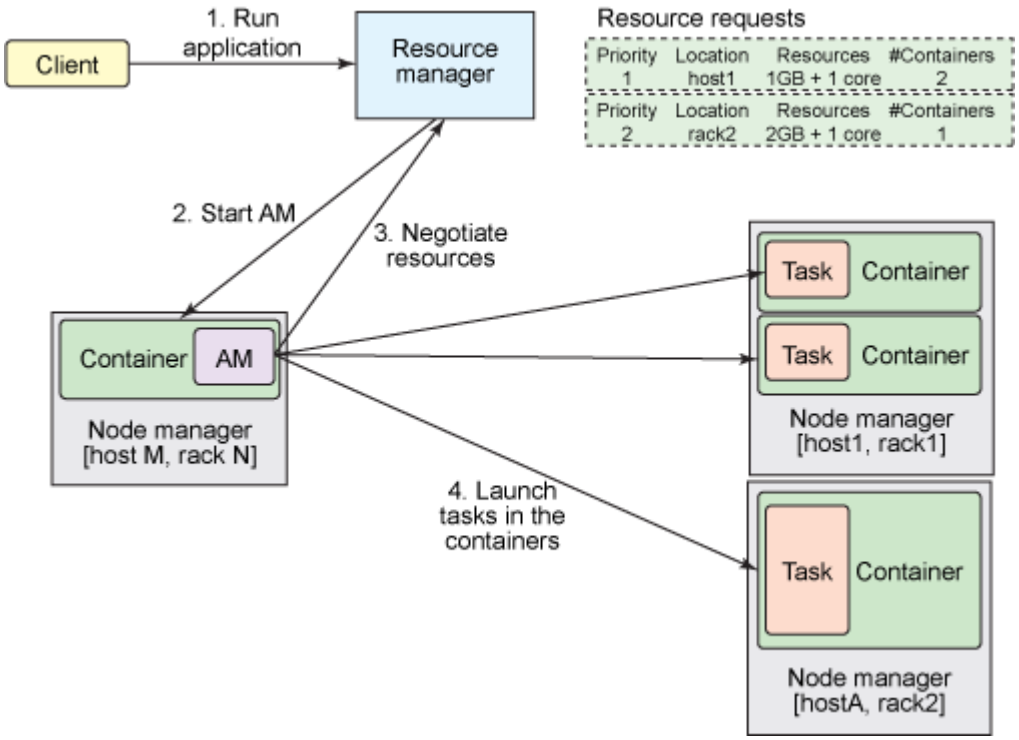
- Τον **Application Master -AM** (βλέπε Σχήμα 2.5β) που είναι ένας ανά εφαρμογή και μπορεί να δημιουργηθεί σε οποιονδήποτε κόμβο slave και διατηρείται ζωντανός κατά τη διάρκεια της ζωής μιας εφαρμογής. Έχει την ευθύνη της διαπραγμάτευσης πόρων υπό την μορφή containers από το ResourceManager, την παρακολούθηση της κατάστασής τους και της προόδου τους. Συνεργάζεται με το NodeManager για την εκτέλεση και την παρακολούθηση των έργων (tasks).
- Τους **Διαχειριστές Κόμβων - NM** (βλέπε Σχήμα 2.5β) που είναι ένας ανά κόμβο και είναι υπεύθυνοι για τα containers, την ενημέρωση του Διαχειριστή Πόρων με τα αποτελέσματα παρακολούθηση της χρήσης των πόρων τους, την εκτέλεση ενός έργου σε κάθε έναν κόμβο δεδομένων και την διατήρηση αρχείων καταγραφής των κοντέινερ είτε τοπικά σε δίσκους είτε φορτώνοντάς τα στο κατανεμημένο σύστημα αρχείων αφού πρώτα τα συμπιέσει.
- Τα **Containers** (βλέπε Σχήμα 2.5β) αντιπροσωπεύουν τους δεσμευμένους πόρους (μνήμη, cpu, δίκτυο, δίσκοι) ενός κόμβου και έχουν ένα μοναδικό Container Id. Ο αποκλειστικά υπεύθυνος για την εκχώρηση οποιουδήποτε Container σε μια εφαρμογή είναι ο RM.

[7] *Από προεπιλογή, το Hadoop 2.x έχει ενεργοποιημένο τον Χρονοδρομολογητή Χωρητικότητας (Capacity Scheduler)*

Ο Χρονοδρομολογητής κατανέμει πόρους στις διάφορες εφαρμογές που τρέχουν. Το YARN υποστηρίζει τον Capacity Scheduler και τον Fair Scheduler.

Ο Διαχειριστής Εφαρμογών είναι υπεύθυνος για την αποδοχή των υποβαλλόμενων από τον πελάτη εργασιών (jobs), την αρχικοποίηση του ApplicationMaster μιας συγκεκριμένης εργασίας και την επανεκκίνησή του σε περίπτωση αποτυχίας.

(α)



Πηγή: https://sparkbyexamples.com/hadoop/how-yarn-works/

(β)

Σχήμα 2-5: (α) Τα στοιχεία που απαρτίζουν τον Διαχειριστή Πόρων (Resource Manager, RM): Ο Χρονοδρομολογητής (Scheduler) και ο Διαχειριστής Εφαρμογών (Applications Manager, AsM), (β) Υποβολή εφαρμογής στο Apache YARN

MapReduce

Το **MapReduce** είναι ένας κατανεμημένος αλγόριθμος για την επεξεργασία μεγάλων δεδομένων σε δύο φάσεις (Map, Reduce). Κάθε φάση απαιτεί τον προγραμματισμό μιας συνάρτησης (map, reduce), γραμμένης σε γλώσσα γενικής χρήσης όπως Java, στην οποία καθορίζεται ή λογική της επεξεργασίας. Η συνάρτηση map λαμβάνει ένα σύνολο δεδομένων

και το μετατρέπει σε άλλο σύνολο δεδομένων (ενδιάμεσες τιμές), όπου τα μεμονωμένα στοιχεία κατανέμονται σε πλειάδες (ζεύγη κλειδιού / τιμής). Η συνάρτηση reduce παίρνει την έξοδο από μια συνάρτηση map ως είσοδο και συνδυάζει αυτές τις πλειάδες δεδομένων σε ένα μικρότερο σύνολο πλειάδων. Στο προγραμματιστικό μοντέλο MapReduce οι εργασίες (jobs) επεξεργασίας δεδομένων ονομάζονται mappers και reducers. Γενικά το προγραμματιστικό υπόδειγμα MapReduce βασίζεται στην αποστολή των υπολογισμών στον τόπο όπου βρίσκονται τα δεδομένα (τοπικότητα δεδομένων - data locality)!

Το MapReduce είναι μια πολύ απλούστερη προσέγγιση για υπολογισμούς μεγάλης κλίμακας που αποσκοπεί στην απόκρυψη της πολυπλοκότητας της παράλληλης επεξεργασίας [24]. Αυτό είναι ένα σημαντικό πλεονέκτημα για την αναλυτική δεδομένων, καθώς απαλλάσσει τον προγραμματιστή από το βάρος να είναι ειδικός στην αντιμετώπιση πολύπλοκων μοντέλων παράλληλης επεξεργασίας και απαιτεί απ' αυτόν να είναι ειδικός στην εξαγωγή πολύτιμης γνώσης από τα δεδομένα μέσω υπολογισμού [6].

Η φάση **Map** αποτελείται από δυο βήματα (βλέπε Σχήμα 2.6):

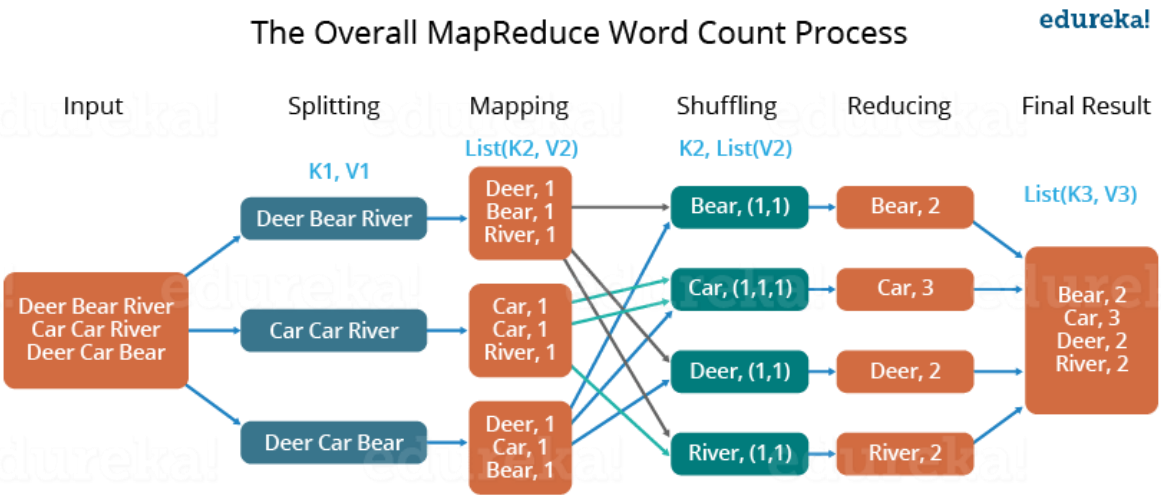
- **Splitting** - Μια είσοδος σε μια εργασία MapReduce χωρίζεται σε υποσύνολα δεδομένων σταθερού μεγέθους που ονομάζονται inputSplits. Το InputSplit αντιπροσωπεύει τα δεδομένα που καταναλώνονται από έναν mapper. Συνεπώς ο αριθμός των mapper ισούται με τον αριθμό των inputSplits. Στη συνέχεια ο RecordReader θα επικοινωνήσει με το inputSplit και θα διαιρέσει τα δεδομένα σε ζεύγη κλειδιού-τιμής που είναι κατάλληλα για ανάγνωση από τον mapper.
- **Mapper** - υλοποιεί τη συνάρτηση map (k1, v1) -> list (k2, v2), στην οποία ο κώδικας που παρέχεται από το χρήστη εκτελείται σε κάθε ζεύγος κλειδιού-τιμής (k1, v1) που διαβάζεται από τα αρχεία εισόδου. Η έξοδος της συνάρτησης map είναι μια λίστα με μηδέν ή περισσότερα ζεύγη κλειδιού-τιμής list(k2, v2) που ονομάζονται ενδιάμεσα ζεύγη. Εδώ το κλειδί είναι τι δεδομένα θα ομαδοποιηθούν και η τιμή είναι οι πληροφορίες που σχετίζονται με την ανάλυση στον Reducer. Οι τύποι δεδομένων των (k1,v1) μπορεί να είναι διαφορετικοί από τους τύπους των (k2,v2).

Η φάση **Reduce** ξεκινά με το βήμα (βλέπε Σχήμα 2.6):

- **Shuffle (ανακατανομή) και Sort (ταξινόμηση)**. Κατά το βήμα αυτό μεταφορτώνονται τα ζεύγη ομαδοποιημένων κλειδιών-τιμών στον τοπικό κόμβο όπου τρέχει ο Reducer. Διεξάγεται μια διαδικασία διαμοιρασμού όπου γίνεται η ταξινόμηση και η ανακατάταξη έτσι ώστε όλες οι πλειάδες με το ίδιο κλειδί να αποστέλλονται στον αντίστοιχο Reducer. Τα μεμονωμένα ζεύγη κλειδιών-τιμών ταξινομούνται ανά κλειδί

σε μια μεγαλύτερη λίστα δεδομένων. Στη λίστα δεδομένων ομαδοποιούνται τα αντίστοιχα κλειδιά έτσι ώστε να διευκολύνεται η χρήση των τιμών τους σε επαναληπτικές διαδικασίες στον Reducer (k2, list(v2)), εκτελεί ανακατανομή (shuffling) της λίστας εξόδου στις κατάλληλες συναρτήσεις reduce () έτσι ώστε λογικά η συνάρτηση reduce () να επεξεργάζεται το ίδιο κλειδί (k2) και ενδιάμεση τιμή (v2).

- **Reducer**. Ο Reducer τρέχει τη συνάρτηση reduce (k2, list (v2)) για κάθε μοναδικό κλειδί k2 και την αντίστοιχη λίστα τιμών (v2). Εδώ, τα δεδομένα μπορούν να συγκεντρωθούν, να φιλτραριστούν και να συνδυαστούν με διάφορους τρόπους. Μόλις ολοκληρωθεί η συνάρτηση reduce, στέλνει μηδέν ή περισσότερα ζεύγη κλειδιού-τιμής - list (k3, v3) - στο τελικό αρχείο εξόδου. Οι τύποι των (k2,v2) μπορεί να είναι διαφορετικοί από τους τύπους των (k3,v3). Οι εργασίες reduce ξεκινούν αμέσως μετά την ολοκλήρωση των εργασιών map.



Φάση Map	Φάση Reduce

Πηγή: https://www.edureka.co/blog/mapreduce-tutorial

Σχήμα 2-6: Παράδειγμα MapReduce

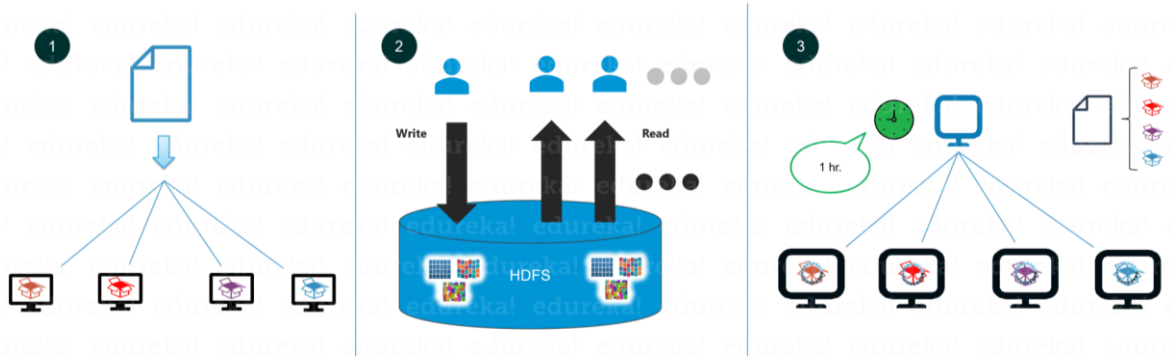
Το Hadoop ως λύση στις προκλήσεις των μεγάλων δεδομένων

Ας δούμε στη συνέχεια πως αντιμετωπίζει το Hadoop τις προκλήσεις που εισάγουν τα μεγάλα δεδομένα:

Η λύση στο πρόβλημα της αποθήκευσης των μεγάλων δεδομένων είναι το HDFS. Το οποίο παρέχει έναν κατανεμημένο τρόπο αποθήκευσης μεγάλων δεδομένων (Σχήμα 2.7-1). Κάθε αρχείο αποθηκεύεται ως ακολουθία μπλοκ, όπου όλα τα μπλοκ σε ένα αρχείο εκτός από το

τελευταίο μπλοκ έχουν το ίδιο μέγεθος. Ας υποθέσουμε ότι ο χρήστης έχει ένα αρχείο με μέγεθος 548MB και έχει ορίσει το μέγεθος των μπλοκ δεδομένων να είναι 128MB. Το HDFS θα χωρίσει τα δεδομένα σε 4 μπλοκ των 128 MB και ένα μπλοκ των 36 MB και θα τα αποθηκεύσει σε διαφορετικούς Κόμβους Δεδομένων (DataNodes). Κατά την αποθήκευση αυτών των μπλοκ δεδομένων σε DataNodes, τα μπλοκ δεδομένων αναπαράγονται σε διαφορετικά DataNodes για να παρέχουν ανοχή σφάλματος (fault tolerance) καθώς με αυτό τον τρόπο αυξάνει τη διαθεσιμότητα των δεδομένων ανά πάσα στιγμή. Τόσο ο παράγοντας αναπαραγωγής (προκαθορισμένη τιμή: 3, δηλαδή το πρωτότυπο μπλοκ και 2 αντίγραφα μπλοκ) όσο και το μέγεθος των μπλοκ (προκαθορισμένη τιμή :128MB) είναι ιδιότητες που μπορούν να οριστούν στο αρχείο διαμόρφωσης HDFS (configuration file) [21].

Με το Hadoop μπορεί κανείς να προσθέσει νέους κόμβους (οριζόντια κλιμάκωση) σε συστάδες HDFS που βρίσκονται σε λειτουργία σύμφωνα κάθε φορά με την απαίτηση, αντί να αυξήσει τη στοίβα υλικού που υπάρχει σε κάθε κόμβο (κάθετη κλιμάκωση).



Πηγή: https://www.edureka.co/blog/hadoop-tutorial

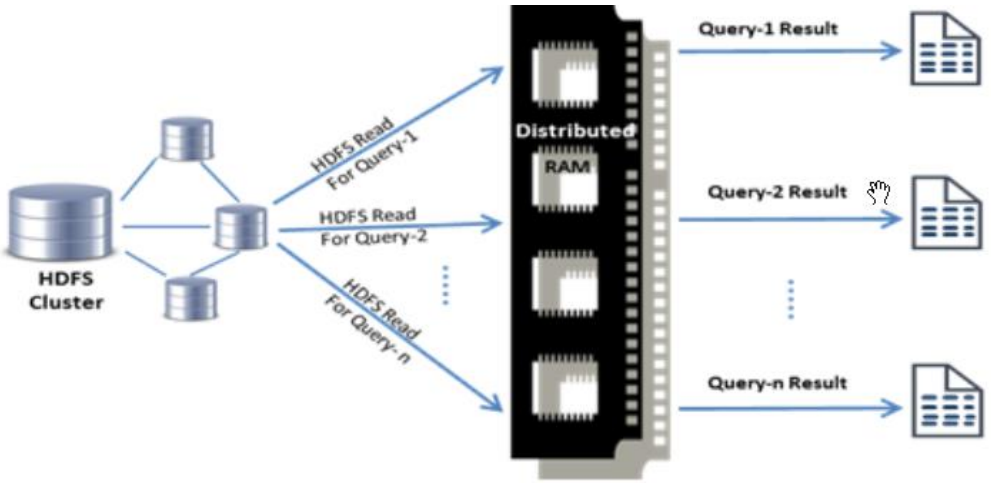
Σχήμα 2-7: Το Hadoop ως λύση στις προκλήσεις των μεγάλων δεδομένων

Η ευελιξία (flexibility) του HDFS είναι η λύση και για πρόβλημα της αποθήκευσης ετερογενών δεδομένων. Όπως φαίνεται στο παραπάνω σχήμα, το HDFS υποστηρίζει την αποθήκευση δομημένων, ημιδομημένων και μη δομημένων δεδομένων. Ακολουθεί επίσης το μοντέλο "γράψε μια φορά και διάβασε πολλές" (write once and read many) που επιτρέπει τον χρήστη να γράψει ένα είδος δεδομένων μία μόνο φορά και ακολούθως μπορεί να το διαβάσει πολλές φορές για να εντοπίσει τις πληροφορίες που τον ενδιαφέρουν [21].

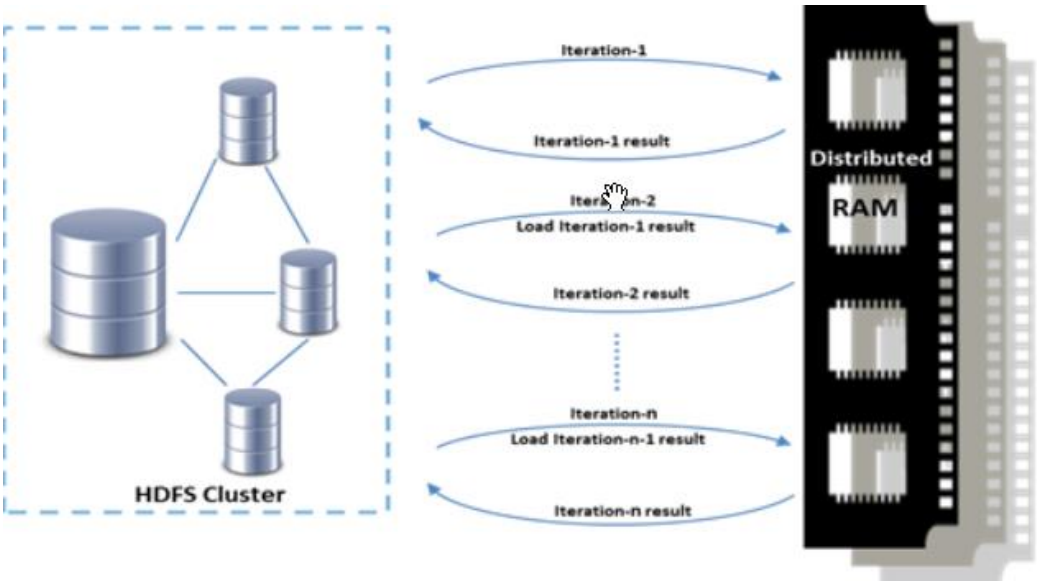
Η τρίτη πρόκληση αφορούσε την ταχύτερη επεξεργασία των δεδομένων. Η λύση ακούει στο όνομα παράλληλη επεξεργασία και τοπικότητα δεδομένων (data locality) που εξασφαλίζει η εγγενής υπολογιστική μηχανή MapReduce του Hadoop. Η τοπικότητα δεδομένων αναφέρεται στη μετακίνηση του υπολογισμού (υπολογιστικής εργασίας) στον κόμβο όπου βρίσκονται τα δεδομένα αντί της μετακίνησης μεγάλου όγκου δεδομένων στον υπολογισμό. Δηλαδή με απλά

λόγια **τοπικότητα δεδομένων** σημαίνει ότι ο υπολογισμός γίνεται στον κόμβο όπου βρίσκονται τα δεδομένα έτσι ελαχιστοποιείται η συμφόρηση του δικτύου και αυξάνει η συνολική απόδοση του συστήματος. Κάθε κόμβος επεξεργάζεται ένα μέρος των δεδομένων παράλληλα. Τα ενδιάμεσα αποτελέσματα των κόμβων συνενώνονται και το τελικό αποτέλεσμα αποστέλλεται πίσω στον πελάτη (client). Η ταχύτατη επεξεργασία μεγάλου όγκου δεδομένων απαιτεί τα δεδομένα να επεξεργάζονται τόσο παράλληλα όσο και τοπικά [21].

Ωστόσο, το **Hadoop MapReduce** είναι κατάλληλο για τη διενέργεια εργασιών σε μεγάλα δεδομένα όπως είναι για παράδειγμα οι εργασίες ενσωμάτωσης δεδομένων (ETL) και επεξεργασίας δέσμης - ή κατά παρτίδες - (batch processing) γενικότερα. Αντίθετα είναι αργό σε διαδραστικές αναλυτικές εργασίες όπως η διενέργεια ad hoc ερωτήσεων σε σύνολα δεδομένων και σε επαναληπτικές εργασίες όπως η μηχανική μάθηση. Για παράδειγμα στις ad hoc ερωτήσεις αυτό συμβαίνει γιατί η ταχύτητα διεκπεραίωσης δεν εξαρτάται από την ταχύτητας της CPU και της μνήμης RAM αλλά περιορίζεται από την ταχύτητα του δικτύου αλλά και την ταχύτητα I/O των δίσκων της συστοιχίας αφού κάθε φορά που γίνεται ένα ερώτημα στο Hadoop, τα δεδομένα διαβάζονται από το δίσκο (HDFS-read) και φορτώνονται στη μνήμη – πράγμα που εισάγει σημαντικές καθυστερήσεις (βλέπε Σχήμα 2.8-α). Σε εργασίες που απαιτούν επαναλήψεις (βλέπε Σχήμα 2.8-β) επειδή το μοντέλο προγραμματισμού του MapReduce δεν επιτρέπει την επικοινωνία μεταξύ map και reduce workers, πρέπει τα ενδιάμεσα αποτελέσματα να αποθηκεύονται σε σταθερή αποθήκευση (HDFS). Ο αριθμός των I / O δίσκου εξαρτάται από τον αριθμό των επαναλήψεων που εμπεριέχονται σε έναν αλγόριθμο και σε αυτή την καθυστέρηση θα πρέπει να προστεθεί και η χρονική επιβάρυνση που εισάγουν οι λειτουργίες serialization / deserialization (βλέπε Σχήμα 2.9) κατά την αποθήκευση και φόρτωση των δεδομένων [20]. Επίσης ένας τυπικός αλγόριθμος μηχανικής μάθησης μπορεί να χρειαστεί να κάνει 10 ή 20 περάσματα πάνω από τα ίδια δεδομένα και στο MapReduce, κάθε πέρασμα θα έπρεπε να γραφτεί ως χωριστή εργασία MapReduce, η οποία θα έπρεπε να ξεκινήσει ξεχωριστά στη συστάδα και να φορτώσει τα δεδομένα από το μηδέν [1].



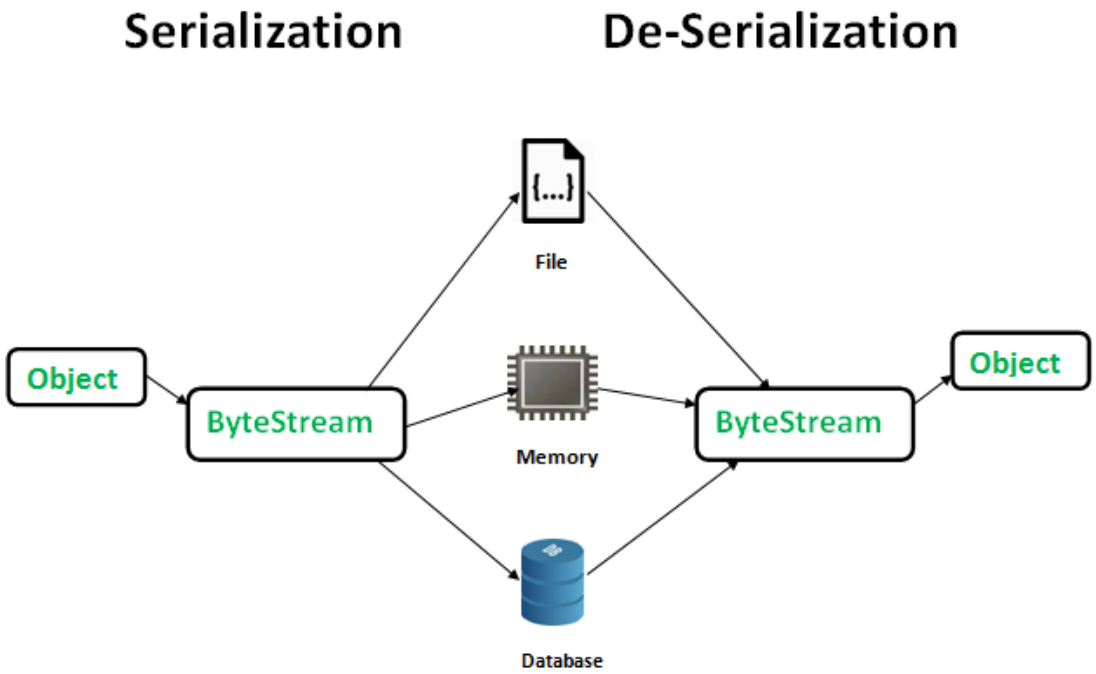
(α)



(β)

Πηγή: [20]

Σχήμα 2-8: To Hadoop Mapreduce είναι αργό: (a) Σε διαδραστικές αναλυτικές εργασίες όπως η διενέργεια ad hoc ερωτήσεων σε σύνολα δεδομένων γιατί η ταχύτητα διεκπεραίωσης δεν εξαρτάται από την ταχύτητας της CPU και της μνήμης RAM αλλά περιορίζεται από την ταχύτητα του δικτύου αλλά και την ταχύτητα I/O των δίσκων της συστοιχίας, (β) Σε εργασίες που απαιτούν επαναλήψεις επειδή το μοντέλο προγραμματισμού του MapReduce δεν επιτρέπει την επικοινωνία μεταξύ map και reduce workers, θα πρέπει τα ενδιάμεσα αποτελέσματα να αποθηκεύονται σε σταθερή αποθήκευση (HDFS). Ο αριθμός των I / O δίσκου εξαρτάται από τον αριθμό των επαναλήψεων που εμπεριέχονται σε έναν αλγόριθμο και σε αυτή την καθυστέρηση θα πρέπει να προστεθεί και η χρονική επιβάρυνση που εισάγουν οι λειτουργίες serialization / deserialization



Πηγή: https://www.geeksforgeeks.org/serialization-in-java

Σχήμα 2-9: Serialization - Είναι ένας μηχανισμός μετατροπής της κατάστασης ενός αντικειμένου σε byte-stream. Deserialization - Είναι η αντίστροφη διαδικασία όπου χρησιμοποιείται το byte-stream για την αναδημιουργία του πραγματικού αντικειμένου Java στη μνήμη. Αυτός ο μηχανισμός χρησιμοποιείται για την διατήρηση του αντικειμένου.

Κεφάλαιο 3
Εισαγωγή στο Apache Spark

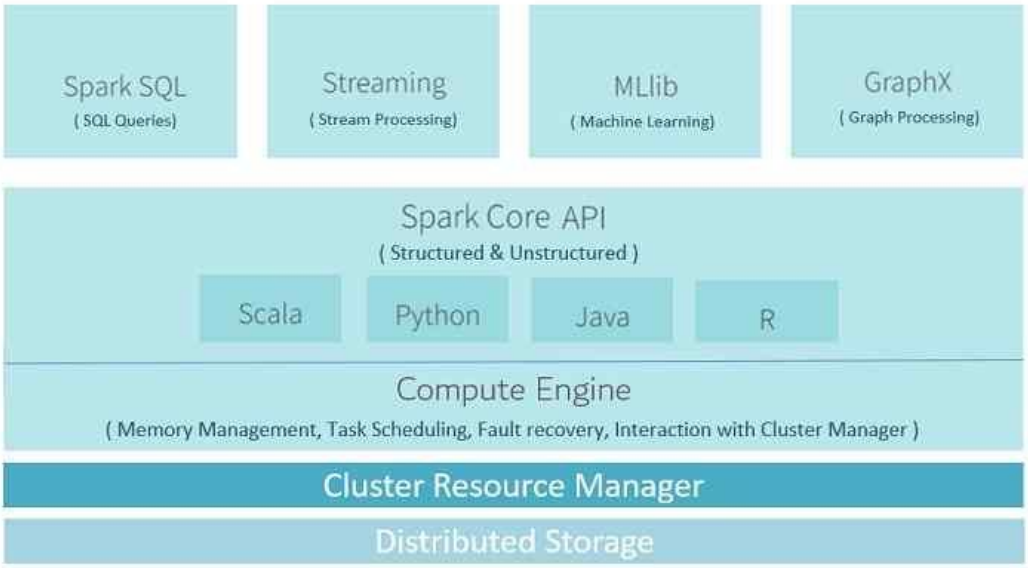
Η ιστορία του Spark

Το Spark ξεκίνησε στο εργαστήριο AMPlab του πανεπιστημίου UC Berkeley το 2009 ως ένα έργο που θα διατηρούσε τα πλεονεκτήματα του Hadoop MapReduce ως ένα πλαίσιο κλιμακώσιμο, κατανεμημένο, ανεκτικό σε σφάλματα αλλά ταυτόχρονα θα έδινε λύσεις στα προβλήματά του που αναφέρθηκαν παραπάνω (κακή απόδοση σε διαδραστικές αναλυτικές εργασίες και επαναληπτικές εργασίες) και θα ήταν συγχρόνως και ευκολότερο στη χρήση του. Το εγχείρημα είχε μεγάλη απήχηση και μετατράπηκε σε λογισμικό ανοικτού κώδικα το 2010. Το 2013 είχε πάνω από 100 συνεισφέροντες (contributors) από περισσότερους από 30 οργανισμούς πέραν του πανεπιστημίου UC Berkeley. Την ίδια χρονιά το Spark παραχωρείται στο Apache Software Foundation, ενώ η αρχική ομάδα του AMPlab ιδρύει την εταιρεία Databricks με κύριο σκοπό την ενδυνάμωση του έργου. Η έκδοση 1.0 του Spark κυκλοφόρησε το 2014, η έκδοση 2.0 το 2016 ενώ η τελευταία σταθερή έκδοση (2.4.3) κυκλοφόρησε το Μάιο του 2019. Αυτή τη στιγμή είναι η πιο ενεργά αναπτυσσόμενη μηχανή ανοιχτού κώδικα για επεξεργασία μεγάλων δεδομένων [1].

Στην αρχή υποστήριζε μόνο εφαρμογές επεξεργασίας δέσμης, σύντομα όμως οι δημιουργοί του κατάλαβαν πως το έργο θα έπρεπε να ακολουθήσει την προσέγγιση της "τυπικής βιβλιοθήκης" (standard library). Προσθέτοντας βιβλιοθήκες όπως το Spark Streaming, GraphX, MLlib , Spark SQL και εξασφαλίζοντας την υψηλή διαλειτουργικότητα αυτών των Προγραμματιστικών Διεπαφών (APIs) επέτρεψαν το Spark να γίνει η πρώτη ενοποιημένη υπολογιστική μηχανή (unified computing engine) ανοικτού κώδικα. Η προσθήκη του SparkSQL API (έκδοση Spark 1.0), για την επεξεργασία δομημένων δεδομένων, επέτρεψε μια σειρά από βελτιστοποιήσεις τόσο στις βιβλιοθήκες όσο και στα API, κατανοώντας με περισσότερη λεπτομέρεια τόσο τη μορφή των δεδομένων όσο και τον κώδικα του χρήστη που εκτελείται σε αυτά [1]. Με την πάροδο του χρόνου, προστέθηκαν μια σειρά νέων API που βασίζονται σε αυτή την ισχυρότερη δομημένη βάση, συμπεριλαμβανομένων των DataFrames, διασωληνώσεων (pipelines), μηχανικής μάθησης και Structured Streaming, ένα υψηλού επιπέδου, αυτόματα βελτιστοποιούμενο API [1].

Η αρχιτεκτονική του Spark

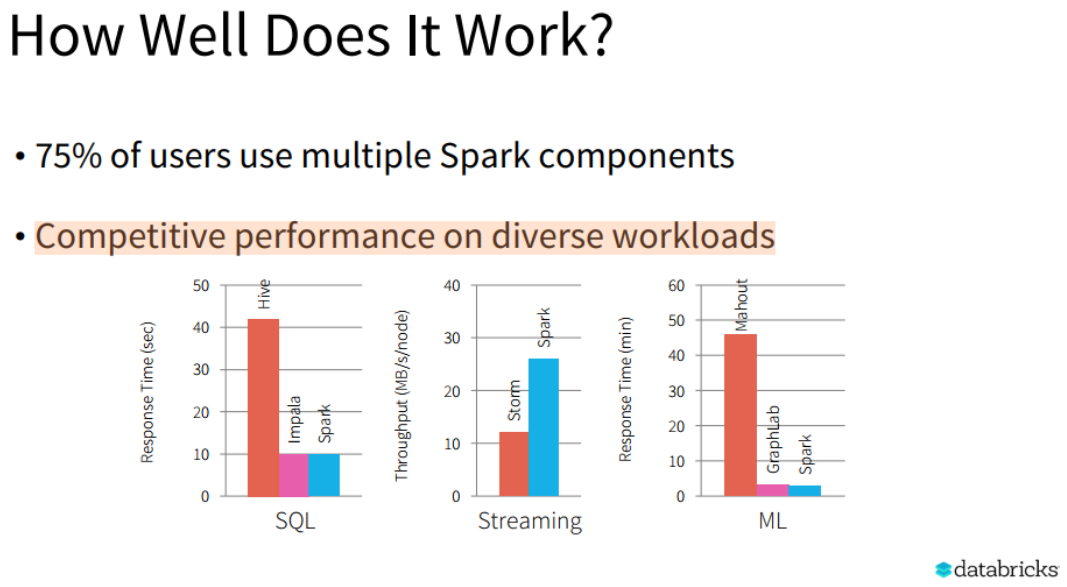
Το Spark είναι ένα ενοποιημένο πλαίσιο (framework) υπολογιστικής συμπλέγματος, για την ανάλυση μεγάλων δεδομένων. Κύρια χαρακτηριστικά του είναι η ευκολία χρήσης, η ταχύτητα, η ανοχή του σε σφάλματα και η γραμμική κλιμάκωσή του. Όπως φαίνεται και στο Σχήμα 3.1, περιλαμβάνει μια στοίβα βιβλιοθηκών για ένα ευρύ φάσμα ροών εργασίας όπως Spark SQL για διενέργεια ερωτημάτων SQL και επεξεργασία δομημένων δεδομένων, Spark Streaming για επεξεργασία ροής δεδομένων , MLlib για μηχανική μάθηση και GraphX για ανάλυση γράφων. Οι εφαρμογές Spark μπορούν να υλοποιηθούν σε Java, Scala, Python, R και SQL ενώ η ενοποιημένη φύση του Spark επιτρέπει στον προγραμματιστή να συνδυάζει απρόσκοπτα διαφορετικούς τύπους επεξεργασίας και βιβλιοθήκες μέσα στην ίδια εφαρμογή Spark σε αντίθεση με προγενέστερα πλαίσια λογισμικού για κατανεμημένη επεξεργασία και ανάλυση μεγάλων δεδομένων όπου έπρεπε να καταφύγει σε εξωτερικές προγραμματιστικές διεπαφές (API) και συστήματα. Το Apache Spark απαιτεί έναν διαχειριστή συστάδας και ένα κατανεμημένο σύστημα αποθήκευσης για να λειτουργήσει (βλέπε Σχήμα 3.1). Το σκεπτικό των δημιουργών του Spark να επικεντρωθεί στη διενέργεια υπολογισμών ανεξάρτητα από το που βρίσκονται αυτά βασίζεται στη διαπίστωση ότι η μετακίνηση των δεδομένων είναι πράξη δαπανηρή. Ωστόσο, το Spark δεν αποθηκεύει δεδομένα μακροπρόθεσμα, ούτε ευνοεί κάποιο από τα συστήματα μόνιμης αποθήκευσης. Η προσπάθεια του επικεντρώνεται στη δημιουργία APIs που θα κάνει τα συστήματα αυτά να φαίνονται παρόμοια, ώστε οι εφαρμογές να έχουν πρόσβαση στα δεδομένα τους ανεξάρτητα από το που βρίσκονται. Όλες οι λειτουργίες που παρέχονται από το Apache Spark είναι δομημένες πάνω στο Spark Core. Στο Σχήμα 3.2 φαίνεται η υπεροχή του Spark έναντι του ανταγωνισμού σε διαφορετικά είδη επεξεργασίας δεδομένων όπως και το γεγονός ότι 75% των χρηστών του Spark συνδυάζει κάποιες από τις βιβλιοθήκες του.



Πηγή: https://www.learningjournal.guru/article/apache-spark/apache-spark-introduction

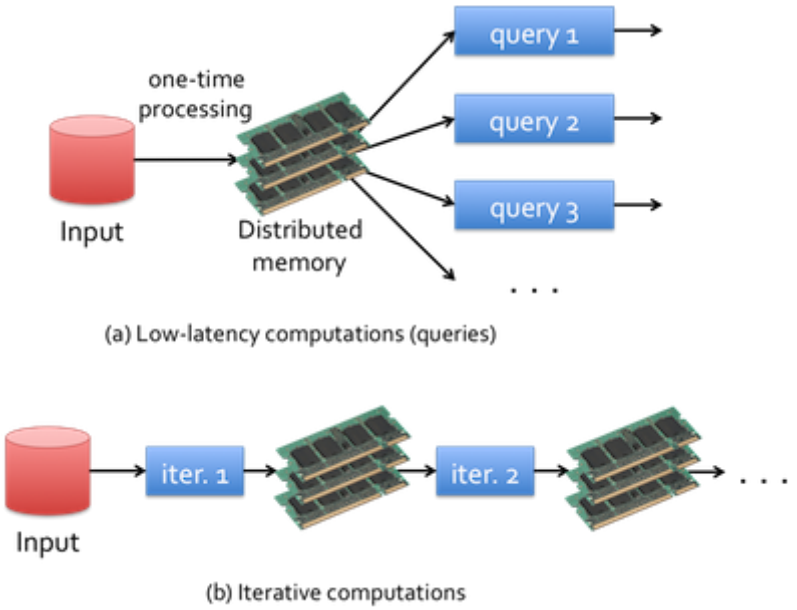
Σχήμα 3-1: Τα δομικά στοιχεία του οικοσυστήματος Spark (με πράσινο ανοικτό χρώμα)

Το Apache Spark επιτυγχάνει υψηλή απόδοση τόσο για δεδομένα δέσμης όσο και για δεδομένα συνεχούς ροής, χρησιμοποιώντας έναν υπερσύγχρονο Χρονοδρομολογητή (Scheduler) DAG (Directed Acyclic Graph – Κατευθυνόμενου Άκυκλου Γράφου), έναν βελτιστοποιητή επερωτήσεων και μια μηχανή φυσικής εκτέλεσης [23]. Το Spark εκμεταλλεύεται τις δυνατότητες – την ταχύτητα και την επεκτασιμότητα – που προσφέρει η εντός της (κατανεμημένης) μνήμης (in-memory) – παράλληλη – επεξεργασία δεδομένων, σε αντίθεση με το Hadoop MapReduce (βλέπε κεφάλαιο 3), όπου τα MapReduce jobs τείνουν να γράφουν συνεχώς τα ενδιάμεσα δεδομένα πίσω στο δίσκο [24]. Όπως απεικονίζεται στο Σχήμα 3.3, αυτό το χαρακτηριστικό επιτρέπει: (α) χαμηλή καθυστέρηση σε διαδραστικές αναλυτικές εργασίες και (β) ταχύτατη επεξεργασία επαναληπτικών εργασιών. Σύμφωνα με το επίσημο ιστότοπό του, το Apache Spark είναι μέχρι 100 φορές ταχύτερο από το Apache Hadoop (MapReduce) όταν τα δεδομένα αποθηκεύονται στη μνήμη και έως και 10 φορές όταν αποθηκεύονται στο δίσκο [23].



Πηγή: https://www.slideshare.net/MatthewStubbs6/big-data-ldn-2017-unifying-big-data-workloads-in-apache-spark

Σχήμα 3-2: Ανταγωνιστική απόδοση του Spark σε διαφορετικές ροές εργασίας (SQL, Συνεχούς Ροής, Μηχανικής Μάθησης)



Πηγή: https://databricks.com/blog/2013/11/21/putting-spark-to-use.html

Σχήμα 3-3: Η εντός της μνήμης (in-memory) – παράλληλη – επεξεργασία δεδομένων του Spark προσφέρει βελτιωμένη απόδοση (α) σε διαδραστικές αναλυτικές εργασίες (Interactive Analytics jobs), και (β) επαναληπτικές εργασίες (iterative computations)

Spark Core

Το **Spark Core** είναι το υπόβαθρο της μηχανής επεξεργασίας κατανεμημένων δεδομένων του Spark. Αποτελείται από την κατανεμημένη υπολογιστική υποδομή και από τα core API τα οποία περιλαμβάνουν τα μη δομημένα API και τα δομημένα API.

Η **κατανεμημένη υπολογιστική υποδομή** είναι υπεύθυνη για τη διανομή, τον συντονισμό και τον προγραμματισμό των έργων[8] υπολογισμού σε πολλούς κόμβους στη συστάδα. Αυτό δίνει τη δυνατότητα παράλληλης επεξεργασίας δεδομένων μεγάλου όγκου αποτελεσματικά και γρήγορα σε ένα μεγάλο σύνολο μηχανών. Επίσης είναι υπεύθυνη για το χειρισμό των αποτυχημένων έργων υπολογισμού και την αποτελεσματική μεταφορά δεδομένων μεταξύ κόμβων, η οποία είναι γνωστή ως ανακατανομή δεδομένων (shuffling) [7]. Παρέχει επίσης ένα ενοποιημένο διαχειριστή μνήμης, το Spark local ένα από τα διαθέσιμα περιβάλλοντα εκτέλεσης χρόνου και το μόνο που δεν απαιτεί διαχειριστή συστάδας καθώς επίσης και βασικές λειτουργίες, όπως η δικτύωση, η ασφάλεια, το Web UI (το WEB UI ή Spark UI επιτρέπει τον χρήστη του Spark να παρακολουθεί τις εκτελέσεις εργασίας, τους χρόνους εργασίας και τα προσωρινά αποθηκευμένα δεδομένα. Είναι χρήσιμο εργαλείο για εκσφαλμάτωση).

Τα μη δομημένα APIs τα οποία είναι APIs χαμηλού επιπέδου και περιλαμβάνουν τα RDD (Resilient Distributed Dataset - ανθεκτικό κατανεμημένο σύνολο δεδομένων) και το API των "κατανεμημένων κοινών μεταβλητών" (Distributed Shared Variables) το οποίο αποτελείται από τους συσσωρευτές (accumulators) και τις μεταβλητές εκπομπής (broadcast variables).

- Το **RDD** είναι μια αμετάβλητη, ανεκτική σε σφάλματα, θεμελιώδη λογική συλλογή δεδομένων που κατανέμεται σε πολλούς υπολογιστικούς κόμβους που μπορούν να χρησιμοποιηθούν παράλληλα με μια χαμηλού επιπέδου προγραμματιστική διεπαφή (API) που προσφέρει μετασχηματισμούς (transformations) και δράσεις (actions) [22]. Στον παραπάνω ορισμό η διατύπωση ανεκτική σε σφάλματα (fault tolerant) σημαίνει ότι μπορεί να ανακάμψει αυτόματα από αποτυχίες, ενώ αμετάβλητη (immutable) σημαίνει ότι όταν δημιουργηθεί ένα RDD, δεν μπορούμε να το αλλάξουμε. Μετά την εφαρμογή ενός μετασχηματισμού σε ένα RDD προκύπτει ένα νέο RDD. Είναι μια δομή δεδομένων "χωρίς σχήμα" (schema-less), που μπορεί να χειριστεί τόσο δομημένα όσο και μη δομημένα δεδομένα. Η κοινή χρήση δεδομένων στη μνήμη κάνει τα RDD 10-100x ταχύτερα από την κοινή χρήση δίσκων και δικτύων. Αναλυτική αναφορά για τα RDD γίνεται στο 5^{ο} κεφάλαιο.
- Οι **μεταβλητές εκπομπής** (broadcast variables) είναι μεταβλητές που δημιουργεί ο χρήστης στο πρόγραμμά του ρητά και που είναι χρήσιμες σε έργα που εκτελούνται σε πολλαπλά στάδια και που χρειάζονται τα ίδια δεδομένα ή όταν η αποθήκευση στη μνήμη δεδομένων σε αποσειριοποιημένη (deserialised) μορφή είναι σημαντική [33]. Η μεταβλητή αυτή είναι μεταβλητή "μόνο ανάγνωσης" (read only) και διατηρείται

[8] *Έργο / Task: η μικρότερη δυνατή υπολογιστική εργασία στο Spark που ανατίθεται σε μία κατάτμηση*

προσωρινά αποθηκευμένη στη μνήμη κάθε κόμβου εργάτη αντί να αποστέλλεται ένα αντίγραφο αυτής με κάθε έργο[9].

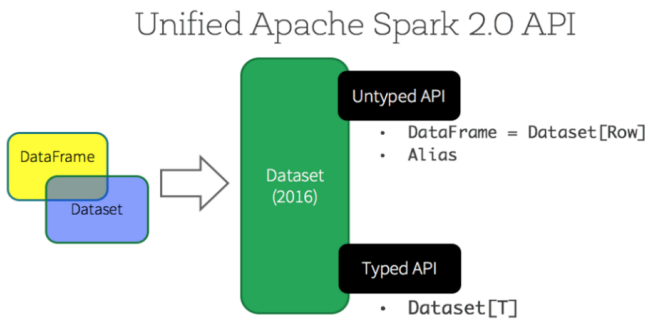
- **Οι συσσωρευτές** (accumulators) είναι μεταβλητές που "προστίθενται" σε μια προσεταιριστική και αντιμεταθετική πράξη "add" και για το λόγο αυτό είναι αποτελεσματικοί σε παράλληλες πράξεις [33]. Χρησιμοποιούνται για τη συσσώρευση των μερικών αποτελεσμάτων σε πολλαπλά έργα (που εκτελούνται στους εκτελεστές) και την προώθηση των αποτελεσμάτων στον κόμβο του οδηγού. Είναι σχεδιασμένοι για να χρησιμοποιούνται με ασφάλεια και αποτελεσματικότητα σε παράλληλους και κατανεμημένους υπολογισμούς Spark και προορίζονται για κατανεμημένους μετρητές και αθροιστές. Υπάρχουν δυο ειδών συσσωρευτές, αυτοί που ο χρήστης τους έχει ορίσει ένα όνομα (named) και αυτοί που δεν έχουν όνομα (unnamed). Τους πρώτους μπορούμε να τους παρακολουθήσουμε μέσα από το Spark UI (Web UI).

Τα δομημένα API αποτελούνται από τα DataFrames και τα DataSet. Είναι σχεδιασμένα και βελτιστοποιημένα για να λειτουργούν με δομημένα δεδομένα.

- Το DataFrame εισήχθη με την έκδοση 1.3 του Spark και είναι μια αφαίρεση, που είναι δομημένη πάνω από το RDD (απολαμβάνει όλα τα χαρακτηριστικά του), για δομημένα σύνολα δεδομένων στα οποία κάθε εγγραφή είναι μια σειρά αποτελούμενη από ένα σύνολο στηλών, και κάθε στήλη έχει έναν καλά καθορισμένο τύπο δεδομένων. Η έννοια των DataFrame είναι γνωστή από τις γλώσσες Python και R όπου και αναφέρεται σε πινακοποιημένα δεδομένα με προγραμματιστικές μεθόδους για φιλτράρισμα, υπολογισμό νέων στηλών και συνάθροιση. Στο Spark τα δεδομένα αυτά είναι κατανεμημένα και οι υπολογισμοί σε αυτά γίνονται μέσω της μηχανής Spark SQL απολαμβάνοντας όλες τις βελτιστοποιήσεις που της παρέχει ο βελτιστοποιητής της Catalyst Optimizer. Αναλυτική αναφορά για τα DataFrames γίνεται στο 5ο κεφάλαιο.
- Το Dataset αφορά μόνο τις γλώσσες Java και Scala συνεπώς δε θα γίνει περαιτέρω αναφορά σ' αυτό. Τα πλεονεκτήματά του όμως μπορεί να δελεάσουν τον προσανατολισμένο στην Python χρήστη του Spark να πειραματισθεί στο μέλλον και με τη γλώσσα Scala. Είναι μια κατανεμημένη συλλογή αντικειμένων JVM που παρέχει ασφάλεια τύπων (type safe) σε αντίθεση με το DataFrame. Σε αντίθεση με τα RDD που χρησιμοποιούν Java σειριοποίηση για σειριοποίηση αντικειμένων τα DataSet χρησιμοποιούν τον μηχανισμό σειριοποίησης του Encoder που επιτρέπει στο Spark να

[9] *Θα πρέπει να γίνει αποσειριοποίησή (deserialized) της στον εργάτη κάθε φορά που αποστέλλεται με ένα έργο (task).*

εκτελεί διάφορες πράξεις όπως φιλτράρισμα, ταξινόμηση, hashing χωρίς αποσειριοποίηση όπως στους κανονικούς μηχανισμούς σειριοποίησης. Προσφέρει τη δυνατότητα πράξεων πάνω σε κλάσεις οριζόμενες από τον χρήστη (user defined). Έτσι για παράδειγμα αφού ορίσει ο χρήστης του Spark τον δικό του τύπο δεδομένων και εκτελέσει τους χειρισμούς του, το Spark μπορεί αυτόματα να το μετατρέψει σε DataFrame και να το χειριστεί ο χρήστης περαιτέρω χρησιμοποιώντας τις εκατοντάδες συναρτήσεις που περιλαμβάνει το Spark. Τα δυο APIs έχουν ενοποιηθεί στην έκδοση 2.0 του Spark στη Scala και στη Java και το DataFrame θεωρείται ψευδώνυμο του Dataset [Row] στη Scala (βλέπε Σχήμα 3.4) [25].



(α)

Typed and Un-typed APIs

Language	Main Abstraction
Scala	Dataset[T] & DataFrame (alias for Dataset[Row])
Java	Dataset[T]
Python*	DataFrame
R*	DataFrame

Note: *Since Python and R have no compile-time type-safety, we only have untyped APIs, namely DataFrames.*

(β)

Πηγή: [25]

Σχήμα 3-4: (α) Ενοποιημένο Dataset API στο Spark 2.0 για Scala και Java, (β) Η Python και η R υποστηρίζουν μόνο το DataFrame

Spark SQL

Το Spark SQL ασχολείται κυρίως με την διερεύνηση, ανάλυση και επεξεργασία δομημένων δεδομένων. Η βασική ιδέα είναι να αντληθούν όσο το δυνατόν περισσότερες πληροφορίες σχετικά με τη δομή των δεδομένων προκειμένου να αξιοποιηθούν από τον βελτιστοποιητή Catalyst optimizer για την εκτέλεση των βελτιστοποιήσεων που γίνονται συνήθως σε πολλές μηχανές αναλυτικής βάσης δεδομένων. Για παράδειγμα σε πολλές περιπτώσεις το φιλτράρισμα και το κλάδεμα στήλης μπορεί να ωθηθεί μέχρι την πηγή δεδομένων

(Predicate/filter push-down and column pruning/column projection) όπως και το φιλτράρισμα και το κλάδεμα κατάτμησης (Partition Pruning and Predicate Pushdown).

Μπορεί να θεωρηθεί ως κατανεμημένη μηχανή επερωτήσεων SQL. Μπορεί ο χρήστης του Spark να εκτελέσει ερωτήματα σε δομημένα δεδομένα χρησιμοποιώντας είτε τη δομημένη γλώσσα ερωτημάτων SQL είτε το DataFrame API (μέσω των αντίστοιχων API για Scala, Java, Python, R). Ενεργώντας ως κατανεμημένη μηχανή επερωτήσεων, επιτρέπει τα ερωτήματα Hive να τρέχουν έως και 100 φορές γρηγορότερα σε αυτό χωρίς καμία τροποποίηση.

Ένα σημαντικό χαρακτηριστικό είναι η ενοποιημένη πρόσβαση δεδομένων, μέσω του Data Sources API. Μπορεί ο χρήστης να γράψει και να διαβάσει δεδομένα σε διάφορα μορφότυπα (formats) και συστήματα αποθήκευσης (json, csv, parquet, orc, Avro, Hive, διάφορες πηγές RDBMS, κ.λπ.).

Spark Streaming και Structured Streaming

Το Spark μας παρέχει δύο τρόπους για να δουλέψουμε με δεδομένα ροής :

- Το Spark Streaming
- Το Structured Streaming (από το Spark 2.x)

Spark Streaming

Οι επιχειρήσεις που επεξεργάζονται δεδομένα κατά την άφιξή τους αποκτούν ένα σημαντικό πλεονέκτημα έναντι των ανταγωνιστών τους. Η βιβλιοθήκη επεξεργασίας συνεχούς ροής του Spark, η Spark Streaming, δίνει στον χρήστη τη δυνατότητα επεξεργασίας δεδομένων ροής σε πραγματικό χρόνο από διάφορες πηγές δεδομένων με υψηλή ρυθμαπόδοση (throughput) και ανεκτικότητα σε σφάλματα. Παρέχει το Descritized Stream API ή, εν συντομία Dstream API, το οποίο είναι δομημένο πάνω στο RDD και αντιπροσωπεύει μια ροή δεδομένων χωρισμένη σε μικροδεσμίδες (micro batches).

Μπορεί να καταναλώνει συνεχώς δεδομένα από πηγές όπως το Apache Flume, το Apache Kafka, το Amazon Kinesis Data Streams, ZeroMQ, Twitter, HDFS ή υποδοχές (sockets) TCP τα οποία ακολούθως μπορούν να επεξεργαστούν με συναρτήσεις υψηλού επιπέδου όπως map, reduce, join και window. Τέλος, τα επεξεργασμένα δεδομένα μπορούν να ωθηθούν σε συστήματα αρχείων, βάσεων δεδομένων, και πίνακες ελέγχου πραγματικού χρόνου.

Το Spark Streaming επιτρέπει ροές εργασίας όπως ανάλυση δεδομένων, μηχανική μάθηση και επεξεργασία γράφων σε ροές πραγματικού χρόνου.

Πηγή: https://spark.apache.org/docs/2.1.1/streaming-programming-guide.html

Σχήμα 3-5: Spark Streaming

Structured Streaming

Μια νέα κλιμακώσιμη μηχανή επεξεργασίας συνεχούς ροής που ονομάζεται structured streaming (δομημένη ροή) και παρέχει ένα υψηλότερο επίπεδο αφαίρεσης από την αρχική Spark Streaming παρουσιάστηκε στο Spark 2.1 και δομήθηκε πάνω από τη μηχανή Spark SQL. Αυτό το μοντέλο ροής βασίζεται στο ενοποιημένο πλέον Dataframe και Dataset API. Ως εκ τούτου, με αυτήν τη βιβλιοθήκη, μπορεί κανείς εύκολα να εφαρμόσει οποιοδήποτε ερώτημα SQL (χρησιμοποιώντας το DataFrame API) ή τις συναρτήσεις scala (χρησιμοποιώντας το DataSet API) σε δεδομένα συνεχούς ροής.

Spark MLlib

Το Spark παρέχει την MLlib, μια κατανεμημένη βιβλιοθήκη μηχανικής μάθησης που βασίζεται σε δυο API, το DataFrame API (πακέτο spark.ml) και το RDD API (πακέτο spark.mllib). Από την έκδοση 2.0 του Spark το RDD API βρίσκεται σε φάση συντήρησης και αναμένεται να καταργηθεί στο Spark 3.0. Η μετάβαση αυτή έγινε γιατί τα DataFrame παρέχουν ένα φιλικότερο προς το χρήστη API από τα RDD. Στη συνέχεια κάθε αναφορά στη βιβλιοθήκη Mllib θα αφορά το DataFrame API που παρέχει ένα ομοιόμορφο API σε όλους τους αλγόριθμους ML και σε όλες τις γλώσσες που υποστηρίζει (Scala, Java, Python, R). Η Mllib αποτελείται από γρήγορες και κλιμακώσιμες υλοποιήσεις τυποποιημένων αλγορίθμων μηχανικής μάθησης συμπεριλαμβανομένων των περισσοτέρων αλγορίθμων ταξινόμησης, παλινδρόμησης, συνεργατικού φιλτραρίσματος, συσταδοποίησης και μείωσης διαστασιμότητας (dimensionality reduction).

Παρέχει αφαιρέσεις για τη διαχείριση και απλοποίηση εργασιών που σχετίζονται με τα μοντέλα μηχανικής μάθησης, όπως επιλογή χαρακτηριστικών, μετασχηματισμό, συνδυασμό και εξαγωγή νέων χαρακτηριστικών (featurization) από δομημένα σύνολα δεδομένων, εργαλεία για την κατασκευή, αξιολόγηση και παραμετροποίηση σωληνώσεων[10](pipelining) ML (μηχανικής μάθησης) και την επιμονή (persistence) των μοντέλων.

Παρέχει επίσης μια ποικιλία βοηθητικών προγραμμάτων για στατιστική, κατανεμημένη γραμμική άλγεβρα, και απλές βελτιστοποιήσεις.

Spark Graphx και GraphFrames

Η επεξεργασία γράφων λειτουργεί σε δομές δεδομένων που αποτελούνται από κορυφές (κόμβους) και ακμές συνδέοντάς τες. Το Spark GraphX είναι ένα στοιχείο που επιτρέπει τον υπολογισμό παράλληλων γράφων παρέχοντας μια αφαίρεση ενός κατευθυνόμενου πολυγράφου/πολλαπλού γράφου (γράφου με παράλληλες ακμές) με ιδιότητες προσαρτημένες σε κάθε κορυφή και ακμή. Το GraphX είναι μια βιβλιοθήκη βασισμένη στο RDD API και περιλαμβάνει μια συλλογή από κοινούς αλγόριθμους επεξεργασίας γράφων. Ή ισχυρή αυτή βιβλιοθήκη όπως και τα RDD έχει μια διασύνδεση χαμηλού επιπέδου δύσκολη στη χρήση και στη βελτιστοποίηση. Επίσης έχει έναν επιπλέον περιορισμό, λειτουργεί μόνο σε Scala. Πλέον υπάρχει η επόμενης γενεάς βιβλιοθήκη αναλυτικής γράφων η GraphFrames, που επεκτείνει τη λειτουργικότητα της GraphX. Βασίζεται στο DataFrame API και η εκτεταμένη λειτουργικότητα που παρέχει περιλαμβάνει την εύρεση μοτίβων, σειριοποίηση (serialization) με βάση το DataFrame και υψηλής εκφραστικότητας ερωτήματα γράφου. Ο χρήστης επωφελείται επίσης από τις βελτιστοποιήσεις απόδοσης του DataFrame μέσα στην κατανεμημένη μηχανή Spark SQL και μπορεί επίσης να αποθηκεύει και να φορτώνει γραφήματα χρησιμοποιώντας μορφότυπα όπως Parquet, JSON και CSV. Το GraphFrames δεν έχει συγχωνευθεί ακόμα στον πυρήνα του Spark και είναι επί του παρόντος διαθέσιμο ως ένα εξωτερικό πακέτο Spark που πρέπει να το φορτώσει ο χρήστης του Spark όταν ξεκινάει την εφαρμογή Spark.

[10] *Μια σωλήνωση (pipeline) είναι μια ακολουθία από μετασχηματισμούς όπου η έξοδος ενός σταδίου είναι είσοδος ενός άλλου, σχηματίζοντας μια αλυσίδα.*

Κεφάλαιο 4

Εκτέλεση ενός προγράμματος Spark

Υπάρχουν δύο μέθοδοι με τις οποίες μπορεί να τρέξει κανείς ένα πρόγραμμα Spark:

- Με Διαδραστικούς Πελάτες (Pyspark, Jupyter Notebook, κ.λπ.)
- Υποβάλλοντας μια εφαρμογή με το spark-submit[11].

Κατά την διαδικασία της εγκατάστασης του Spark, στο κεφάλαιο 6, η πρόσβαση στο Spark πραγματοποιείται χρησιμοποιώντας το Pyspark Shell και το Jupyter notebook. Και οι δύο εφαρμογές είναι διαδραστικοί πελάτες (interactive clients). Οι διαδραστικοί πελάτες χρησιμοποιούνται ευρέως τόσο κατά τη διάρκεια εκμάθησης του Spark όσο και κατά την εξερεύνηση και την διαδικασία ανάπτυξης. Στο τέλος, όλη η διαδραστική εξερεύνησή του προγραμματιστή θα καταλήξει σε μια πλήρη εφαρμογή Spark που θα μπορεί να χρησιμοποιηθεί για παραγωγή. Στη περίπτωση αυτή παίρνει ο προγραμματιστής τον κώδικα της εφαρμογής του που θα χρησιμοποιηθεί για παραγωγή και τον υποβάλλει για εκτέλεση με την μέθοδο *spark-submit*.

Τρόποι ανάπτυξης του Spark

Ο τρόπος ανάπτυξης μιας εφαρμογής Spark καθορίζει τη θέση όπου θα εκτελείται ο οδηγός στο περιβάλλον ανάπτυξης.

Υπάρχουν οι ακόλουθοι δύο τρόποι ανάπτυξης του Spark:

- Ο τρόπος ανάπτυξης πελάτη (client deploy mode), όπου το πρόγραμμα οδήγησης βρίσκεται σε μια μηχανή πελάτη εξωτερικά της συστάδας και
- Ο τρόπος ανάπτυξης συστάδας (cluster deploy mode), όπου το πρόγραμμα οδήγησης βρίσκεται μέσα στην ίδια τη συστάδα.

Ο πρώτος τρόπος ανάπτυξης ενδείκνυται για την ανάπτυξη εφαρμογών, ενώ ο δεύτερος τρόπος ανάπτυξης ενδείκνυται για παραγωγή [9].

Σε τρόπο ανάπτυξης πελάτη, το Python πρόγραμμα του χρήστη (δηλ. ο οδηγός) θα εκτελεστεί στον ίδιο υπολογιστή όπου εκτελείται η spark-submit. Είναι προς το συμφέρον του χρήστη να βεβαιωθεί ότι ο συγκεκριμένος υπολογιστής βρίσκεται κοντά στους κόμβους των εργατών του

[11] *Βοηθητικό πρόγραμμα υποβολής εφαρμογών Spark*

για να μειωθεί η καθυστέρηση του δικτύου. Αν ξεκινήσει ο χρήστης το πρόγραμμα οδήγησης στο τοπικό του μηχάνημα, η εφαρμογή του θα εξαρτάται άμεσα από αυτό. Αυτό σημαίνει ότι αν πάει κάτι στραβά στον υπολογιστή του αυτό θα έχει επίπτωση και στον οδηγό του που θα χάσει όλες τις απαραίτητες πληροφορίες της εφαρμογής (κρέμασμα της εφαρμογής). Είναι σίγουρο πως δεν επιθυμεί κανείς μια τέτοιου είδους εξάρτηση σε μια εφαρμογή παραγωγής. Επομένως, ο τρόπος ανάπτυξης συστάδας έχει νόημα για εφαρμογές παραγωγής καθώς ο οδηγός ξεκινά στη συστάδα. Επίσης μετά την υποβολή της εφαρμογής Spark, μπορεί να απενεργοποιήσει ο χρήστης τον τοπικό υπολογιστή του και η εφαρμογή του θα συνεχίσει να εκτελείται ανεξάρτητα μέσα στη συστάδα.

Στον τρόπο ανάπτυξης πελάτη η είσοδος και η έξοδος της εφαρμογής προσαρτώνται στον πίνακα ελέγχου Spark UI. Έτσι, αυτός ο τρόπος λειτουργίας είναι ιδιαίτερα κατάλληλος για εφαρμογές που εκτελούνται από διαδραστικό πελάτη (π.χ. κέλυφος (shell) pypark, Jupyter notebook, κ.λπ), για την εξερεύνηση, την ανάπτυξη εφαρμογών ή για τον εντοπισμό σφαλμάτων σε μια εφαρμογή [32] μπορεί όμως κάλλιστα να χρησιμοποιηθεί και για υποβολή μη διαδραστικών εφαρμογών με το spark-submit με όλα όμως τα μειονεκτήματα βέβαια που περιγράφονται παραπάνω. Αν ο οδηγός είναι απομακρυσμένος από τη συστάδα θα υπάρχουν καθυστερήσεις κατά την εκτέλεση της εφαρμογής για το λόγο αυτό συνιστάται ο χρήστης να υποβάλει την εφαρμογή του από έναν κόμβο πύλης (gateway / edge node[12]) που βρίσκεται σε φυσική εγγύτητα με τους κόμβους (μηχανές) των εργατών του (π.χ. master κόμβος σε αυτόνομη συστάδα EC2) [31].

Οι εργασίες σε μεγάλα δεδομένα μπορούν να εκτελεστούν σε διάφορους διαχειριστές συστάδας όπως Hadoop YARN, Mesos, αυτόνομη συστάδα Spark (Spark StandAlone mode) και Kubernetes είτε σε τοπική συστάδα είτε στο με πρόσβαση σε διάφορα συστήματα αποθήκευσης όπως:

- αποθηκευτικά νέφη (cloud storage systems) όπως το Amazon S3 και το Azure Blob Storage,
- κατανεμημένα συστήματα αρχείων όπως το HDFS-Hadoop, το IBM GPFS (General Parallel File System) και το Google File System.
- Key-value stores, όπως το Apache Cassandra.
- Κανάλια μηνυμάτων (message buses), όπως το Kafka.

12 Ένας ακραίος κόμβος (edge node) είναι ένας υπολογιστής που λειτουργεί ως πύλη τελικού χρήστη για επικοινωνία με άλλους κόμβους σε υπολογιστικές συστάδες. Οι ακραίοι κόμβοι ονομάζονται επίσης και κόμβοι πύλης (gateway).

Τρέξιμο της εφαρμογής Spark τοπικά (local mode)

Το Spark μπορεί να εκτελείται τοπικά σε ένα μόνο μηχάνημα με μία μόνο εικονική μηχανή Java - JVM - (τοπική λειτουργία – local mode) όπου ο οδηγός (driver) και οι εκτελεστές (executors) τρέχουν ως νήματα, αυτή η δυνατότητα επιτρέπει την εύκολη ανάπτυξη των εφαρμογών του χρήστη σε ένα προσωπικό υπολογιστή.

Παραδείγματα ανάπτυξης εφαρμογών με το spark-submit

Μπορεί να καθορίσει ο χρήστης τον τρόπο ανάπτυξης μιας εφαρμογής Spark χρησιμοποιώντας την επιλογή --deploy-mode στο βοηθητικό πρόγραμμα υποβολής *spark-submit* στη γραμμή εντολών. Μια ακόμη επιλογή που συνδυάζεται με την παραπάνω είναι η *--master* με την οποία προσδιορίζει ο χρήστης τη διεύθυνση URL του master (local, spark standalone, mesos, yarn, kubernetes), όπως φαίνεται στο παρακάτω παράδειγμα:

```
# Εκτέλεση μιας εφαρμογής Python με προεπιλεγμένο τρόπο ανάπτυξης σε συστάδα Spark
# standalone cluster
./bin/spark-submit \
  --master spark://192.168.1.100:7077 \
  /opt/spark/examples/src/main/python/pi.py \
  100
```

Όπου στο παραπάνω παράδειγμα:

- **--master** [13] : Η διεύθυνση URL του master για τη συστάδα (π.χ. *spark: //192.168.1.100:707,* συνδέεται ο χρήστης με τον master της δοθείσας αυτόνομης συστάδας Spark).
- **--deploy-mode**: cluster ή client

  cluster[14]: Αν πρόκειται να αναπτύξει ο χρήστης το πρόγραμμα οδήγησης στους κόμβους των εργατών (worker nodes) της συστάδας.

  client: Αν πρόκειται να αναπτύξει ο χρήστης το πρόγραμμα οδήγησης τοπικά ως εξωτερικός πελάτης (client) (προεπιλογή: client, γι' αυτό και στο ανωτέρω παράδειγμα δεν φαίνεται η επιλογή αυτή).

Για να τρέξει την εφαρμογή ο χρήστης τοπικά (local mode) μπορεί για παράδειγμα να χρησιμοποιήσει την ακόλουθη master διεύθυνση URL:

```
# Εκτέλεση εφαρμογής Spark τοπικά (local mode) με 8 πυρήνες (λογικούς)
./bin/spark-submit \
  /opt/spark/examples/src/main/python/pi.py \
```

[13] *Σε μια συστάδα YARN η διεύθυνση του ResourceManager (master) λαμβάνεται από τη διαμόρφωση Hadoop. Για το λόγο αυτό στην παράμετρο --master τίθεται η τιμή yarn (βλέπε παραδείγματα ανάπτυξης εφαρμογών Spark σε συστάδα YARN στην ενότητα "Τροποποιήσεις και τελικές ρυθμίσεις στους κόμβους" του 6ου Κεφαλαίου).*

[14] *Το Spark δεν υποστηρίζει προς το παρόν σε standalone mode την επιλογή cluster (mode) για εφαρμογές Python.*

```
--master local[8] \
100
```

Όπου:

- **--master** local[8]: Επιλογή για εκτέλεση του Spark τοπικά με 8 νήματα εργατών (ιδανικά ο χρήστης θα πρέπει να ρυθμίσει αυτόν τον αριθμό πυρήνων σύμφωνα με το μηχάνημά του).

Σημείωση: Για εφαρμογές Python, η εντολή spark-submit μπορεί να φορτώσει και να επεξεργαστεί όλες τις εξαρτήσεις που παρέχει ο χρήστης ως αρχεία .py, .zip ή .egg με την παράμετρο --py-files της spark-submit.

Για περισσότερες πληροφορίες σχετικά με την υποβολή μιας εφαρμογής με την spark-submit θα πρέπει να ανατρέξει ο χρήστης στην επίσημη ιστοσελίδα του Spark [31] όπου μπορεί να δει το σύνολο των master URLs (για local, spark standalone, mesos, yarn, kubernetes) καθώς και τις υπόλοιπες επιλογές της spark-submit.

Αρχιτεκτονική μιας εφαρμογής Spark

Η εφαρμογή Spark (Spark application) είναι η υψηλότερου επιπέδου μονάδα υπολογισμού (unit of computation) στο Spark σε αντίθεση με το MapReduce που η υψηλότερου επιπέδου μονάδα υπολογισμού είναι η εργασία (job). Μια εφαρμογή μπορεί να αποτελείται από μια ή περισσότερες εργασίες (jobs) για επεξεργασία κατά δεσμίδες (batch processing), μπορεί να είναι μια αλληλεπιδραστική συνεδρία (interactive session) αποτελούμενη από πολλές εργασίες ή μπορεί να είναι ένας διακομιστής που είναι αφιερωμένος αποκλειστικά στην εξυπηρέτηση αιτημάτων (queries). Οι εφαρμογές Spark λειτουργούν ως ανεξάρτητα σύνολα διεργασιών σε μια συστάδα. Το Spark ως κατανεμημένη μηχανή επεξεργασίας, ακολουθεί την αρχιτεκτονική master-slave (αφέντη/σκλάβου). Για **κάθε εφαρμογή το Spark**, δημιουργεί μια διεργασία master και μια σειρά από διεργασίες slave. Σύμφωνα με την ορολογία του Spark η διεργασία master είναι η διεργασία οδηγού (driver) που ονομάζεται SparkSession [1] και οι διεργασίες slave είναι οι διεργασίες εκτελεστή (executor).

Πρόγραμμα οδήγησης Spark

Ο οδηγός (driver) είναι η master διεργασία. Είναι αυτός που θα διασυνδεθεί με τον διαχειριστή συστάδας μέσω του αντικειμένου *SparkContext* που δημιουργείται εντός του οδηγού και θα αιτηθεί να του εκχωρηθούν φυσικοί πόροι στους οποίους θα ξεκινήσει τους εκτελεστές. Πρώτα μετατρέπει το πρόγραμμα χρήστη σε μια σειρά από έργα (η μικρότερη μονάδα υπολογιστικής εργασίας) και μετά προγραμματίζει τα έργα στους εκτελεστές και παρακολουθεί την εκτέλεσή τους. Ο οδηγός είναι επίσης υπεύθυνος για τη διατήρηση όλων των απαραίτητων πληροφοριών

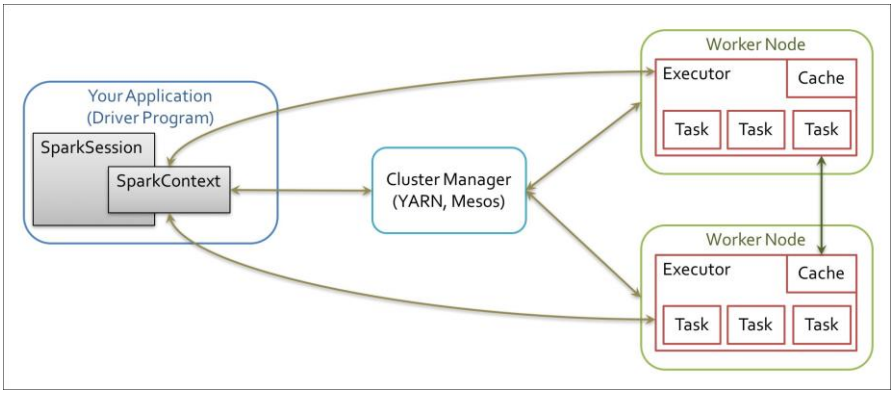
κατά τη διάρκεια της ζωής της εφαρμογής. Η φυσική θέση του προγράμματος οδήγησης μπορεί να είναι σε έναν πελάτη (client) ή σε έναν κόμβο της συστάδας, όπως αναφέρθηκε προηγουμένως.

Εκτελεστές Spark

Οι εκτελεστές Spark είναι διεργασίες στους κόμβους εργατών (worker nodes) και είναι υπεύθυνοι μόνο για την εκτέλεση του κώδικα που τους έχει εκχωρηθεί από τον οδηγό και για να αναφέρουν την κατάστασή τους πίσω στον οδηγό.

Το παρακάτω σχήμα δείχνει μια εφαρμογή Spark που τρέχει σε μια συστάδα με τρόπο ανάπτυξης πελάτη (client deploy mode – βλέπε τρόποι ανάπτυξης του Spark). Η απεικόνιση είναι ανεξάρτητη της υλοποίησης, δηλαδή ανεξάρτητα από τον διαχειριστή συστάδας και τον κώδικα της εφαρμογής.

- Χρησιμοποιώντας το spark-submit, ο πελάτης (client) υποβάλλει μια εφαρμογή.
- Η εφαρμογή εκκινεί δημιουργώντας το SparkSession ως μια δομή δεδομένων όπου ο οδηγός διατηρεί όλες τις πληροφορίες, συμπεριλαμβανομένης της θέσης των εκτελεστών και της κατάστασής τους και το πρόγραμμα οδήγησης ζητά ακολούθως την εκχώρηση πόρων (σύμφωνα με τα ορίσματα της spark-submit – π.χ. αριθμός εκτελεστών, μνήμη ανά εκτελεστή, κ.λπ.) για την εκκίνηση των εκτελεστών.
- Ο διαχειριστής συστάδας αφού εκκινήσει τους εκτελεστές στέλνει πληροφορίες σχετικά με τη τοποθεσία τους στον οδηγό. Με την ολοκλήρωση αυτού του βήματος έχει δημιουργηθεί μια συστάδα Spark.
- Ο οδηγός και οι εργάτες (workers) μπορούν πλέον να επικοινωνήσουν απευθείας μεταξύ τους, εκτελώντας κώδικα και μετακινώντας δεδομένα. Ο οδηγός και συγκεκριμένα το αντικείμενο SparkSession προγραμματίζει τα έργα για κάθε εργάτη και κάθε εργάτης αποκρίνεται με την κατάσταση αυτών των έργων καθώς και την επιτυχία ή αποτυχία ολοκλήρωσής τους. Τα αποτελέσματα είτε αποστέλλονται πίσω στην εφαρμογή του προγράμματος οδήγησης είτε αποθηκεύονται στον τοπικό δίσκο του εργάτη.
- Όταν εκτελεστεί η εντολή sparkSession.stop() (που σταματά το αντικείμενο SparkContext) στο πρόγραμμα οδήγησης ή όταν η κύρια μέθοδος τερματιστεί ή καταρρεύσει, όλοι οι εκτελεστές θα τερματιστούν και οι πόροι της συστάδας θα αποδεσμευτούν από τον διαχειριστή συστάδας.



Πηγή: [37]

https://blogs.perficient.com/2015/06/04/how-to-configure-eclipse-for-spark-application-in-the-cluster/

Σχήμα 4-1: Η αρχιτεκτονική του Spark

Τρόποι ανάπτυξης μιας εφαρμογής Spark στο Hadoop YARN

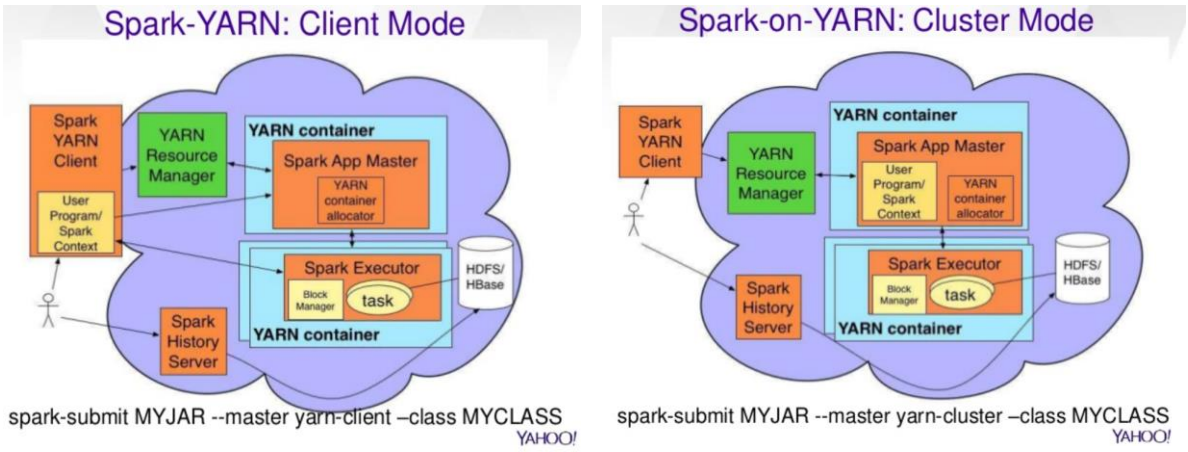
Η θέση του οδηγού Spark σε σχέση με τον πελάτη και τον Application Master ορίζει τον τρόπο ανάπτυξης μιας εφαρμογή Spark σε μια συστάδα YARN (βλέπε Σχήμα 4-2) :

- Όταν ο οδηγός τρέχει στον πελάτη ορίζεται ως λειτουργία πελάτη YARN (Yarn client mode) και ο πελάτης πρέπει να είναι ενεργός καθ' όλη τη διάρκεια ζωής της εφαρμογής.
- Όταν ο οδηγός Spark τρέχει μέσα στον Application Master ορίζεται ως λειτουργία συστάδας YARN (Yarn cluster mode). Στην περίπτωση αυτή μια ενιαία διεργασία σε ένα container YARN είναι υπεύθυνη τόσο για την οδήγηση της εφαρμογής όσο και για την υποβολή αιτημάτων χορήγησης πόρων στο Διαχειριστή Πόρων YARN.

Ανάπτυξη σε YARN client mode έναντι YARN cluster mode

- Στο Yarn όταν ξεκινά μια εφαρμογή μέσω του SparkContext ζητά από τον Διαχειριστή Πόρων την εκχώρηση πόρου (container) σε κάποιον slave κόμβο στο οποίο θα αναπτυχθεί ο Application Master (AM) της εφαρμογής (Σχήμα 4-2), ο οποίος θα είναι υπεύθυνος για την διαπραγμάτευση πόρων από τον Διαχειριστή Πόρων (RM). Αφού ο ResourceManager λάβει το αίτημα, επιλέγει ένα NodeManager στη συστάδα και του αναθέτει το πρώτο container για τον Application Master της εφαρμογής. Η διαφορά του YARN client mode από το YARN Cluster mode είναι ότι ο AM δεν εκτελεί το SparkContext μόνο επικοινωνεί με το SparkContext (Σχήμα 4-2α) για την κατανομή πόρων. Ο Οδηγός ενημερώνει τον Application Master για τις απαιτήσεις της εφαρμογής σε πόρους και ο Application Master διαπραγματεύεται με τον Διαχειριστή

Πόρων τους πόρους (containers) που θα φιλοξενήσουν τους εκτελεστές. Μόλις εκχωρηθούν οι πόροι ο Application Master επικοινωνεί με τους αντίστοιχους NodeManagers ζητώντας του να ξεκινήσουν τους εκτελεστές στα containers που έχουν διατεθεί στην εφαρμογή. Μόλις ξεκινήσουν οι εκτελεστές εγγράφονται στο SparkContext στον πελάτη και υποβάλουν αίτημα για ένα έργο[15]. Το SparkContext στον πελάτη αναθέτει στους εκτελεστές ένα έργο και αυτοί αναλαμβάνουν την εκτέλεσή του και αναφέρουν την τρέχουσα κατάσταση και την πρόοδό του στο πρόγραμμα οδήγησης έτσι ώστε ο πελάτης να μπορεί αν αποτύχει το έργο να το επανεκκινήσει σε κάποιον άλλον εκτελεστή. Μετά την ολοκλήρωση της εφαρμογής, το SparkContext του πελάτη[16] υποβάλλει αίτημα αποσύνδεσης στο ResourceManager και κατόπιν κλείνει το ίδιο.



Πηγή: https://www.slideshare.net/Hadoop_Summit/sparkonyarn-empower-spark-applications-on-hadoop-cluster

Σχήμα 4-2: Σύγκριση ανάπτυξης μιας εφαρμογής Spark σε client mode έναντι cluster mode σε συστάδα Hadoop YARN

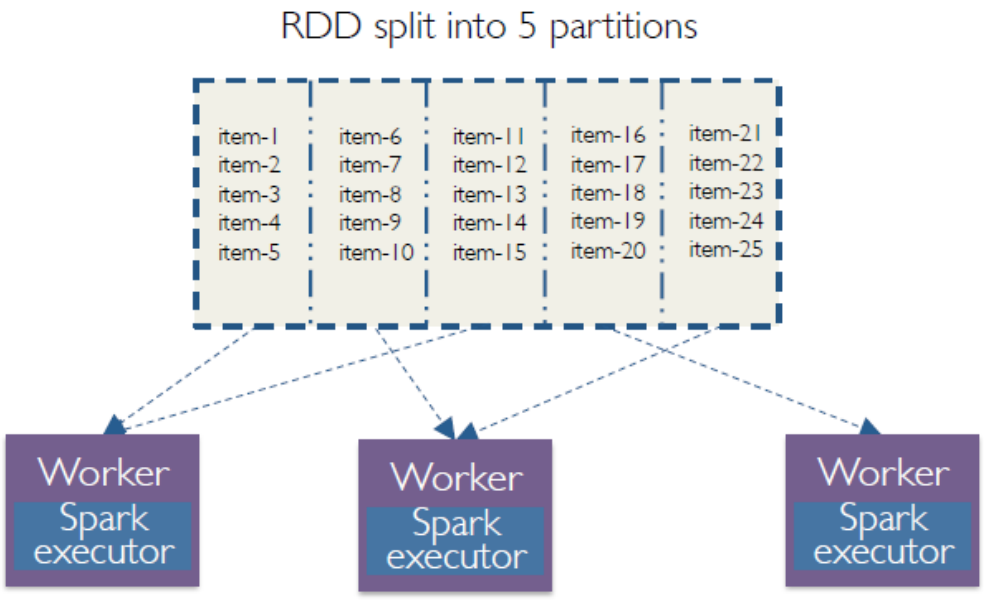
[15] *Ο αριθμός των έργων (tasks) εξαρτάται από τον αριθμό των πυρήνων του εκτελεστή. Ένας πυρήνας σημαίνει ότι μόνο ένα έργο μπορεί να εκτελείται ανά πάσα στιγμή στον εκτελεστή.*

[16] *Σε Cluster Mode το Spark Context στον Application Master.*

Κεφάλαιο 5
Οι δομές δεδομένων RDD και DataFrame

Κατατμήσεις

Πριν την αναλυτική αναφορά στα RDD θα προηγηθεί μια αναφορά στην κατάτμηση / διαμέριση (partition). Η κατάτμηση στα κατανεμημένα συστήματα είναι ένα λογικό κομμάτι ενός μεγάλου κατανεμημένου συνόλου δεδομένων. Κάθε κατάτμηση αποτελείται από εγγραφές. Το Spark λειτουργεί βάσει της αρχής της τοπικότητας των δεδομένων (data locality). Οι κόμβοι των εργατών (worker nodes) παίρνουν τα δεδομένα για επεξεργασία που είναι πιο κοντά σε αυτούς. Με τον τρόπο αυτό, πετυχαίνουμε μείωση των I / O δικτύου και κατά συνέπεια αύξηση της ταχύτητας επεξεργασίας των δεδομένων (βλέπε Σχήμα 5.1).



Πηγή: https://techmagie.wordpress.com/2015/12/19/understanding-spark-partitioning/

Σχήμα 5-1: Οπτικοποίηση της διαμέρισης ενός RDD - Καθένας από τους εργάτες θα επεξεργαστεί τα δεδομένα που βρίσκονται στις κατατμήσεις που είναι αποθηκευμένες σε αυτόν (Τοπικότητα Δεδομένων - Data Locality). Για παράδειγμα ο εργάτης στα αριστερά θα επεξεργαστεί τις κατατμήσεις 1 και 3 από αριστερά.

Οι κατατμήσεις είναι οι μονάδες παραλληλισμού. Από προεπιλογή, δημιουργείται μια κατάτμηση για κάθε HDFS block, το οποίο από προεπιλογή είναι 128MB.

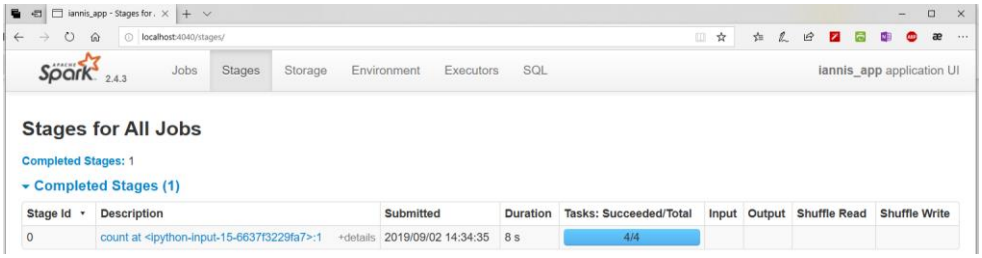
Κάθε κόμβος σε μια συστάδα Spark περιέχει μια ή περισσότερες κατατμήσεις (βλέπε Σχήμα 5.1). Το Spark εκχωρεί ένα έργο ανά κατάτμηση και κάθε εργάτης μπορεί να επεξεργαστεί ένα έργο κάθε φορά. Επομένως σε μια συστάδα ο μέγιστος αριθμός των έργων που μπορεί το Spark να εκτελέσει παράλληλα είναι ίσος με τον αριθμό πυρήνων τη συστάδας. Επομένως, εάν έχει κανείς μια συστάδα με **x** πυρήνες, θα πρέπει τα RDD του να έχουν τουλάχιστον **x** κατατμήσεις (και ίσως 2-3 **x**) [23]. Ο διαμερισμός του RDD και η διανομή των κατατμήσεων σε διαφορετικούς κόμβους γίνεται αυτόματα στο Spark. Όταν εκτελέσει ο προγραμματιστής την εργασία Spark, δηλ. *sc.parallelize (range (1,101)).count()*, στο παράδειγμα του Σχήματος 5.2[17] θα πρέπει να δει τα παρακάτω στο Web UI[18] (Σχήμα 5.3) του κελύφους Spark. Το Spark δημιουργεί 4 έργα για την εργασία (job) Spark, γιατί εξορισμού ο αριθμός των πυρήνων (local[*]) σε μια local mode ανάπτυξη (βλέπε παράγραφο 'τρόποι ανάπτυξης του Spark') είναι ίσος με τον αριθμό των λογικών πυρήνων[19] του υπολογιστή (στον υπολογιστή του γράφοντα οι λογικοί πυρήνες = 4).

Σχήμα 5-2: Παράδειγμα όπου το Spark δημιουργεί 4 έργα (tasks) για την εργασία (job) Spark, γιατί από προεπιλογή ο αριθμός των πυρήνων (local[*]) σε μια local mode ανάπτυξη είναι ίσος με τον αριθμό των λογικών πυρήνων του υπολογιστή (στον υπολογιστή του γράφοντα οι λογικοί πυρήνες = 4)

[17] *Στο εξής στα παραδείγματα μέσα στο jupyter δε θα φαίνονται τα εισαγωγικά βήματα δημιουργίας του στιγμιότυπου spark (1ο και 2ο κελί).*

[18] *Το web UI του Spark (επίσης αναφέρεται και ως Spark UI ή Application UI) επιτρέπει τον χρήστη να παρακολουθήσει την πρόοδο μιας εργασίας σε ένα πρόγραμμα περιήγησης ιστού. Είναι διαθέσιμο από προεπιλογή στη θύρα 4040 του κόμβου του οδηγού. Σε τοπική λειτουργία (local mode), είναι στη http://localhost:4040 ενώ όταν χρησιμοποιείται διαχειριστής συστάδας είναι στη http://<driver-node>:4040. Το Spark UI εμφανίζει πληροφορίες σχετικά με την κατάσταση των εργασιών Spark, του περιβάλλοντος και της κατάστασης της συστάδας αλλά εξορισμού είναι διαθέσιμες μόνο για όσο χρόνο διαρκεί η εφαρμογή. Είναι πολύ χρήσιμο, ειδικά για ρυθμίσεις βελτιστοποίησης της απόδοσης τόσο της εφαρμογής όσο και της συστάδας αλλά και για εκσφαλμάτωση.*

[19] *Ο αριθμός των λογικών πυρήνων είναι ίσος με τον αριθμό των φυσικών πυρήνων επί τον αριθμό των νημάτων που μπορούν να τρέξουν σε κάθε πυρήνα.*

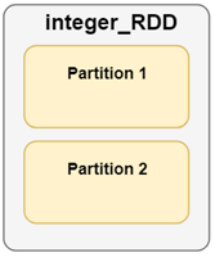


Σχήμα 5-3: Το Spark Web UI για το παράδειγμα του Σχήματος 5.2

Ωστόσο, κατά καιρούς οι προγραμματιστές μπορεί να επιθυμούν να αλλάξουν το σχήμα της κατάτμησης, αλλάζοντας το μέγεθος και τον αριθμό των κατατμήσεων βάσει των απαιτήσεων της εφαρμογής. Η τροποποίηση του αριθμού των κατατμήσεων ενός RDD γίνεται χρησιμοποιώντας τους μετασχηματισμούς repartition (κατάτμησης) ή coalesce (συγχώνευσης). Ο μετασχηματισμός repartition δημιουργεί νέες κατατμήσεις ανακατανέμοντας τα δεδομένα μεταξύ των εκτελεστών (είναι μια αρκετά κοστοβόρα πράξη γιατί κάνει μια πλήρη ανακατανομή) οι δε κατατμήσεις που δημιουργούνται είναι περίπου ισομεγέθεις. Ο μετασχηματισμός repartition() μπορεί να χρησιμοποιηθεί για να αυξήσει ή να μειώσει τον αριθμό των κατατμήσεων ενός RDD σε αντίθεση με τον μετασχηματισμό coalesce() όπου χρησιμοποιείται μόνο για μείωση των κατατμήσεων . Ο μετασχηματισμός coalesce() θα μπορούσε να θεωρηθεί ως μια βελτιστοποιημένη έκδοση του μετασχηματισμού repartition() που αποφεύγει την πλήρη ανακατανομή (full shuffle) των δεδομένων, όταν είναι γνωστό εκ των προτέρων ότι μειώνεται ο αριθμός των κατατμήσεων RDD μετακινώντας μόνο τα δεδομένα από τις επιπλέον κατατμήσεις στις κατατμήσεις που διατηρούνται. Αυτός ο αλγόριθμος είναι γρήγορος σε ορισμένες περιπτώσεις, επειδή ελαχιστοποιεί την μετακίνηση δεδομένων.

Στο Σχήμα 6.4 δίδεται ένα απλό παράδειγμα σε γλώσσα Python (μέσω του PySpark API) όπου δημιουργείται μια λίστα με 10 ακέραιους αριθμούς με 2 κατατμήσεις (αντί για τις 4 από προεπιλογή του προηγούμενου παραδείγματος):

```
//Python
// Το sc.parallelize() είναι μετασχηματισμός
// Το πρώτο όρισμα είναι η συλλογή αντικειμένων που παραλληλίζονται, το δεύτερο ο αριθμός των κατατμήσεων
integer_RDD = sc.parallelize (range (1,11), 2)
// Το Integer_RDD.glom() είναι μετασχηματισμός
// Το glom παίρνει κάθε κατάτμηση του RDD και τη μετατρέπει σε πίνακα
integer_RDD2 = Integer_RDD.glom()

//Εκτύπωση στην οθόνη των δυο κατατμήσεων του RDD
// Το Integer_RDD.collect() είναι δράση και θα προκαλέσει
// την εκτέλεση των παραπάνω μετασχηματισμών.
// Η δράση χρησιμοποιείται είτε για να αποθηκεύσει κανείς το αποτέλεσμα
// σε    κάποια θέση είτε για να το εμφανίσει.
integer_RDD2.collect()
Out[1]: [[1,2,3,4,5], [6,7,8,9,10]]

//Δράση που εκτυπώνει στην οθόνη τον αριθμό των κατατμήσεων
integer_RDD.getNumPartitions()
Out[2]: 2
```



Σχήμα 5-4: Ένα απλό παράδειγμα σε γλώσσα Python (μέσω του PySpark API) όπου δημιουργεί ο χρήστης μια λίστα με 10 ακέραιους αριθμούς με 2 κατατμήσεις

Η δομή δεδομένων RDD

Το RDD που συγκαταλέγεται στις αφαιρέσεις του Spark, είναι μια χαμηλού επιπέδου προγραμματιστική διεπαφή για την επεξεργασία μη δομημένων δεδομένων που θα πρέπει να καθορίσει ο χρήστης το σχήμα τους (schema). Η χρήση του RDD αποκρύπτει τόσο την κατάτμηση των δεδομένων όσο και τη διανομή, η οποία με τη σειρά της επέτρεψε τους δημιουργούς του Spark να σχεδιάσουν ένα παράλληλο υπολογιστικό πλαίσιο με μια προγραμματιστική διεπαφή (API) υψηλότερου επιπέδου για τέσσερις κύριες γλώσσες προγραμματισμού (Scala, Java, Python, R). Με αυτό τον τρόπο τα RDD επιτρέπουν τον χρήστη του Spark να εργάζεται με μια κατανεμημένη συλλογή με τον ίδιο τρόπο που θα εργαζόταν με μια τοπική μη κατανεμημένη συλλογή δεδομένων.

Είναι το αρχικό API που εισήγαγε το Spark και σχεδόν όλα τα API υψηλότερου επιπέδου (DataFrames, DataSets) αποσυντίθενται σε RDD. Σύμφωνα με τον δημιουργό του Spark θα πρέπει να παραμείνει κανείς στα Δομημένα API (DataFrames, DataSets) και να χρησιμοποιεί το RDD μόνο όταν είναι αναγκαίο (π.χ. διάβασμα και επεξεργασία ανεπεξέργαστων

δεδομένων (raw data)) [1]. Παρόλα αυτά επειδή η κατανόησή τους κρίνεται αναγκαία και για τις άλλες δομές δεδομένων, γίνεται μια εκτενής αναφορά σ' αυτό.

Τα RDD μπορούν να δημιουργηθούν με δύο τρόπους. Ο ένας είναι με την αναφορά συνόλων δεδομένων σε εξωτερικά συστήματα αποθήκευσης και ο δεύτερος με την εφαρμογή μετασχηματισμών όπως (π.χ. map, flatMap, filter, reduce, κ.λπ.) σε υπάρχοντα RDD.

Τα RDD είναι οκνηρά!

Για την υλοποίηση της *ανοχής σφάλματος*, οι κατατμήσεις RDD καταγράφουν ένα γενεαλογικό δέντρο (lineage): Μια συνταγή επανυπολογισμού της κατάτμησης RDD με βάση τα γονικά RDD και τη πράξη μετασχηματισμού που χρησιμοποιήθηκε για τη δημιουργία της κατάτμησης (βλέπε Σχήμα 6.5). Στο συγκεκριμένο πλαίσιο είναι ένας Κατευθυνόμενος Άκυκλος Γράφος (DAG) τελεστών που τους υπολογισμούς που έγιναν στο RDD. Κάθε κόμβος είναι μια κατάτμηση RDD και η ακμή αποτελεί το μετασχηματισμό πάνω στα δεδομένα της κατάτμησης του γονικού RDD. Εάν λόγω απώλειας ενός κόμβου εργάτη απολεσθεί οποιοδήποτε κατάτμηση ενός RDD, τότε αυτή η κατάτμηση μπορεί να υπολογιστεί εκ νέου από το αρχικό σύνολο δεδομένων που είναι ανεκτικό σε σφάλματα (καθότι τα RDD είναι αμετάβλητα) χρησιμοποιώντας το DAG.

Αυτό το DAG είναι που αναλύει το Spark για να κάνει βελτιστοποιήσεις. Η βελτιστοποίηση του DAG γίνεται με αναδιατάξεις και συνδυασμούς τελεστών όπου αυτό είναι δυνατόν. Για παράδειγμα, ας υποθέσουμε ότι πρέπει να υποβάλει κανείς μια εργασία Spark η οποία περιέχει μια πράξη map που ακολουθείται από μια πράξη filter. Ο μηχανισμός βελτιστοποίησης DAG του Spark θα αναδιατάξει τη σειρά αυτών των τελεστών, καθώς το φιλτράρισμα θα μειώσει τον αριθμό των εγγραφών που θα υποβληθούν στην πράξη map.

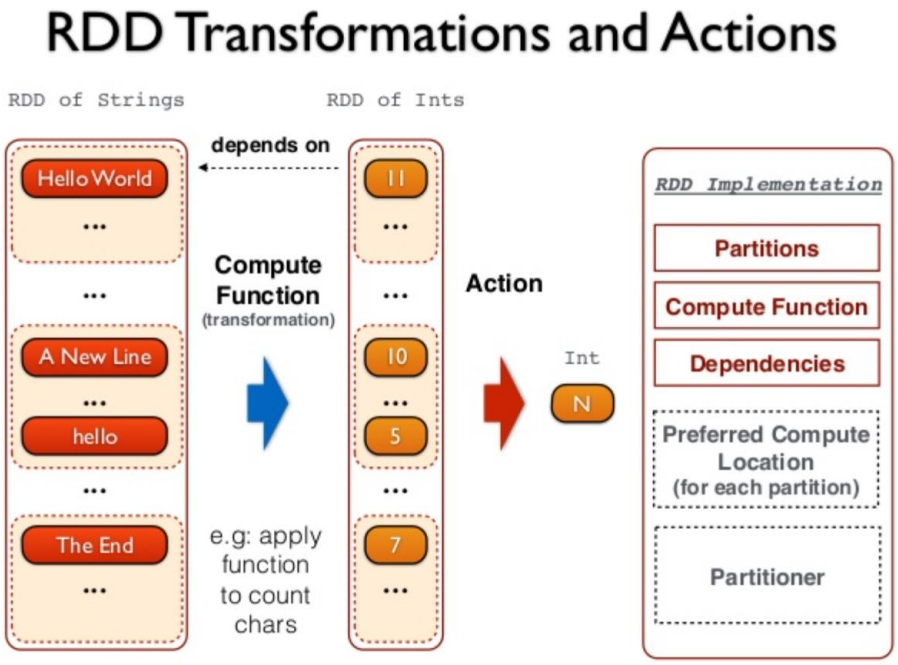
Η αποτίμηση του RDD είναι οκνηρή (δηλ. η εκτέλεση δεν ξεκινά μέχρι να εκτελεστεί ο κώδικας μιας δράσης) γεγονός που παρέχει πολλές ευκαιρίες για βελτιστοποιήσεις χαμηλού επιπέδου.

Το RDD αποθηκεύει τα ακόλουθα μεταδεδομένα (βλέπε Σχήμα 6.5):

- Κατατμήσεις
- Εξαρτήσεις (dependencies) - κατάλογος των γονικών RDDs που εμπλέκονται στον υπολογισμό δηλαδή το γενεαλογικό δέντρο.
- Υπολογισμός (Compute Function) – Η συνάρτηση για τον υπολογισμό του τέκνου RDD από το γονικό RDD που προκύπτει από τις εξαρτήσεις.

▪ Προτιμώμενες τοποθεσίες υπολογισμού (Preferred compute locations) - Ποιο είναι το καλύτερο μέρος για την τοποθέτηση των υπολογισμών σε κατατμήσεις (τοπικότητα δεδομένων)

▪ Partitioner - είναι ο αλγόριθμος που καθορίζει τον τρόπο με τον οποίο χωρίζονται τα δεδομένα στις κατατμήσεις (HashPartitioner, RangePartitioner, κλπ.).



Πηγή: https://www.slideshare.net/frodriguezolivera/apache-spark-41601032

Σχήμα 5-5: Τα μεταδεδομένα που αποθηκεύει το RDD

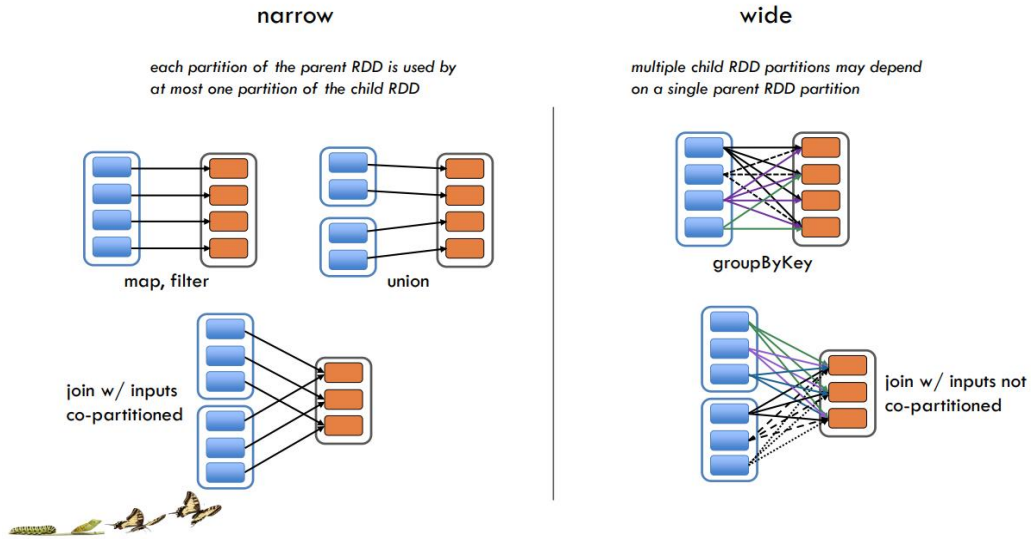
Πράξεις στα RDD

Υπάρχουν δύο τύποι πράξεων που μπορούμε να κάνει ο χρήστης του Spark σε ένα RDD: οι μετασχηματισμοί (transformations) και οι δράσεις (actions).

- Ένας μετασχηματισμός είναι μια συνάρτηση που επιστρέφει ένα νέο RDD. Επίσης, το νέο RDD διατηρεί έναν δείκτη στο γονικό RDD του. Μπορεί να εφαρμοστεί είτε σε δεδομένα που βρίσκονται σε μόνιμη αποθήκευση είτε σε άλλα RDD. Ο μετασχηματισμός είναι μια οκνηρή πράξη. Δηλαδή είναι μια πράξη που δεν θα ολοκληρωθεί τη στιγμή που εκτελεί κανείς τον κώδικά του. Θα εκτελεστεί μόνο όταν καλέσει μια δράση. Υπάρχουν δύο είδη μετασχηματισμών: εκείνοι που καθορίζουν στενές εξαρτήσεις και εκείνοι που καθορίζουν ευρείες εξαρτήσεις. Οι πρώτοι ονομάζονται στενοί μετασχηματισμοί (narrow transformations) ενώ οι δεύτεροι ονομάζονται εκτεταμένοι μετασχηματισμοί (wide transformations).

Μια **αναδιάταξη** (shuffle) συμβαίνει όταν αναδιανεμηθούν τα δεδομένα μεταξύ κατατμήσεων. Αυτό απαιτείται όταν ένας μετασχηματισμός απαιτεί πληροφορίες από άλλες κατατμήσεις, όπως αθροίζοντας όλες τις τιμές σε μια στήλη. Το Spark θα συλλέξει τα απαιτούμενα δεδομένα από κάθε κατάτμηση και θα τα συνδυάσει σε ένα νέο διαμέρισμα, πιθανότατα σε διαφορετικό εκτελεστή.

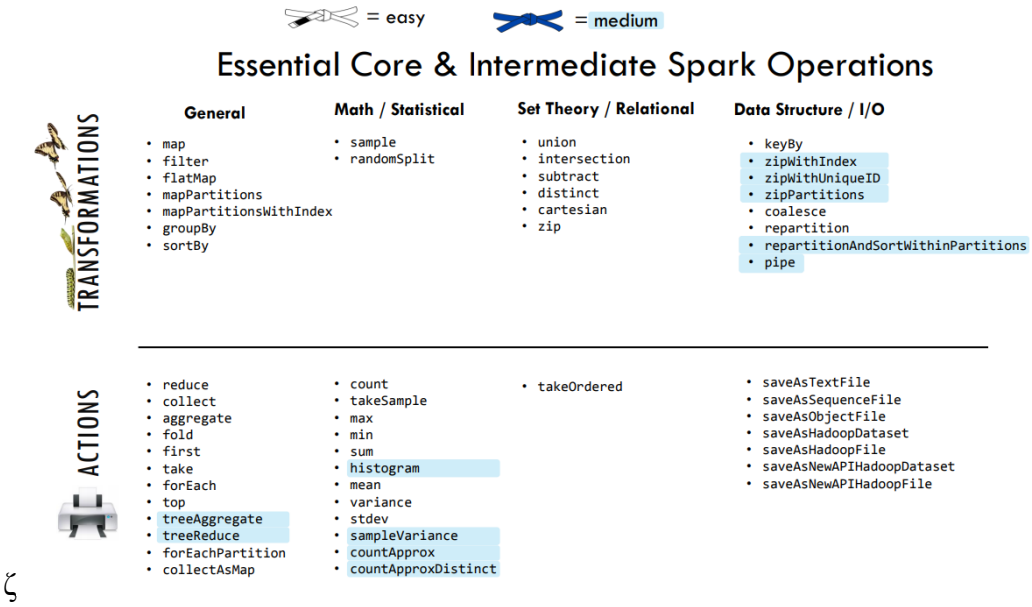
- Οι στενοί μετασχηματισμοί δεν απαιτούν την αναδιάταξη / ανακατεύθυνση (shuffling) των δεδομένων σε άλλες κατατμήσεις. Κάθε κατάτμηση του γονικού RDD χρησιμοποιείται από το πολύ μια κατάτμηση του τέκνου RDD (βλέπε Σχήμα 6.6). Είναι το αποτέλεσμα συναρτήσεων όπως map, filter (βλέπε Σχήμα 6.7) [27]. Η ομαδοποίηση των στενών μετασχηματισμών σε ένα στάδιο ονομάζεται σωλήνωση ή διασωλήνωση (pipelining) και είναι μια από τις βελτιστοποιήσεις που κάνει το Spark. Έτσι οι μετασχηματισμοί που έχουν στενές εξαρτήσεις είναι γρήγοροι.
- Στους εκτεταμένους μετασχηματισμούς πολλαπλές κατατμήσεις RDD τέκνων, εξαρτώνται από μια μόνο κατάτμηση γονικού RDD (βλέπε Σχήμα 6.6). Οι εκτεταμένοι μετασχηματισμοί είναι το αποτέλεσμα συναρτήσεων όπως groupByKey και reduceByKey (βλέπε Σχήμα 6.8) [27]. Είναι αργοί μετασχηματισμοί. Απαιτούν την αναδιάταξη όλων ή ορισμένων δεδομένων μέσω του δικτύου.



Πηγή: https://training.databricks.com/visualapi.pdf

Σχήμα 5-6: Στενοί (narrow) και εκτεταμένοι (wide) μετασχηματισμοί (transformations)

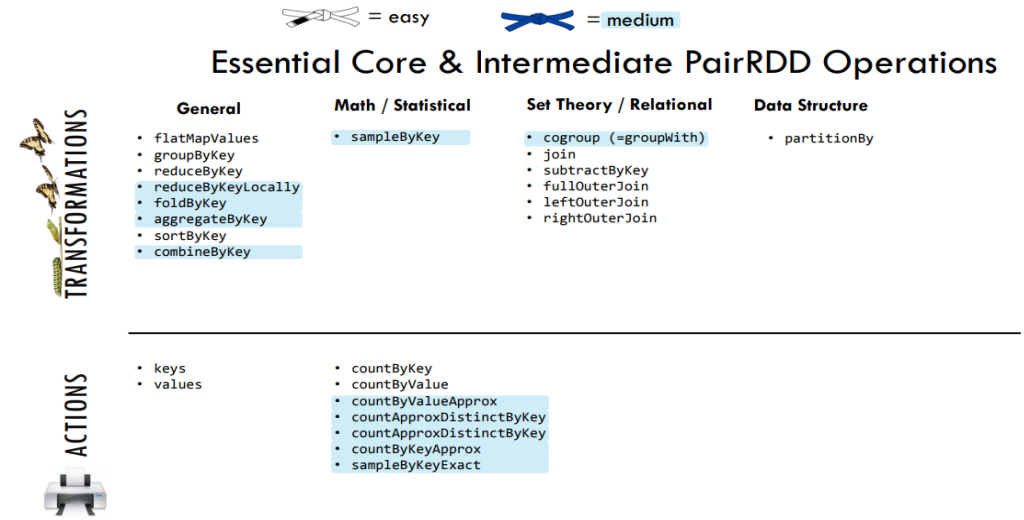
Χάρη στην τεχνική της οκνηρής αποτίμησης (lazy evaluation), ο Χρονοδρομολογητής (Scheduler) θα είναι σε θέση να βελτιστοποιήσει τα στάδια πριν από την υποβολή της εργασίας (job): διασωληνώνει στενές πράξεις σε ένα στάδιο, επιλέγει αλγορίθμους που βασίζονται σε κατατμήσεις (προσπαθεί να ελαχιστοποιήσει τυχόν αναδιατάξεις), επαναχρησιμοποιεί δεδομένα που έχουν αποθηκευτεί προηγουμένως στην cache.



ζ

Πηγή: https://training.databricks.com/visualapi.pdf

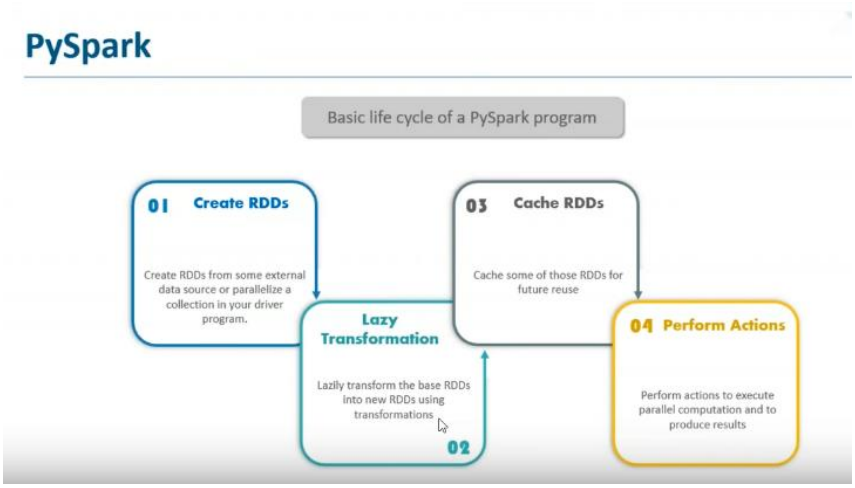
Σχήμα 5-7: Βασικές πράξεις στο Spark



Πηγή: https://training.databricks.com/visualapi.pdf

Σχήμα 5-8: Βασικές Πράξεις στα PairRDD

Επομένως, ουσιαστικά ο κύκλος ζωής ενός προγράμματος pyspark επεξεργασίας δεδομένων περιλαμβάνει την ανάγνωση της πηγής δεδομένων, την εφαρμογή μιας σειράς μετασχηματισμών, την αποθήκευση κάποιων εξ αυτών στην μνήμη (cache) για μελλοντική χρήση και την υλοποίηση δράσεων για την εξαγωγή του αποτελέσματος (βλέπε Σχήμα 6.9).



Πηγή: https://www.slideshare.net/EdurekaIN/pyspark-tutorial-introduction-to-apache-spark-with-python-pyspark-training-edureka

Σχήμα 5-9: Ο βασικός κύκλος ζωής ενός pyspark προγράμματος

Πώς λειτουργεί το Spark;

Πριν από τη ανάλυση του πώς λειτουργεί το Apache Spark, ας κατανοήσουμε τη ορολογία του που παρουσιάζεται στον παρακάτω πίνακα.

Πίνακας 5-1: Ορολογία του Spark	
Application:	Αντιστοιχεί στην εκκίνηση του SparkSession / SparkContext. Κάθε εφαρμογή αποτελείται από μια η περισσότερες εργασίες (jobs)
Driver (Οδηγός):	Το πρόγραμμα / διεργασία που είναι υπεύθυνο για την εκτέλεση της εργασίας πάνω από την μηχανή Spark (Spark engine)
SparkSession:	Εισήχθη στο Spark 2.0 με σκοπό να αποτελέσει τον νέο ενιαίο τρόπο πρόσβασης στα διαφορετικά περιβάλλοντα (contexts) που προϋπήρχαν (SparkContext, SQLContext, HiveContext) αλλά και το Spark StreamingContext[20] (το σημείο εισόδου για τη λειτουργία Spark Streaming). Με την πρόσβαση στο SparkSession, έχει ο χρήστης αυτόματα πρόσβαση στο SparkContext και μπορεί να έχει πρόσβαση και σε όλα τα παραπάνω περιβάλλοντα. Το αντικείμενο SparkSession αποτελεί το νέο σημείο εισόδου σε ένα περιβάλλον χρόνου εκτέλεσης (runtime environment) Spark και επιτρέπει τη χρήση των DataFrame και Dataset APIs για τον προγραμματισμό του Spark. Θα πρέπει να σημειωθεί ότι δεν χρειάζεται να δημιουργήσει ο χρήστης ένα αντικείμενο sparkSession όταν χρησιμοποιεί το διαδραστικό κέλυφος του Spark (pyspark) διότι δημιουργείται από το Spark για λογαριασμό του χρήστη ως μεταβλητή spark.
SparkContext:	Το SparkContext (πλέον είναι προσβάσιμο από το SparkSession) είναι το σημείο πρόσβασης σε όλες τις λειτουργίες του Spark. Μόνο ένα SparkContext υπάρχει ανά

[20] *Μέσω του SparkContext που είναι ενσωματωμένο στο SparkSession*

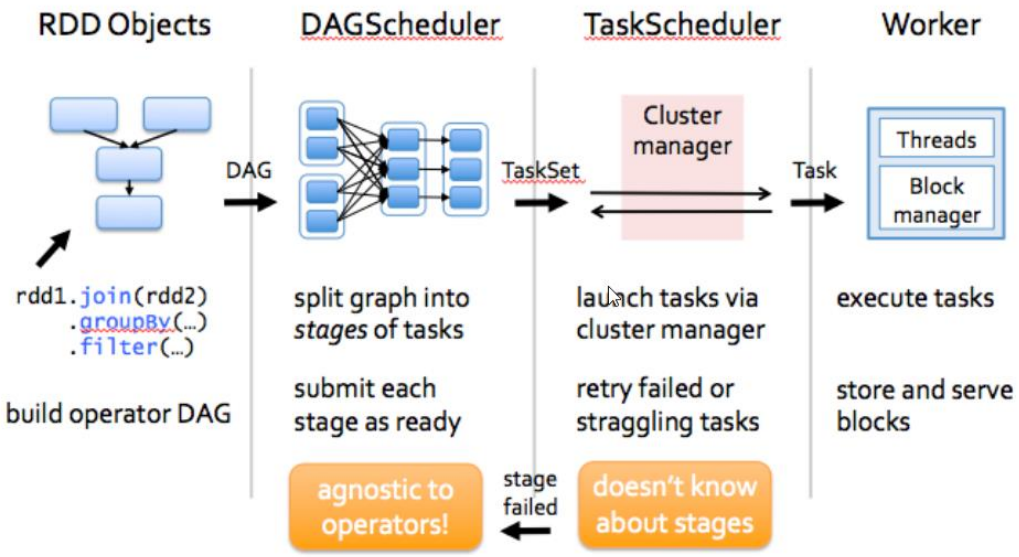
	JVM. Είναι το πρώτο αντικείμενο που πρέπει να δημιουργήσει ένα πρόγραμμα Spark για να αποκτήσει πρόσβαση στη συστάδα.
Job (Εργασία):	Όταν κληθεί μια δράση (action) σε ένα RDD, δημιουργείται μια εργασία (job). Job αποκαλείται η εργασία που υποβάλλεται στο Spark και έχει ένα η περισσότερα στάδια. Ένα κομμάτι κώδικα που διαβάζει κάποια είσοδο από το HDFS ή τοπικά, εκτελεί κάποιον παράλληλο υπολογισμό στα δεδομένα και γράφει κάποια δεδομένα εξόδου.
Stage (Στάδιο):	Τα jobs χωρίζονται σε στάδια. Ένα νέο στάδιο θα δημιουργηθεί όταν εμφανιστεί ένας εκτεταμένος μετασχηματισμός.
Task (Εργο) (η μικρότερη μονάδα υπολογιστικής εργασίας του Spark):	Κάθε στάδιο χωρίζεται περαιτέρω σε έργα που αντιστοιχούν στον αριθμό των κατατμήσεων (partitions) στο RDD (ένα έργο ανά κατάτμηση). Έτσι τα έργα είναι οι μικρότερες μονάδες υπολογιστικής εργασίας στο Spark.
Master (Αφέντης):	Η μηχανή στην οποία εκτελείται το πρόγραμμα οδήγησης.
Slave (Σκλάβος):	Η μηχανή στην οποία εκτελείται η διεργασία εκτελεστής (executor).
Cluster Manager (Διαχειριστής Συστάδας):	Το Spark βασίζεται στον διαχειριστή συστάδας για την εκκίνηση και τη διανομή των εκτελεστών σύμφωνα με τις παραμέτρους διαμόρφωσης που ορίζονται από την εφαρμογή Spark. Σε ορισμένες περιπτώσεις, ακόμη και οι οδηγοί ξεκινούν μέσω αυτού (ανάπτυξη εφαρμογής Spark σε cluster mode).
Executor (Εκτελεστής):	Η διεργασία που είναι υπεύθυνη για την εκτέλεση ενός έργου και την επιστροφή του αποτελέσματος. Επίσης κρατάει δεδομένα εντός μνήμης (in-memory) ή σε δίσκους αποθήκευσης. Διαβάζει και γράφει δεδομένα σε εξωτερικές πηγές.
DAG:	Το DAG (Directed Acyclic Graph – Κατευθυνόμενος Άκυκλος Γράφος), στο παρόν πλαίσιο είναι ένας DAG τελεστών.
Pipelining (Διασωλήνωση):	Η ομαδοποίηση των υπολογισμών σε ένα μόνο στάδιο, όταν οι μετασχηματισμοί RDD μπορούν να υπολογιστούν χωρίς να απαιτείται μετακίνηση δεδομένων (ομαδοποίηση διαδοχικών στενών μετασχηματισμών).

Στη συνέχεια ακολουθεί μια επεξήγηση της λειτουργίας του Spark:

- Καθώς εκτελείται το τμήμα του κώδικα του χρήστη που περιλαμβάνει μετασχηματισμούς στην κονσόλα Spark, το Spark δημιουργεί ένα γράφο τελεστών [4].
- Όταν το Spark εκτελέσει μια δράση (όπως collect), ο γράφος υποβάλλεται σε έναν χρονοδρομολογητή DAG (DAG Scheduler) ο οποίος μετασχηματίζει το λογικό πλάνο εκτέλεσης (logical execution plan - δηλαδή το γενεαλογικό γράφο του RDD ή DAG τελεστών) σε φυσικό πλάνο εκτέλεσης (physical execution plan), διαιρώντας το DAG τελεστών (βλέπε Σχήμα 5.10) σε στάδια [4].
- Ένα στάδιο αποτελείται από έργα που βασίζονται στις κατατμήσεις των δεδομένων εισόδου. Ο DAG Scheduler διασωληνώνει τους τελεστές στενών μετασχηματισμών για να βελτιστοποιήσει το γράφο. Για παράδειγμα αν έχουμε διαδοχικούς τελεστές map αυτοί μπορούν να προγραμματιστούν σε ένα μόνο στάδιο. Η διασωλήνωση των υπολογισμών είναι πολύ γρηγορότερη από την εγγραφή των ενδιάμεσων

αποτελεσμάτων στη μνήμη ή στο δίσκο μετά από κάθε βήμα. Αυτή η βελτιστοποίηση είναι καθοριστική στην απόδοση του Spark. Το τελικό αποτέλεσμα ενός DAG Scheduler είναι ένα σύνολο σταδίων (βλέπε Σχήμα 5.12) [4].

- Τα στάδια μεταβιβάζονται στον Task Scheduler (βλέπε Σχήμα 5.10) ο οποίος είναι υπεύθυνος για την εκκίνηση των έργων (tasks) που υπάρχουν ανά στάδιο μέσω του διαχειριστή συστάδας (Spark Standalone/ Yarn/ Mesos/ kubernetes). Ο Task Scheduler δεν είναι ενήμερος για τις εξαρτήσεις μεταξύ των σταδίων [4].
- Ο εργάτης (worker) εκτελεί τα έργα και επιστρέφει τα αποτελέσματα των υπολογισμών στον πελάτη. Ένα νέο JVM ξεκινά ανά εργασία (job). Ο εργάτης γνωρίζει μόνο τον κώδικα που έχει περάσει σε αυτόν [4].

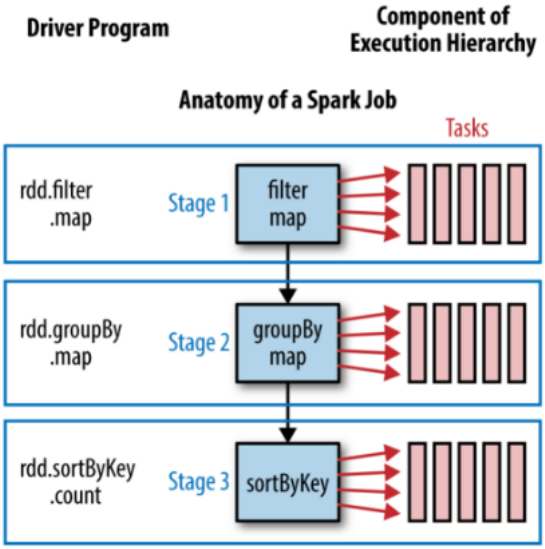


Πηγή: [4]

Σχήμα 5-10: Ο τρόπος λειτουργίας του Spark

```
#
rdd1 = sc.textFile("hdfs://...")
rdd1.filter(filterFunc)\
       .map(someFunc)\
        .groupByKey()\
        .map(someOtherFunc)\
        .sortByKey()\
        .count()

# όπου red_color :stage 1, black_color:stage2,  blue_color:stage3, green_color = action
```

Σχήμα 5-11: Παράδειγμα με στενούς και εκτεταμένους μετασχηματισμούς και η οριοθέτηση των σταδίων από τους εκτεταμένους μετασχηματισμούς (groupByKey(), sortByKey())



Πηγή: [2]

Σχήμα 5-12: Ένα DAG σταδίων για το παραπάνω πρόγραμμα.

Τα μπλε τετράγωνα υποδηλώνουν στάδια τα οποία οριοθετούνται από τους εκτεταμένους μετασχηματισμούς groupByKey() και sortByKey(). Τα κόκκινα τετράγωνα αναφέρονται στα έργα (tasks) ανά στάδιο, τα οποία είναι ίσα με τον αριθμό των κατατμήσεων και τα οποία εκτελούνται παράλληλα. Παρατηρούμε ότι επειδή οι μετασχηματισμοί map() και filter() έχουν στενές εξαρτήσεις μπορούν να διασωληνωθούν.

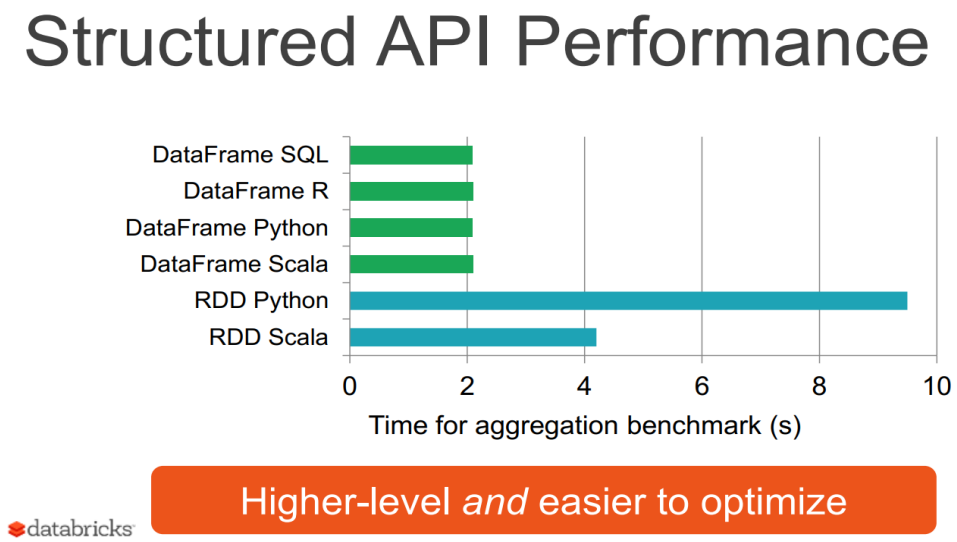
Η δομή δεδομένων DataFrame

Όπως ήδη έχουμε αναφέρει στο κεφάλαιο 3 τα Dataframe είναι δομημένα πάνω στα RDD και κατά συνέπεια απολαμβάνουν τα πλεονεκτήματά τους (*αμετάβλητη*, *ανεκτική σε σφάλματα κατανεμημένη* συλλογή δεδομένων). Σε αντίθεση όμως με τα RDD, τα δεδομένα οργανώνονται σε ονομασμένες στήλες, όπως ένας πίνακας σε μια σχεσιακή βάση δεδομένων. Το DataFrame περιέχει πληροφορίες (μεταδεδομένα) σχετικά με τη δομή των δεδομένων του όπως τα ονόματα των στηλών, οι τύποι δεδομένων κ.λπ, που βοηθά το Apache Spark να κατανοήσει το σχήμα (βλέπε κεφάλαιο 1) ενός DataFrame γεγονός που του επιτρέπει να εφαρμόσει ένα σύνολο από βελτιστοποιήσεις που δεν είναι δυνατές στο RDD. Όπως φαίνεται στο Σχήμα 5.13 η απόδοση του DataFrame είναι η ίδια ανεξάρτητα από τη γλώσσα (Scala / Python / SQL / R API γλώσσας) που θα επιλέξει ο χρήστης αφού στο τέλος μετά τις βελτιστοποιήσεις του Catalyst optimizer (βλέπε στην παρακάτω ενότητα "Πλάνο εκτέλεσης") καταλήγουν όλα στο ίδιο πλάνο εκτέλεσης. Με τη χρήση υψηλότερων προγραμματιστικών διεπαφών όπως SQL, DataFrame, DataSet τα προγράμματα περιγράφουν τις πράξεις που απαιτούνται να γίνουν πάνω στα δεδομένα και όχι τον τρόπο με τον οποίο θα υλοποιηθούν οι πράξεις αυτές.

Το Spark SQL ενσωματώνει τη σχεσιακή επεξεργασία με τον συναρτησιακό προγραμματισμό του Spark. Με το Spark SQL, μπορεί να εκτελέσει κανείς εύκολα πράξεις σε δομημένα

δεδομένα χρησιμοποιώντας ερωτήματα γραμμένα σε SQL ή DataFrame dsl (domain specific language / Γλώσσα προγραμματισμού Ειδικού Πεδίου ή Γ.Ε.Π). Όπως τα RDD έτσι και τα DataFrame υποστηρίζουν δύο ειδών πράξεις, τους **Μετασχηματισμούς** και τις **Δράσεις** και ισχύουν ότι και για τα RDD. Οι μετασχηματισμοί αποτιμώνται οκνηρά ενώ οι δράσεις αποτιμώνται ανυπόμονα (βλέπε ενότητα "*Πράξεις στα RDD*").

Ένα χαρακτηριστικό των ερωτημάτων SQL είναι ότι επιστρέφουν DataFrames, έτσι ένας χρήστης μπορεί στη συνέχεια να χρησιμοποιήσει προγραμματιστικά μια από τις γλώσσες (Scala, Java, Python και R) που υποστηρίζει το Spark για να ολοκληρώσει την ανάλυσή του.



Πηγή:https://www.slideshare.net/databricks/largescale-data-science-in-apache-spark-20

Σχήμα 5-13: Με τα DataFrame πετυχαίνουμε καλύτερη απόδοση σε σχέση με τα RDD ανεξάρτητα από την επιλογή γλώσσας (SQL, R, Python, Scala).

Πλάνο Εκτέλεσης

Θα επικεντρωθούμε στη συνέχεια στο πώς το Spark μετατρέπει ένα Λογικό Πλάνο (Logical Plan) σε ένα Βελτιστοποιημένο Λογικό Πλάνο (Optimized Logical Plan) και έπειτα σε ένα ή περισσότερα Φυσικά Πλάνα (Physical Plans).

Αυτό που θα πρέπει να γίνει κατανοητό είναι ότι το Spark είναι το ίδιο μια γλώσσα προγραμματισμού και ως εκ τούτου διαθέτει τους δικούς του τύπους δεδομένων. Οι τύποι δεδομένων αυτοί αντιστοιχούν απευθείας στα διαφορετικά API γλώσσας που διατηρεί το Spark και υπάρχει ένας πίνακας αναζήτησης (lookup table) για κάθε ένα από αυτούς σε Scala, Java, Python, SQL, και R. Ακόμα και αν χρησιμοποιούμε δομημένα API του Spark μέσα από την Python ή την R, η πλειονότητα των χειρισμών μας θα λειτουργήσει αυστηρά στους τύπους Spark και όχι στους τύπους Python. Για παράδειγμα, ο ακόλουθος κώδικας δεν πραγματοποιεί πρόσθεση σε Python [1]. Η πρόσθεση γίνεται σε Spark:

```
In [6]: # Python
        df = spark.range(10).toDF("number")
        df.select(df["number"] + 10)

Out[6]: DataFrame[(number + 10): bigint]
```

Στη πράξη αυτή της πρόσθεσης το Spark θα μετατρέψει πρώτα την Python έκφραση στην εσωτερική αναπαράσταση Catalyst του Spark και ακολούθως θα τελέσει την πράξη σε αυτή την εσωτερική αναπαράσταση [1].

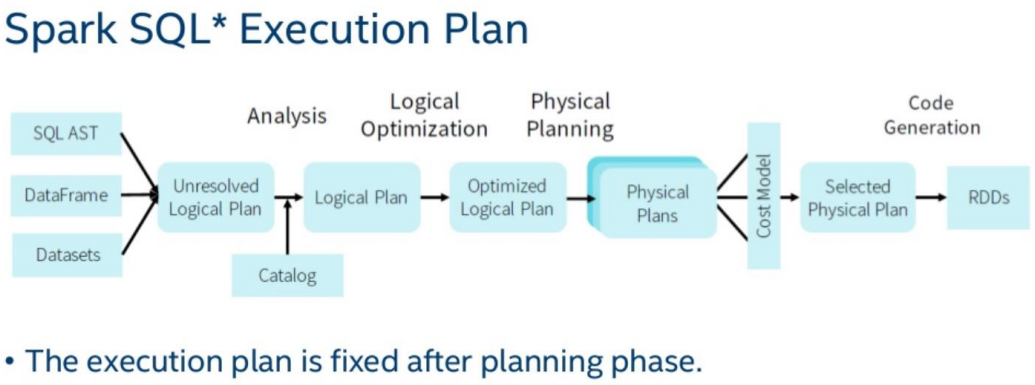
Στα δομημένα API εκτός από τους πίνακες και τις όψεις (views) SQL ανήκουν και τα Dataframes (untyped) και τα Datasets (typed). Ο προσδιορισμός untyped (χωρίς τύπους) για τα Dataframes δεν είναι απολύτως ακριβής. Τα Dataframes έχουν τύπους (βλέπε πληροφορίες σχετικά με τους τύπους του Spark), αλλά το Spark ελέγχει εάν αυτοί οι τύποι ευθυγραμμίζονται με εκείνους που καθορίζονται στο σχήμα μόνο κατά το χρόνο εκτέλεσης. Αντίθετα τα Datasets (μόνο σε JVM γλώσσες) ελέγχουν αν υπάρχει συμμόρφωση με τους τύπους του σχήματος κατά το χρόνο μεταγλώττισης. Η Python έχει μόνο Dataframes και όταν τα χρησιμοποιεί κανείς, εκμεταλλεύεται το βελτιστοποιημένο εσωτερικό τύπο δεδομένων του Spark 'Row'.

Κάθε Dataframe αποτελείται από ένα σύνολο εγγραφών, όπου κάθε εγγραφή είναι ένα αντικείμενο τύπου Row. Ο χειρισμός των Row γίνεται με εκφράσεις στήλης (column expressions) προκειμένου να προκύψουν χρήσιμες τιμές.

Στον πυρήνα του Spark SQL υπάρχει ένας επεκτάσιμος βελτιστοποιητής ο Catalyst optimizer, που βοηθά στην υποστήριξη ενός ευρέος φάσματος πηγών δεδομένων και αλγορίθμων. Όταν χρησιμοποιεί κανείς το Dataframe, δεν καθορίζει πια ένα DAG (Directed Acyclic Graph), δημιουργεί ένα Αφηρημένο Συντακτικό Δέντρο (Abstract Syntax Tree / AST) που ο βελτιστοποιητής Catalyst θα αναλύσει συντακτικά, θα ελέγξει και θα βελτιώσει χρησιμοποιώντας κανόνες βελτιστοποίησης και τη βελτιστοποίηση βάσει κόστους. Χρησιμοποιείται το γενικό πλαίσιο μετατροπής δέντρων του Catalyst σε τέσσερις φάσεις, όπως φαίνεται στο παρακάτω σχήμα: (1) ανάλυση ενός λογικού πλάνου για την επίλυση αναφορών, (2) βελτιστοποίηση λογικού πλάνου, (3) φυσικό πλάνο, και (4) ερώτημα σε Java bytecode. Όλες οι φάσεις βασίζονται αποκλειστικά σε κανόνες εκτός από τη φάση του φυσικού πλάνου, όπου ο Catalyst μπορεί να δημιουργήσει πολλαπλά πλάνα και να τα συγκρίνει με βάση το κόστος [21]. Κάθε φάση χρησιμοποιεί διαφορετικούς τύπους κόμβων δέντρων. Ο Catalyst

[21] *Ο χρόνος εκτέλεσης και η χρήση πόρων ορίζεται ως κόστος*

περιλαμβάνει βιβλιοθήκες κόμβων για εκφράσεις, τύπους δεδομένων και λογικούς και φυσικούς τελεστές.



Πηγή: https://www.slideshare.net/databricks/an-adaptive-execution-engine-for-apache-spark-with-carson-wang

Σχήμα 5-14: Πλάνο Εκτέλεσης του Spark SQL

Λογικό Πλάνο

Στη **φάση της ανάλυσης ενός λογικού πλάνου** ο κώδικας του χρήστη εφόσον είναι έγκυρος μετατρέπεται σε λογικό πλάνο το οποίο αναπαριστά τους μετασχηματισμούς που περιλαμβάνονται στον κώδικα χωρίς να υπάρχει καμία αναφορά για τους εκτελεστές. Έτσι προκύπτει ένα **ανεπίλυτο λογικό πλάνο** (unresolved logical plan) δεδομένου ότι μπορεί να αναφέρεται σε πίνακες και στήλες οι οποίες μπορεί να μην υφίστανται. Το Spark προκειμένου να το επιλύσει χρησιμοποιεί τον κατάλογο [22](catalog) στον αναλυτή ο οποίος μπορεί να το απορρίψει αν οι πίνακες ή τα ονόματα των στηλών δεν υπάρχουν στον κατάλογο. Σε διαφορετική περίπτωση το επιλυμένο λογικό πλάνο περνά στον βελτιστοποιητή Catalyst[1].

Η **φάση της λογικής βελτιστοποίησης** εφαρμόζει στο λογικό πλάνο τυπικές βελτιστοποιήσεις βασισμένες σε κανόνες. Αυτές περιλαμβάνουν την *αναδίπλωση σταθερών* [23](constant folding), την ώθηση προς τα κάτω κατηγορημάτων (predicate push down), το κλάδεμα προβολής (projection pruning), τη διάδοση null (null propagation), την απλοποίηση λογικής έκφρασης (boolean expression simplification) και άλλους κανόνες. Ο βελτιστοποιητής Catalyst είναι

[22] *Ο κατάλογος είναι η διεπαφή για τη διαχείριση ενός καταλόγου μεταδεδομένων των σχεσιακών οντοτήτων (π.χ. βάσεις δεδομένων, πίνακες, συναρτήσεις, στήλες πινάκων και προσωρινές όψεις).*

[23] *είναι η διαδικασία της αναγνώρισης και της αξιολόγησης των σταθερών εκφράσεων κατά το χρόνο μεταγλώττισης αντί να υπολογιστούν κατά τη διάρκεια του χρόνου εκτέλεσης. Οι όροι σε σταθερές εκφράσεις είναι τυπικά απλά λεκτικά, όπως το ακέραιο λεκτικό 2, αλλά μπορεί επίσης να είναι μεταβλητές των οποίων οι αξίες είναι γνωστές στον χρόνο μεταγλώττισης. Για παράδειγμα στη δήλωση: i = 320 * 200 * 32, οι περισσότεροι μεταγλωττιστές δεν θα παράγουν δύο εντολές πολλαπλασιασμού και μια αποθήκευσης για αυτή τη δήλωση. Αντιθέτως, εντοπίζουν κατασκευές όπως αυτές και αντικαθιστούν τις υπολογιζόμενες τιμές κατά τον χρόνο μεταγλώττισης (στην περίπτωση αυτή, 2.048.000) (πηγή: https://en.wikipedia.org/wiki/Constant_folding).*

επεκτάσιμος που σημαίνει ότι τα διάφορα πακέτα μπορούν να τον επεκτείνουν ώστε να συμπεριλάβει τους δικούς τους κανόνες για βελτιστοποιήσεις ειδικών πεδίων (domain specific) [1].

Φυσικό Πλάνο

Στη **φάση φυσικού σχεδιασμού**, το Spark SQL λαμβάνει ένα λογικό πλάνο και παράγει ένα ή περισσότερα φυσικά πλάνα (ή πλάνα Spark). Το φυσικό πλάνο καθορίζει τον τρόπο εκτέλεσης του λογικού πλάνου στη συστάδα. Στη συνέχεια επιλέγει ένα πλάνο χρησιμοποιώντας ένα μοντέλο κόστους. Στη φάση αυτή τα ερωτήματα σε SQL, DataFrame και Dataset (για τις γλώσσες Scala και Java) μετατρέπονται σε RDD μετασχηματισμούς και είναι ο λόγος που το Spark αναφέρεται και ως μεταγλωττιστής. Ο φυσικός προγραμματιστής εκτελεί επίσης φυσικές βελτιστοποιήσεις βασισμένες σε κανόνες, όπως προβολές σωλήνωσης (pipelining projections) ή φίλτρων σε μια πράξη map του Spark. Επιπλέον, μπορεί να ωθήσει πράξεις από το λογικό πλάνο σε πηγές δεδομένων που υποστηρίζουν ώθηση προς τα κάτω κατηγορήματος (predicate push down) ή προβολής.

Εκτέλεση Φυσικού Πλάνου

Η **τελική φάση** της βελτιστοποίησης των ερωτημάτων περιλαμβάνει περαιτέρω βελτιστοποιήσεις κατά το χρόνο εκτέλεσης, δημιουργώντας εγγενή Java bytecode, που μπορεί να αφαιρέσει ολόκληρα έργα ή στάδια, για εκτέλεση σε κάθε μηχανή.

Προβολή του Λογικού και Φυσικού Πλάνου

Με τη μέθοδο DataFrame.explain() μπορεί να δει ο χρήστης τα αποτελέσματα των βελτιστοποιήσεων και να εξετάσει τα δημιουργούμενα πλάνα μιας εντολής χωρίς να την εκτελέσει το Spark. Χρησιμοποιώντας το όρισμα true στην explain() μπορεί να δει τόσο τα λογικά (συντακτικά αναλυμένο / ανεπίλυτο λογικό πλάνο, αναλυμένο / επιλυμένο λογικό πλάνο καθώς και το βελτιστοποιημένο λογικό πλάνο) πλάνα όσο και το φυσικό πλάνο, διαφορετικά του δείχνει μόνο το φυσικό πλάνο. Μπορεί επίσης να χρησιμοποιήσει αυτό το φυσικό πλάνο εκτέλεσης για εκσφαλμάτωση ή για να βελτιστοποιήσει τα ερωτήματά του. Τα πλάνα της *explain* μπορούν να διαβαστούν από πάνω προς τα κάτω, το πάνω μέρος είναι το τελικό αποτέλεσμα και το κάτω είναι η πηγή (ή οι πηγές) των δεδομένων.

Παράδειγμα:

Αφού φορτώσει ο χρήστης το αρχείο books.csv, (i) επιλέγει τις στήλες bookID, title και authors, (ii) ομαδοποιεί τις εγγραφές ως προς τον συγγραφέα (στήλη authors), (iii) μετράει τους τίτλους που είναι διαφορετικοί με την countDistinct(“title”) καθώς επίσης μετονομάζει

την στήλη που θα προστεθεί από τον μετασχηματισμό σε “Book_Titles”, (iv) ταξινομεί τα αποτελέσματα ως προς τη στήλη “Book_Titles”, κατά φθίνουσα διάταξη και (v) περιορίζει τα αποτελέσματα στις 10 πρώτες εγγραφές, (vi) ζητά να του δώσει τα λογικά πλάνα και το φυσικό πλάνο με την explain(True). Μπορεί στη συνέχεια ο χρήστης να δει το αποτέλεσμα αν αντικαταστήσει την explain(True) με την show().

```
#Python
booksDF = spark\
      .read\
      .option("inferSchema", "true")\
      .option("header", "true")\
      .csv("./test/books.csv")

from pyspark.sql.functions import desc
booksDF.select("bookID", "title", "authors")\
       .groupBy("authors")\
       .agg(countDistinct("title")\
       .alias("Book_Titles"))\
       .sort(desc("Book_Titles"))\
       .limit(10).explain(True)
```

```
== Parsed Logical Plan ==
GlobalLimit 10
+- LocalLimit 10
   +- Sort [Book_Titles#926L DESC NULLS LAST], true
      +- Aggregate [authors#104], [authors#104, count(distinct title#103) AS Book_Titles#926L]
         +- Project [bookID#102, title#103, authors#104]
            +-        Relation[bookID#102,title#103,authors#104,average_rating#105,isbn#106,isbn13#107,language_code#108,#
num_pages#109,ratings_count#110,text_reviews_count#111] csv

== Analyzed Logical Plan ==
authors: string, Book_Titles: bigint
GlobalLimit 10
+- LocalLimit 10
   +- Sort [Book_Titles#926L DESC NULLS LAST], true
      +- Aggregate [authors#104], [authors#104, count(distinct title#103) AS Book_Titles#926L]
         +- Project [bookID#102, title#103, authors#104]
            +-
Relation[bookID#102,title#103,authors#104,average_rating#105,isbn#106,isbn13#107,language_code#108,#
num_pages#109,ratings_count#110,text_reviews_count#111] csv

== Optimized Logical Plan ==
GlobalLimit 10
```

+- LocalLimit 10

+- Sort [Book_Titles#926L DESC NULLS LAST], true

+- Aggregate [authors#104], [authors#104, count(distinct title#103) AS Book_Titles#926L]

+- Project [title#103, authors#104]

+- Relation[bookID#102,title#103,authors#104,average_rating#105,isbn#106,isbn13#107,language_code#108,#num_pages#109,ratings_count#110,text_reviews_count#111] csv

== Physical Plan ==

TakeOrderedAndProject(limit=10, orderBy=[Book_Titles#926L DESC NULLS LAST], output=[authors#104,Book_Titles#926L])

+- *(3) HashAggregate(keys=[authors#104], functions=[count(distinct title#103)], output=[authors#104, Book_Titles#926L])

+- Exchange hashpartitioning(authors#104, 200)

+- *(2) HashAggregate(keys=[authors#104], functions=[partial_count(distinct title#103)], output=[authors#104, count#931L])

+- *(2) HashAggregate(keys=[authors#104, title#103], functions=[], output=[authors#104, title#103])

+- Exchange hashpartitioning(authors#104, title#103, 200)

+- *(1) HashAggregate(keys=[authors#104, title#103], functions=[], output=[authors#104, title#103])

+- *(1) FileScan csv [title#103,authors#104] Batched: false, Format: CSV, Location: InMemoryFileIndex[file:/C:/Users/ianni/test/books.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<title:string,authors:string>

Στον παρακάτω πίνακα δίδονται μερικά από τα χαρακτηριστικά που είναι διαθέσιμα στα πλάνα της μεθόδου explain() για να διευκολυνθεί ο αναγνώστης στην κατανόησή τους :

Πίνακας 5-2: Μερικά χαρακτηριστικά για την επεξήγηση των πλάνων της explain()

Πηγή: http://dwgeek.com/spark-sql-explain-operator-and-examples.html/

Feature Name	**Description**
Sort	Number of Sort
Exchange rangepartitioning	range partitioning
Project	Number of select statements
SortMergeJoin	Inner Joins
Exchange hashpartitioning	Hash Partitioning
HashAggregate	Aggregate Functions
BroadcastHashJoin	Join condition in case of non co-located tables
Filter	Where condition
HiveTableScan	Table scan operation
MetastoreRelation	Number of relations in given query
BroadcastExchange HashedRelationBroadcastMod	Casting function
TakeOrderedAndProject	WITH Clause feature

Scan OneRowRelation	Single row query without any relation
CollectLimit	Limit Clause in query
Window	Analytics Fucntions
Inner	Inner Join
LeftOuter	left outer join
RightOuter	Right outer join
CartesianProduct	Cartesian or cross Join
BroadcastNestedLoopJoin	Nested loop join
BroadcastExchange IdentityBroadcastMode	Broadcast join
CreateTableCommand	Create table
DropTableCommand	Drop table command
AlterTableRenameCommand	Alter table

Τα Dataframes πριν την έκδοση 1.3 αποκαλούνταν schemaRDD και αποτελούσαν προέκταση της κλάσης RDD έτσι οι πράξεις map και filter των RDD ήταν άμεσα διαθέσιμες σε αυτά. Πλέον αν θέλει ο χρήστης του Spark να χρησιμοποιήσει διάφορους μετασχηματισμούς των RDD πρέπει να τα μετατρέψει σε RDD με τη βοήθεια της μεθόδου .rdd() να χρησιμοποιήσει τους μετασχηματισμούς των RDD που θέλει και ακολούθως να τα μετατρέψει εκ νέου σε DataFrame μέσω της μεθόδου toDF() ονοματίζοντας τις στήλες μέσα σε αγκύλες εντός της παρένθεσης (βλέπε παράδειγμα σελ. 90) ή χρησιμοποιώντας τη μέθοδο createDataFrame(rdd, schema) όπου rdd είναι το RDD που θέλουμε να μετατρέψουμε σε DataFrame και schema είναι το σχήμα. Και με τους δύο τρόπους επιβάλει ο χρήστης ένα σχήμα κατά την ανάγνωση (schema on read). Μπορούν να κατασκευαστούν με μια ποικιλία πηγών δεδομένων που έχουν σχήματα όπως το Hive, το Parquet, το JSON, άλλες πηγές RDBMS, καθώς και από υπάρχοντα RDD. Ένα χαρακτηριστικό των ερωτημάτων SQL είναι ότι επιστρέφουν Dataframes, έτσι ένας χρήστης μπορεί στη συνέχεια να χρησιμοποιήσει προγραμματιστικά μια από τις γλώσσες (Scala, Java, Python και R) που υποστηρίζει το Spark για να ολοκληρώσει την ανάλυσή του. Όπως και στα RDD, οι δύο τύποι πράξεων που μπορεί να κάνει ο χρήστης σε ένα Dataframe είναι: οι μετασχηματισμοί (transformations) και οι δράσεις (actions).

```
df = spark.read.format('json').load('py/test/sql/people.json')
```

ή:

```
df = spark.read.json('py/test/sql/people.json')
```

Το DataFrame:

- Υποστηρίζει ένα ευρύ φάσμα μορφότυπων δεδομένων και συστημάτων αποθήκευσης
- Διαθέτει API για Python, Java, Scala και R.
- Επιτρέπει τη ρητή διαχείριση των ελλειπόντων δεδομένων (missing data).
- Περιλαμβάνει λειτουργίες όπως "επιλογή" γραμμών, στηλών και κελιών με βάση το όνομα ή τον αριθμό, φιλτράρισμα των γραμμών κ.λπ.

Κεφάλαιο 6

Εγκατάσταση του Spark

Πριν την επεξήγηση της διαδικασίας εγκατάστασης του Spark θα γίνει μια εισαγωγή στο τρόπο που σύμφωνα με τον γράφοντα θα πρέπει κανείς να διαχειρίζεται ένα νέο έργο (project). Διευκρινίζεται ότι στο κεφάλαιο αυτό οι όροι χρήστης, κανείς και προγραμματιστής χρησιμοποιούνται εναλλάξιμα επίσης οι εντολές που εκτελούνται στο τερματικό υποδεικνύονται με έντονα μαύρα γράμματα τα δε αποτελέσματα των εντολών αυτών καθώς και τα σχόλια υποδεικνύονται με μπλε γράμματα.

Διαχείριση έργων

Θα πρέπει να αποφύγει κανείς τον πειρασμό να εγκαθιστά νέα πακέτα Python στο εξ ορισμού (default) περιβάλλον της Python, διότι η πρακτική αυτή θα οδηγήσει σε μια εγκατάσταση υπερβολικά περίπλοκη, γιγάντια και δύσκολη στη συντήρηση.

Σύμφωνα με τον γράφοντα ο ενδεδειγμένος τρόπος είναι να δημιουργήσει ο προγραμματιστής ένα εξειδικευμένο περιβάλλον για το έργο και να το αποθηκεύσει ως μέρος του έργου του. Αυτό βοηθά στην απομόνωση των εξαρτήσεων των έργων του από άλλες εργασίες, αλλά ταυτόχρονα καθιστά πολύ πιο εύκολο για τους άλλους να αναπαράγουν το τοπικό περιβάλλον του αρχικού έργου.

Ένας τρόπος για να διαχειρίζεται ένας προγραμματιστής τα περιβάλλοντα της Python είναι να χρησιμοποιήσει το conda, που είναι μέρος της πλατφόρμας Anaconda. Το conda είναι ένα σύστημα διαχείρισης πακέτων γενικού σκοπού. Το Conda έχει σχεδιαστεί για να διαχειρίζεται πακέτα και εξαρτήσεις σε οποιαδήποτε στοίβα λογισμικού. Αυτό το καθιστά ιδιαίτερα χρήσιμο για τη διαχείριση σύνθετων έργων της επιστήμης δεδομένων. Παρακάτω θα περιγραφεί ο τρόπος χρήσης του conda για τη δημιουργία, αποθήκευση και κοινή χρήση νέων περιβαλλόντων έργου.

Δημιουργία νέου καταλόγου για το έργο του προγραμματιστή

Κάθε νέο έργο, ανεξάρτητα από το πόσο μικρό είναι, θα πρέπει να έχει δικό του φάκελο. Ακόμη και τα έργα που ξεκινούν μικρά μπορούν γρήγορα να γιγαντωθούν. Η εξασφάλιση μιας

στέγης για ένα έργο σε μεταγενέστερο χρόνο, είναι μια χρονοβόρα και επίπονη διαδικασία και γι' αυτό είναι προτιμότερο να δοθεί στο έργο μια στέγη από την αρχή.

```
C:\Users\ianni
λ mkdir myproject

C:\Users\ianni
λ cd myproject\

C:\Users\ianni\myproject
λ
```

Το λ που προηγείται της εντολής, είναι το σύμβολο του ***cmder*** (φορητός εξομοιωτής κονσόλας για Windows).

Θα πρέπει το έργο του προγραμματιστή να έχει μια λογική δομή, για το λόγο αυτό θα πρέπει να δημιουργήσει ξεχωριστούς φακέλους για δεδομένα, πηγαίο κώδικα, δοκιμές, τεκμηρίωση κ.λπ.

Δημιουργία νέου Περιβάλλοντος για το έργο του προγραμματιστή

Το επόμενο βήμα στο οποίο πρέπει να προβεί ο προγραμματιστής μετά την δημιουργία του καταλόγου (και των υποκαταλόγων που καθορίζουν τη δομή του) για το έργο του είναι η δημιουργία του περιβάλλοντος του έργου. Η παρακάτω εντολή κάνει αυτό ακριβώς, δημιουργεί ένα περιβάλλον με ξεχωριστό αντίγραφο της Python (3.7) και εγκαθιστά τα απαραίτητα πακέτα. Αν δεν προκαθορίσει ο προγραμματιστής την έκδοση της python που θα χρησιμοποιεί το νέο περιβάλλον, με το όρισμα *python=<έκδοση Python>*, το περιβάλλον που θα δημιουργηθεί θα χρησιμοποιεί την έκδοση της Python που χρησιμοποιεί ο χρήστης την στιγμή της δημιουργίας του. Από προεπιλογή τα περιβάλλοντα conda δημιουργούνται στον κατάλογο *envs* εντός του καταλόγου που έχει εγκαταστήσει το *Anaconda* ο χρήστης.

```
C:\Users\ianni\myproject
λ conda create --name pyspark-env python=3.7
```

Διαγραφή περιβάλλοντος

Η διαγραφή ενός περιβάλλοντος είναι εύκολη:

```
# Εναλλακτικά μπορεί να χρησιμοποιηθεί και η εντολή:
# conda remove --name pyspark_env --all
conda env remove --name pyspark_env
```

Ενεργοποίηση / Απενεργοποίηση του περιβάλλοντος

Η ενεργοποίηση του νέου περιβάλλοντος γίνεται με την ακόλουθη εντολή:

```
C:\Users\ianni\myproject
λ conda activate pyspark_env

C:\Users\ianni\myprojectconda env
(pyspark_env) λ
```

Όπως παρατηρεί κανείς όταν ενεργοποιηθεί ένα περιβάλλον conda στην αρχή της γραμμής εντολών εμφανίζεται το όνομα του (`pyspark_env`).

Για να εγκαταλείψει ο χρήστης το περιβάλλον, μπορεί είτε να ενεργοποιήσει ένα άλλο περιβάλλον (*conda activate new_env*) είτε να απενεργοποιήσει το τρέχον περιβάλλον με την εντολή:

```
(pyspark_env) λ conda deactivate
```

Τέλος, μπορεί κανείς να δει τη λίστα με τα περιβάλλοντα conda που έχουν δημιουργηθεί χρησιμοποιώντας την εντολή:

```
# Εναλλακτικά μπορεί να χρησιμοποιηθεί και η εντολή:
# conda info –envs
C:\Users\ianni\myproject
(pyspark_env) λ conda env list
# conda environments:
#
base                     C:\Users\ianni\Anaconda3
py3                      C:\Users\ianni\Anaconda3\envs\py3
pyspark_env           *  C:\Users\ianni\Anaconda3\envs\pyspark_env
```

Το * δείχνει το ενεργό περιβάλλον

Εγκατάσταση πακέτων

Ένας βασικός λόγος για τη δημιουργία ενός απομονωμένου περιβάλλοντος έργου είναι η διευκόλυνση της εγκατάστασης νέων πακέτων χωρίς να παρεμβαίνει κανείς στην εγκατάσταση Python του συστήματός του ή σε οποιαδήποτε άλλη εγκατάσταση έργου. Για να εγκαταστήσει κανείς νέα πακέτα, μπορεί να χρησιμοποιήσει την εντολή *conda install* μέσα από το ενεργό περιβάλλον:

```
λ conda activate pyspark_env
(pyspark_env) λ conda install pandas
```

ή

Για να εγκαταστήσει ένα συγκεκριμένο πακέτο όπως το *SciPy* σε ένα υπάρχον περιβάλλον π.χ το "*pyspark_env*":

```
λ conda install --name pyspark_env scipy
```

Για να εγκαταστήσει κανείς πολλά πακέτα ταυτόχρονα, όπως το *SciPy* και το *cURL* στο τρέχον περιβάλλον:

```
(pyspark_env) λ conda install scipy curl
```

Για να καταργήσει κανείς ένα πακέτο όπως το *SciPy* στο τρέχον περιβάλλον:

```
(pyspark_env) λ conda remove scipy
```

Σημείωση: Διευκρινίζεται ότι δεν είναι διαθέσιμα όλα τα πακέτα απευθείας μέσω *conda* – μπορεί κανείς να χρησιμοποιήσει την αναζήτηση *conda search <ερώτημα>* για να αναζητήσει αυτά που είναι διαθέσιμα - αλλά είναι επίσης δυνατό να χρησιμοποιήσει το *pip install* της Python για την προσθήκη πακέτων. Για να χρησιμοποιήσει κανείς το *pip* πρέπει πρώτα να το εγκαταστήσει στο νέο του περιβάλλον:

```
(pyspark_env) λ conda install pip
```

Σε αυτό το στάδιο υπάρχουν δύο εκδόσεις του *pip*: μια καθολικά προσβάσιμη που σχετίζεται με την Python του συστήματος και την πρόσφατα εγκατεστημένη στο περιβάλλον *pyspark_env*. Για να χρησιμοποιήσει κανείς το *pip* της τοπικής εγκατάστασης θα πρέπει να εκτελέσει την εντολή *pip* μέσα απ' το τοπικό περιβάλλον:

```
C:\ Users\ianni
(pyspark_env) λ echo %CONDA_PREFIX%
C:\Users\ianni\Anaconda3\envs\pyspark_env

#Για Linux : echo $CONDA_PREFIX

C:\Users\ianni
λ which pip
C:\Users\ianni\AppData\Local\Microsoft\WindowsApps\pip.EXE

C:\Users\ianni
(base) λ which pip
C:\Users\ianni\Anaconda3\Scripts\pip.EXE

C:\Users\ianni
(pyspark_env) λ which pip
C:\Users\ianni\Anaconda3\envs\pyspark_env\Scripts\pip.EXE

C:\Users\ianni
(pyspark_env) λ pip --version
pip  19.2.2  from  C:\Users\ianni\Anaconda3\envs\pyspark_env\lib\site-packages\pip
(python 3.7)

C:\Users\ianni
(pyspark_env) λ pip install pandas

C:\Users\ianni
(pyspark_env) λ conda list pandas
# packages in environment at C:\Users\ianni\Anaconda3\envs\pyspark_env:
#
# Name                    Version                   Build  Channel
pandas                    0.25.1                   pypi_0    pypi
```

Για να καταργήσει κανείς την εγκατάσταση ενός πακέτου που έχει εγκαταστήσει με το *pip*:

```
C:\Users\ianni
(pyspark_env) λ pip uninstall pandas

 C:\Users\ianni
(pyspark_env) λ conda list pandas
# packages in environment at C:\Users\ianni\Anaconda3\envs\pyspark_env:
#
# Name                    Version                   Build  Channel

C:\Users\ianni
(pyspark_env) λ
```

Προβολή της λίστας των εγκατεστημένων πακέτων σε ένα περιβάλλον

Για να δει κανείς μια λίστα με όλα τα πακέτα που είναι εγκατεστημένα σε ένα συγκεκριμένο περιβάλλον:

```
# Εάν το περιβάλλον δεν είναι ενεργοποιημένο
C:\Users\ianni
(pyspark_env) λ conda list -n myenv
# Εάν το περιβάλλον είναι ενεργοποιημένο
C:\Users\ianni
```

```
(pyspark_env) λ conda list
```

Αποθήκευση περιβάλλοντος

Αφού δημιουργήσει κανείς το νέο του περιβάλλον, το ενεργοποιήσει και εγκαταστήσει όλα τα πακέτα που θα χρειαστεί για το έργο του, αξίζει να πραγματοποιήσει ένα τελευταίο βήμα: να αποθηκεύσει τις προδιαγραφές περιβάλλοντος σε ένα αρχείο μέσα στο φάκελο του έργου του. Με αυτόν τον τρόπο καθιστά το έργο του φορητό και επαναχρησιμοποιήσιμο, επειδή αυτό το αρχείο προδιαγραφών περιβάλλοντος μπορεί να χρησιμοποιηθεί από αυτόν ή και άλλους χρήστες για να αναδημιουργήσει το ακριβές περιβάλλον του εάν επιθυμεί να ανακατασκευάσει /ή και τρέξει το έργο του.

Χρησιμοποιώντας κανείς την εντολή *conda export* μπορεί να αποθηκεύσει σε ένα αρχείο προδιαγραφής *yaml* (αρχείο που περιέχει τις λεπτομερείς εξαρτήσεις του περιβάλλοντός του) το τρέχον περιβάλλον conda.

```
C:\Users\ianni\myproject
(pyspark_env) λ conda env export > my_env.yml

C:\Users\ianni\myproject
(pyspark_env) λ more my_env.yml
name: pyspark_env
channels:
  - conda-forge
  - anaconda-fusion
  - defaults
dependencies:
  - ca-certificates=2019.6.16=hecc5488_0
  - certifi=2019.6.16=py37_1
  - openssl=1.1.1c=hfa6e2cd_0
  - pip=19.2.2=py37_0
  - python=3.7.3=h510b542_1
  - setuptools=41.2.0=py37_0
  - sqlite=3.29.0=hfa6e2cd_0
  - vc=14.1=h0510ff6_4
  - vs2015_runtime=14.15.26706=h3a45250_4
  - wheel=0.33.6=py37_0
  - wincertstore=0.2=py37_1002
  - pip:
    - numpy==1.17.0
    - python-dateutil==2.8.0
    - pytz==2019.2
    - six==1.12.0
prefix: C:\Users\ianni\Anaconda3\envs\pyspark_env
```

Κατόπιν μπορεί να χρησιμοποιήσει το αρχείο *my_env.yml*, για να αναδημιουργήσει το περιβάλλον του σε κάποιον άλλο υπολογιστή.

```
λ conda env create -f my_env.yml
```

Θα πρέπει ο χρήστης να ενεργοποιήσει το περιβάλλον όταν εργάζεται στο συγκεκριμένο έργο και να το απενεργοποιήσει όταν τελειώσει. Αν χρειαστεί να εγκαταστήσει πρόσθετα πακέτα τότε πρέπει επίσης να θυμάται να δημιουργήσει ξανά το αρχείο *my_env.yml* για να συμπεριλάβει τις αλλαγές αυτές.

PySpark

Όπως προαναφέρθηκε το Spark αναπτύσσεται στη γλώσσα προγραμματισμού Scala. Το Apache Spark έχει API για τις γλώσσες προγραμματισμού Python, Scala, Java και R. Το Pyspark είναι το Python API για το Spark που επιτρέπει στον προγραμματιστή να χρησιμοποιήσει το Spark μέσα από την Python. Ο λόγος που επιλέχθηκε η Python ως η γλώσσα προγραμματισμού για την παρούσα διπλωματική εργασία είναι ότι είναι μία από τις πιο διαδεδομένες γλώσσες προγραμματισμού μεταξύ των επιστημόνων δεδομένων που την χρησιμοποιούν για να πραγματοποιούν ανάλυση δεδομένων, μηχανική μάθηση αλλά και πολλές άλλες εργασίες σε μεγάλα δεδομένα. Είναι μια γλώσσα προγραμματισμού γενικού σκοπού, με κύρια χαρακτηριστικά της την απλότητα, την ευελιξία, αλλά και την ευκολία στην εκμάθησή της. Χρησιμοποιείται ευρέως στην ανάπτυξη πολύπλοκων αριθμητικών και επιστημονικών εφαρμογών. Έτσι, είναι αρκετά προφανές ότι ο συνδυασμός του Spark και της Python είναι ιδανικός για κάποιον που θέλει να μυηθεί στον κόσμο των μεγάλων δεδομένων. Βέβαια μπορούμε να πούμε ότι αυτή την στιγμή η απόλυτη γλώσσα για το Spark (μετά κυρίως την εισαγωγή των Datasets) είναι η Scala και για αυτό θα αποτελέσει για τον γράφοντα τον επόμενο στόχο σ' αυτό το ταξίδι στον κόσμο των μεγάλων δεδομένων.

Εγκατάσταση του PySpark στα Windows 10 σε τοπική λειτουργία (local mode)

Για περισσότερες πληροφορίες σχετικά με την τοπική λειτουργία του Spark θα πρέπει να ανατρέξει ο χρήστης στην ενότητα "Τρέξιμο της εφαρμογής Spark τοπικά (local mode)". Τα περισσότερα ζητήματα στην εγκατάσταση του Pyspark προκαλούνται από μη κατάλληλα καθορισμένες μεταβλητές περιβάλλοντος, για το λόγο αυτό θα πρέπει κανείς να είναι ιδιαίτερα προσεκτικός κατά τον καθορισμό τους. Ένα άλλο σύνολο προβλημάτων οφείλεται στο αρχείο winutils.exe, το οποίο αποτελεί συστατικό στοιχείο του hadoop για το λειτουργικό σύστημα των Windows. Χρησιμοποιείται για την εκτέλεση εντολών κελύφους και την πρόσβαση σε τοπικά αρχεία.

Βήμα 1ο :Λήψη και Εγκατάσταση της Java 8

Έλεγχος της εγκατεστημένης έκδοσης

Είναι απόλυτα εξακριβωμένο ότι η έκδοση αυτή θα απαλλάξει τον χρήστη από πολλά προβλήματα που δημιουργούν οι μεταγενέστερες εκδόσεις, γι' αυτό θα πρέπει να βεβαιωθεί ότι είναι εγκατεστημένη στο σύστημά του. Ο έλεγχος αυτός πραγματοποιείται πληκτρολογώντας στη γραμμή εντολών την εντολή:

```
C:\Users\ianni\myproject
(pyspark_env) λ Java -version
```

Το λ που προηγείται της εντολής, είναι το σύμβολο του cmder (φορητός εξομοιωτής κονσόλας για Windows Με έντονο μαύρο είναι η εντολή που πληκτρολογεί ο χρήστης.

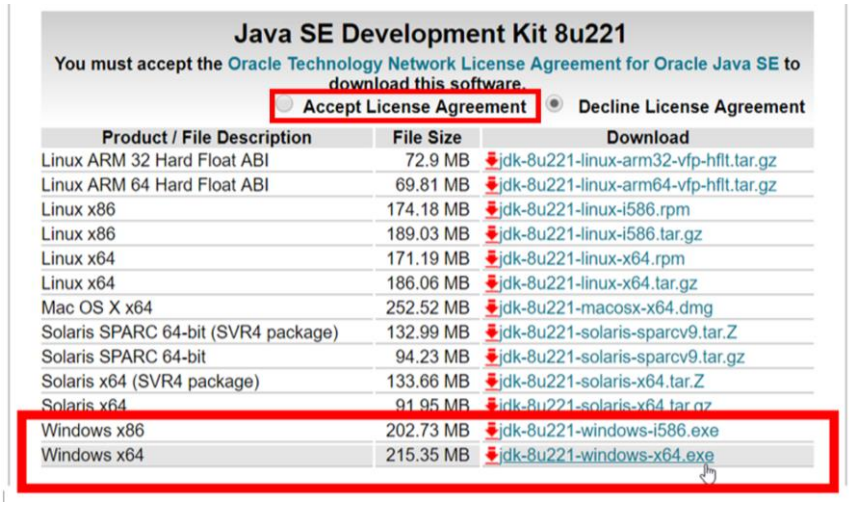
Θα πρέπει να δει ένα μήνυμα παρόμοιο με το παρακάτω:

```
java version "1.8.0_221"
Java(TM) SE Runtime Environment (build 1.8.0_221-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.221-b11, mixed mode)
```

Αν έχει ήδη εγκατεστημένη την έκδοση αυτή, θα πρέπει να ελέγξει ότι έχει ορίσει τις μεταβλητές περιβάλλοντος που υποδεικνύονται στο τέλος του επόμενου βήματος.

Εγκατάσταση της Java 8

Θα πρέπει να μεταβεί ο χρήστης στην ιστοσελίδα λήψεων του επίσημου ιστότοπου της Java, και αφού αποδεχτεί την άδεια χρήσης της *Oracle* να κατεβάσει το *Java JDK 8*, που είναι κατάλληλο για το σύστημά του (θα χρειαστεί να κάνει πρώτα εγγραφή, σύμφωνα με τους νέους όρους της *Oracle*, για να του επιτραπεί η μεταφόρτωση του αρχείου).



Σχήμα 6-1: Ο χρήστης πρέπει να αποδεχθεί πρώτα την άδεια χρήσης πριν την μεταφόρτωση του αρχείου που τον ενδιαφέρει (Windows x86 / Windows x64)

Αφού κάνει τη λήψη του αρχείου, θα εκκινήσει το πρόγραμμα εγκατάστασης του JDK 8 κάνοντας διπλό κλικ στο εικονίδιο του προγράμματος εγκατάστασης ή στο όνομα του αρχείου εγκατάστασης στη θέση λήψης (πρέπει να έχει δικαιώματα διαχειριστή). Για παράδειγμα στο σύστημα του γράφοντα η εγκατάσταση έγινε από τον φάκελο "Στοιχεία λήψης" (βλέπε παρακάτω εικόνα) :

Σχήμα 6-2: Εγκατάσταση του jdk με διπλό κλικ στο όνομα του αρχείου

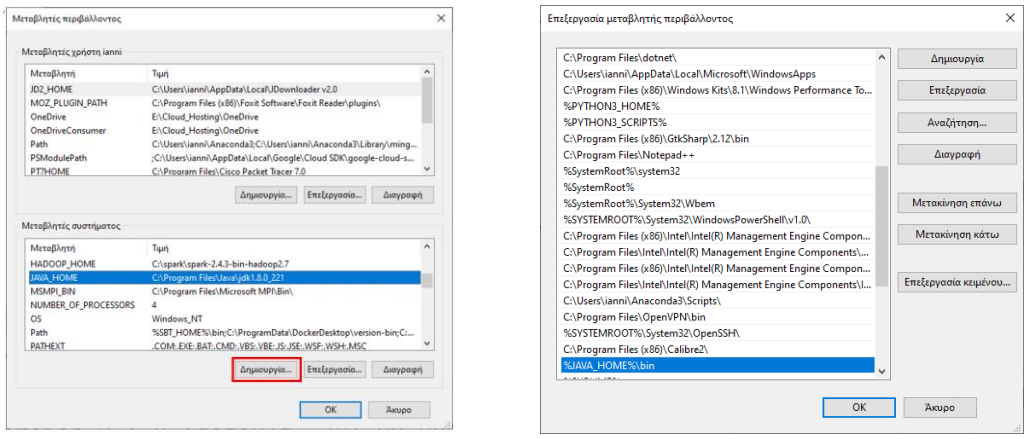
Ακολούθως θα πρέπει ο χρήστης να προσθέσει τις παρακάτω μεταβλητές περιβάλλοντος (συστήματος), μέσα από το παράθυρο μεταβλητές περιβάλλοντος (στον file explorer των Windows επιλέγει, **Αυτός ο υπολογιστής** -> δεξί κλικ ποντικιού, επιλογή **ιδιότητες** -> **Ρυθμίσεις συστήματος για προχωρημένους** -> **Μεταβλητές περιβάλλοντος**):

Θα πρέπει να πατήσει το κουμπί "*Δημιουργία*" στις "*Μεταβλητές συστήματος*" και στο παράθυρο που θα ανοίξει θα πρέπει να συμπληρώσει ως " *Όνομα μεταβλητής*" το JAVA_HOME και ως " Τιμή μεταβλητής " το *C:\Program Files\Java\jdk1.8.0_221*

```
JAVA_HOME = C:\Program Files\Java\jdk1.8.0_221
```

Ακολούθως θα πρέπει να κάνει διπλό κλικ στη μεταβλητή συστήματος **Path** και να προσθέσει το ακόλουθο μονοπάτι (path):

```
C:\Program Files\Java\jdk1.8.0_221\bin
ή
%JAVA_HOME%\bin
```



(α) (β)

Σχήμα 6-3: (α) Δημιουργία της μεταβλητής περιβάλλοντος JAVA_HOME σε επίπεδο συστήματος

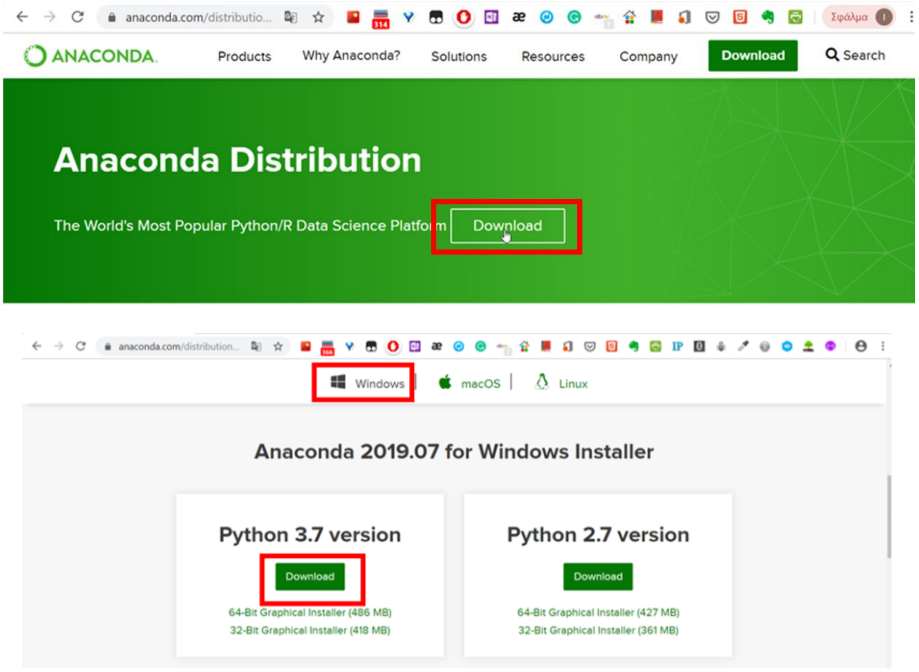
Βήμα 2ο: Λήψη και Εγκατάσταση του Anaconda (για python)

Ανεξάρτητα από το αν έχει κανείς εγκατεστημένη ή όχι την επίσημη διανομή CPython που διατίθεται από το python.org (έλεγχος με εντολές *where python*, *python3 --version* βλέπε Σχήμα 6.6α), θα ήταν προτιμότερο να εγκαταστήσει και το Anaconda, μια ελεύθερη και ανοικτού κώδικα διανομή των γλωσσών προγραμματισμού Python και R με πολλά πλεονεκτήματα που αποσκοπεί στην απλοποίηση της διαχείρισης και της ανάπτυξης πακέτων. "Η διανομή Anaconda περιλαμβάνει περισσότερα από 1.500 πακέτα, καθώς και τον διαχειριστή πακέτων και εικονικών περιβαλλόντων conda, που χειρίζεται την εγκατάσταση πακέτων Python αλλά και εξωτερικών απαιτήσεων λογισμικού τρίτων κατασκευαστών. Περιλαμβάνει επίσης ένα γραφικό περιβάλλον, το Anaconda Navigator, ως γραφική εναλλακτική λύση στη διεπαφή γραμμής εντολών (CLI)" [28]. Τα εικονικά περιβάλλοντα επιτρέπουν σε κάποιον να εγκαταστήσει συγκεκριμένες εκδόσεις πακέτων για ένα συγκεκριμένο έργο χωρίς να ανησυχεί για συγκρούσεις εκδόσεων.

Με τη λήψη του Anaconda, αποκτά κανείς πρόσβαση στα **conda**, **Python**, **Jupyter Notebook** και εκατοντάδες άλλα πακέτα ανοιχτού κώδικα.

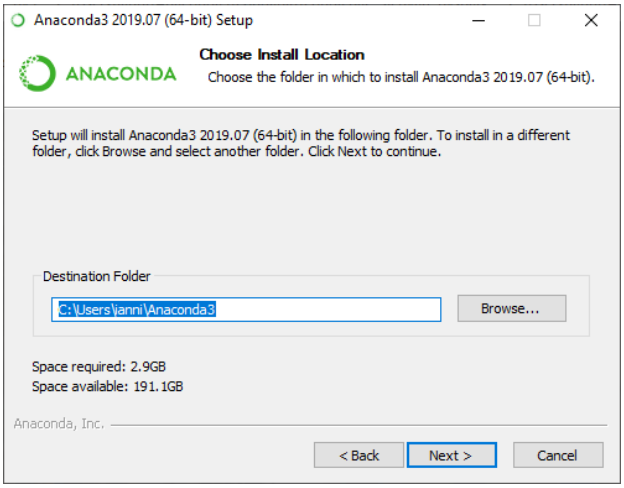
Συνιστάται η μεταφόρτωση και εκτύπωση του Anaconda Cheatsheet.

Για τη λήψη του Anaconda θα πρέπει να μεταβεί κανείς στην ιστοσελίδα λήψεων του επίσημου ιστότοπου του Anaconda , να επιλέξει την έκδοση για windows και να κατεβάσει την πιο πρόσφατη έκδοση (τη στιγμή που γράφονταν η διπλωματική η πιο πρόσφατη έκδοση ήταν η 3.7).

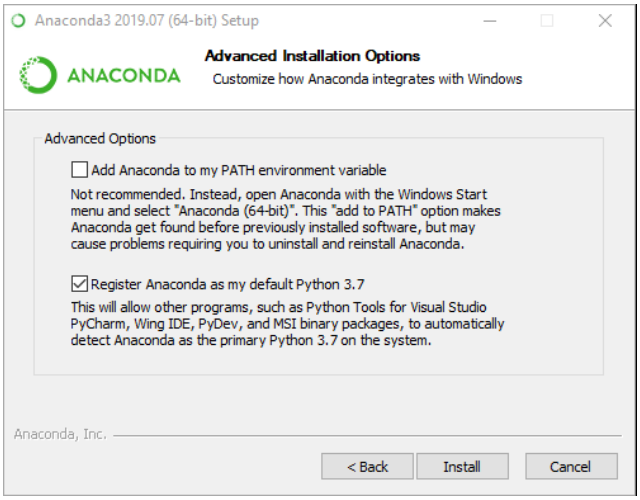


Σχήμα 6-4: Βήματα για την μεταφόρτωση του αρχείου Anaconda 2019.07 (Python 3.7 version) για Windows OS

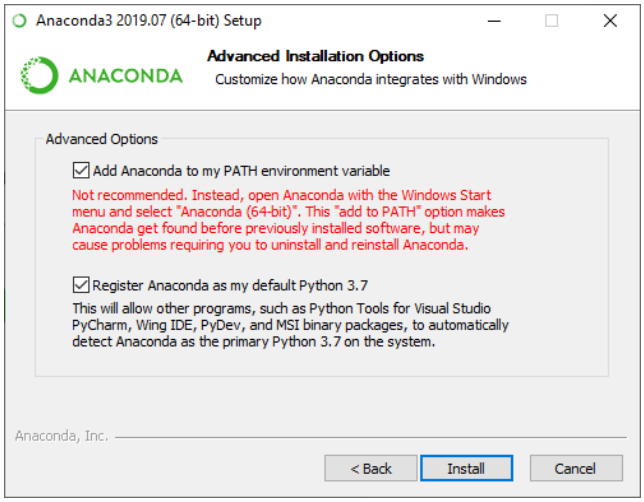
Αφού εντοπίσει τη λήψη θα πρέπει να κάνει διπλό κλικ σε αυτήν. Κατόπιν θα πρέπει επιλέξει να εγκατασταθεί μόνο σε αυτόν (επιλογή - just me). Θα πρέπει να ρυθμίσει την τοποθεσία εγκατάστασης (βλέπε Σχήμα 6.5α) και να κάνει κλικ στο κουμπί Επόμενο / Next. Ακολουθεί ένα σημαντικό μέρος της διαδικασίας εγκατάστασης. Η συνιστώμενη προσέγγιση είναι να μην επιλέξει το πλαίσιο ελέγχου για την προσθήκη το Anaconda στην μεταβλητή περιβάλλοντος *PATH* (βλέπε Σχήμα 6.5β). Αυτό σημαίνει ότι θα πρέπει να χρησιμοποιήσει το Anaconda Navigator ή το Anaconda Command Prompt ή το Anaconda Powershell Prompt (που βρίσκονται στο μενού "Έναρξη" κάτω από το "Anaconda3 (64-bit)") όταν θα θέλει να χρησιμοποιήσει το Anaconda (μπορεί πάντα να προσθέσει το Anaconda στο *PATH* του αργότερα, αν δεν το επιλέξει τώρα). Εάν θέλει να μπορεί να χρησιμοποιήσει το Anaconda στη γραμμή εντολών (ή git bash, **cmder**, powershell κ.λπ.), θα πρέπει να επιλέξει το πλαίσιο ελέγχου δίπλα στην πρώτη επιλογή (βλέπε Σχήμα 6.5γ). Στο παράδειγμα που ακολουθεί ο γράφων επέλεξε τη δεύτερη προσέγγιση (βλέπε Σχήμα 6.5γ) για να μπορεί να χρησιμοποιήσει το Anaconda μέσα στο cmder (φορητός εξομοιωτής κονσόλας για Windows).



(α)



(β)



(γ)

Σχήμα 6-5: (α) Επιλογή φακέλου προορισμού, (β) Η εξορισμού επιλογή είναι να μην προστεθεί το Anaconda στο Path (αποεπιλεγμένο πλαίσιο ελέγχου), (γ) Προσθήκη του Anaconda στο Path

(γ) Αφού ολοκληρωθεί η εγκατάσταση, θα πρέπει να ελέγξει την ορθότητάς της. Θα πρέπει να κλείσει τη γραμμή εντολών εάν ήταν ήδη ανοικτή, στη συνέχεια να ανοίξει μια νέα και να ελέγξει αν μπορεί να εκτελέσει με επιτυχία την εντολή *python --version* στην περίπτωση που δεν

είχε εγκαταστήσει την επίσημη διανομή της Python (CPython) (βλέπε Σχήμα 6.6β). Σε διαφορετική περίπτωση πληκτρολογεί (βλέπε Σχήμα 6.6γ):

```
conda activate base
python --version
```

```
C:\Users\ianni
λ where python
C:\Users\ianni\Anaconda3\python.exe
C:\Users\ianni\AppData\Local\Microsoft\WindowsApps\python.exe
```

(a)

```
C:\Users\ianni
λ python --version
Python 3.7.3
```

(β)

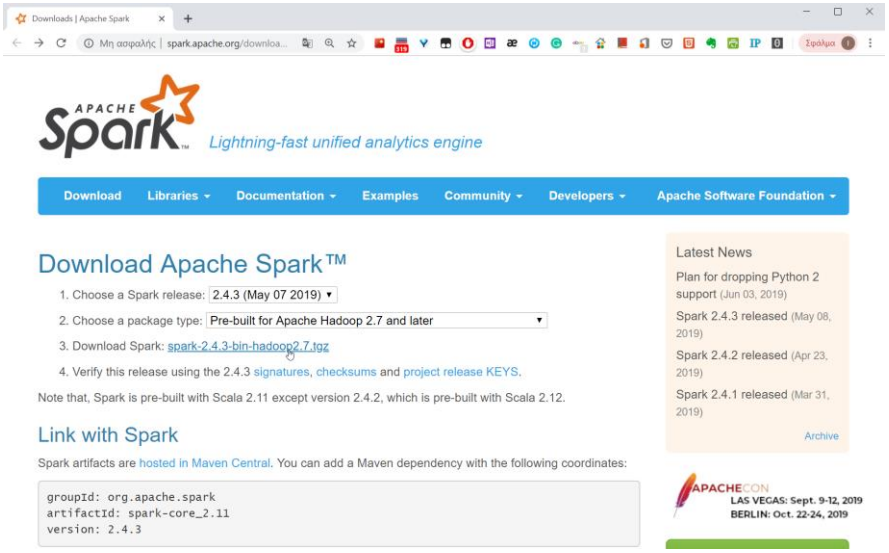
```
C:\Users\ianni
λ conda activate base

C:\Users\ianni
(base) λ python --version
Python 3.7.3
```

(γ)

Σχήμα 6-6: Έλεγχος ορθής εγκατάστασης Anaconda

Βήμα 3ο: Λήψη και εγκατάσταση του Spark

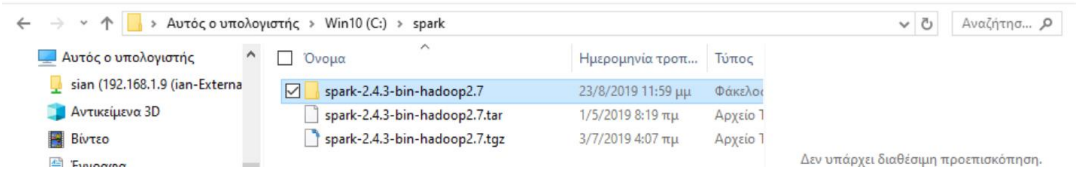
Πηγαίνοντας στην ιστοσελίδα λήψεων του επίσημου ιστότοπου Spark μπορεί ο χρήστης να κατεβάσει την πιο πρόσφατη έκδοση (2.4.3) του αρχείου με επέκταση .tgz (βλέπε Σχήμα 6.7).



Σχήμα 6-7: Επιλογή της έκδοσης Spark (prebuilt for Apache Hadoop 2.7 and later) που επιθυμεί ο χρήστης

Ένα αρχείο με επέκταση .tgz ή .tar.gz είναι ένα TAR Archive αρχείο που έχει συμπιεστεί χρησιμοποιώντας το λογισμικό Gnu Zip (gzip). Με το 7z μπορεί κανείς σε πρώτο βήμα να το αποσυμπιέσει (.tgz -> .tar) και σε δεύτερο βήμα να εξάγει τη συλλογή αρχείων ή φάκελων από το TAR στον κατάλογο που έχει επιλέξει. Στη συνέχεια μεταφέρει το αρχείο *spark-2.4.3-bin-hadoop2.7.tgz* από τον φάκελο "*Στοιχεία λήψεις*" στον φάκελο "*spark*" που θα έχει δημιουργήσει

στη διαδρομή *C:\spark* (βλέπε Σχήμα 6.8) και εξάγει με το 7z (πατώντας δεξί πλήκτρο και επιλέγοντας "αποσυμπίεση εδώ") το αρχείο *spark-2.4.3-bin-hadoop2.7.tar*. Ακολούθως πατώντας δεξί πλήκτρο και επιλέγοντας "αποσυμπίεση εδώ" στο αρχείο *tar* εξάγει τον φάκελο *spark-2.4.3-bin-hadoop2.7*. Τέλος μπορεί αν το επιθυμεί να διαγράψει τα αρχεία με επέκταση .tar και .tgz.



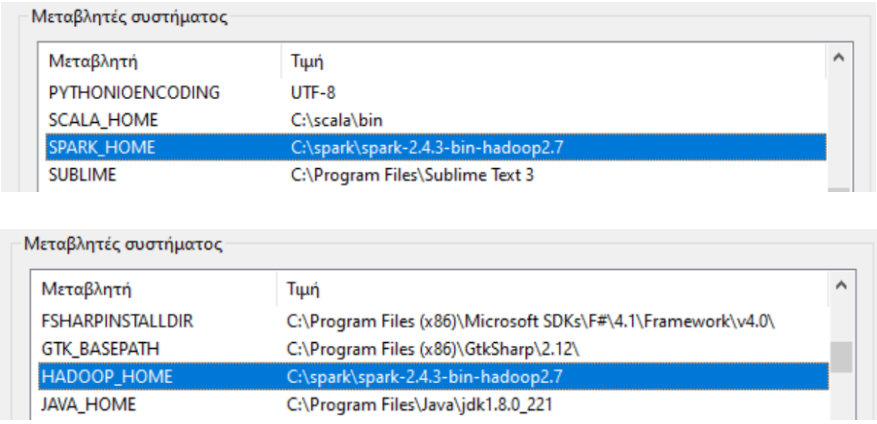
Σχήμα 6-8: Μεταφορά του αρχείου spark-2.4.3-bin-hadoop2.7.tgz στη θέση C:\spark και ακολούθως αποσυμπίεση και εξαγωγή με την εφαρμογή 7z στον ομώνυμο φάκελο spark-2.4.3-bin-hadoop2.7

Στη συνέχεια θα πρέπει να δημιουργήσει τις μεταβλητές περιβάλλοντος (στις μεταβλητές συστήματος - βλέπε Σχήμα 6.9) :

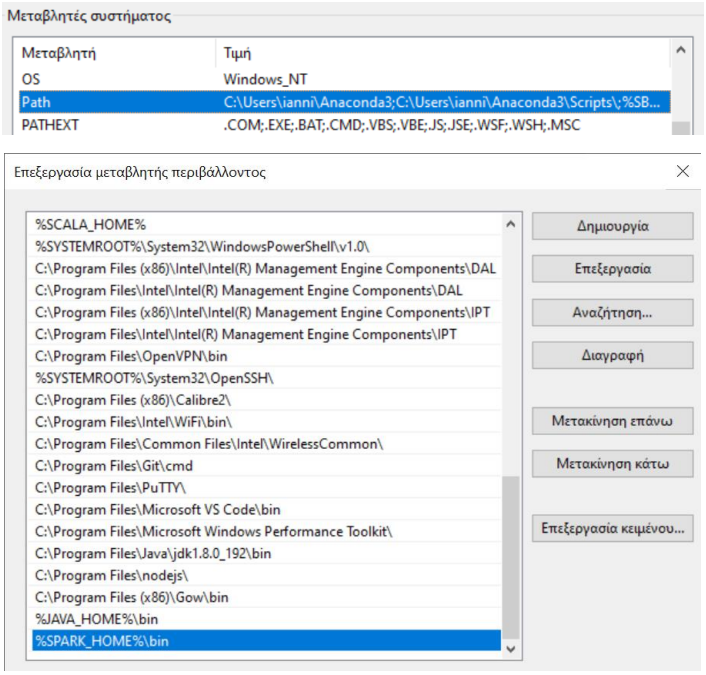
```
SPARK_HOME = C:\spark\spark-2.4.3-bin-hadoop2.7
HADOOP_HOME = C:\spark\spark-2.4.3-bin-hadoop2.7
```

Κατόπιν θα πρέπει να κάνει διπλό κλικ στη μεταβλητή περιβάλλοντος (συστήματος) *Path* και να προσθέσει την ακόλουθη διαδρομή (βλέπε Σχήμα 6.10):

```
C:\spark\spark-2.4.3-bin-hadoop2.7\bin
   ή
%SPARK_HOME%\bin
```



Σχήμα 6-9: Καθορισμός των μεταβλητών περιβάλλοντος HADOOP_HOME και SPARK_HOME



Σχήμα 6-10: Προσθήκη στη μεταβλητή περιβάλλοντος Path, της διαδρομής %SPARK_HOME%\bin

Βήμα 4ο: Λήψη και ρύθμιση του winutils.exe

Από το αποθετήριο του hadoop, https://github.com/steveloughran/winutils θα πρέπει ο χρήστης να κάνει λήψη του αρχείου winutils.exe (ο σύνδεσμος παραπέμπει στην έκδοση 2.7). Στην περίπτωση του γράφοντα η λήψη του αρχείου δεν ευοδώθηκε με το Google Chrome καθώς εσφαλμένα το αναγνώρισε ως αρχείο προερχόμενο από site κακόβουλου λογισμικού. Αντίθετα δεν υπήρξε κανένα πρόβλημα με τα Microsoft Edge και Mozilla Firefox που το κατέβασαν αδιαμαρτύρητα ενώ στη συνέχεια ελέγχθηκε για τυχόν ιούς με το Windows Defender και το IObit Malware Fighter 7.1 Free.

Θα πρέπει ο χρήστης να αποθηκεύσει το winutils.exe στον κατάλογο bin της εγκατάστασης του Spark (*%SPARK_HOME%\bin*). Στην περίπτωση του γράφοντα ο φάκελος *bin* είναι στη διαδρομή:

```
C:\Users\ianni
λ echo %SPARK_HOME%\ bin
C:\spark\spark-2.4.3-bin-hadoop2.7\bin
```

Με γαλάζιο χρώμα δεικνύεται η απάντηση του συστήματος – με έντονα μαύρα η εντολή που εισάγουμε.

Ένα επιπλέον βήμα που πιθανόν να γλυτώσει τον χρήστη από προβλήματα, όπως συνέβη και στην περίπτωση του γράφοντα και που συνιστάται από πολλούς οδηγούς εγκατάστασης του Spark, είναι το ακόλουθο:

Θα πρέπει να δημιουργήσει το φάκελο C:\tmp\hive και να εκτελέσει ως διαχειριστής τις ακόλουθες εντολές στη γραμμή εντολών του cmd (μέσα από τον κατάλογο *bin* – *C:\spark\spark-2.4.3-bin-hadoop2.7\bin*):

```
winutils.exe chmod -R 777 C:\tmp\hive
```

```
winutils.exe ls -F C:\tmp\hive
```

Η εκτέλεση της δεύτερης εντολής θα εμφανίσει στην οθόνη του χρήστη κάτι παρόμοιο με το ακόλουθο:

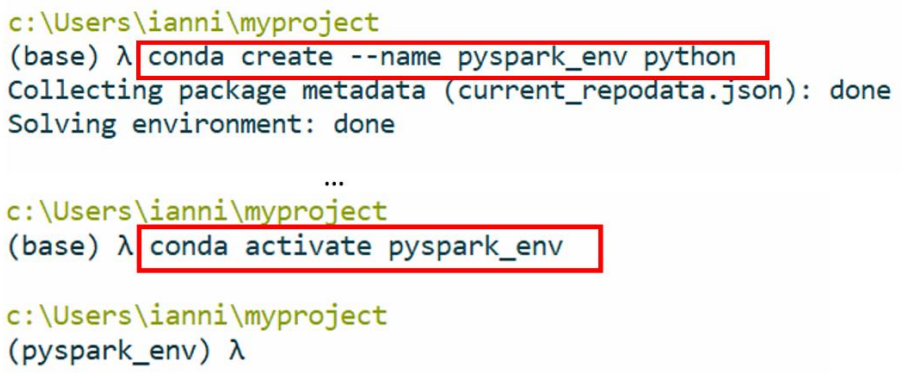
```
drwxrwxrwx|1|BUILTIN\Administrators|MYWIN-PC\ianni|0|Apr| 8|2019|C:\tmp\hive
```

Βήμα 5ο: Έλεγχος της εγκατάστασης του PySpark

Θα πρέπει ο χρήστης να δημιουργήσει πρώτα ένα περιβάλλον conda για το έργο του, από το οποίο θα τρέχει το Jupyter Notebook (στην περίπτωσή του γράφοντα το pyspark_env). Στην ενότητα "Διαχείριση έργων" δίδονται αναλυτικές πληροφορίες σχετικά με τα περιβάλλοντα Conda. Για το λόγο αυτό θα πρέπει να ανοίξει τη γραμμή εντολών και να πληκτρολογήσει (βλέπε Σχήμα 6.11):

```
#Δημιουργία περιβάλλοντος conda με ονομασία pyspark_env
conda create --name pyspark_env python

#Ενεργοποίηση του ανωτέρω περιβάλλοντος conda
conda activate pyspark_env
```

εισάγετε μόνο τα έντονα μαύρα γράμματα.



Σχήμα 6-11: Δημιουργία περιβάλλοντος conda με ονομασία pyspark_env και ακολούθως ενεργοποίηση του περιβάλλοντος

Στη συνέχεια θα πρέπει να πληκτρολογήσει *pyspark*, για να μπει στο *shell* (κέλυφος) του *pyspark*. Θα πρέπει να δει τα εξής:

```
pyspark
```

Σχήμα 6-12: Έναρξη του κελύφους pyspark

Εκτελώντας τις εντολές που φαίνονται στο Σχήμα 6.13, η έξοδος πρέπει να είναι:

```
[(1, 11), (2, 12), (3, 13), (4, 14), (5, 15), (6, 16), (7, 17), (8, 18), (9, 19),
(10, 20)].
```

```
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.3
      /_/

Using Python version 3.7.3 (default, Jul  1 2019 22:01:29)
SparkSession available as 'spark'.
>>> RDD1 = sc.parallelize (range (1,11))
>>> RDD2 = sc.parallelize (range (11,21))
>>> RDD3 = RDD1.zip(RDD2)
>>> RDD3.collect()
[(1, 11), (2, 12), (3, 13), (4, 14), (5, 15), (6, 16), (7, 17), (8, 18), (9, 19), (10, 20)]
>>>
```

Σχήμα 6-13: Έλεγχος του Spark με ένα σύντομο πρόγραμμα με RDDs

Για να βγει κανείς από το κέλυφος του pyspark θα πρέπει να πληκτρολογήσει:

```
exit()
```

Βήμα 6ο: Το PySpark μέσω του Jupyter Notebook

Σύμφωνα με τον επίσημο ιστότοπό του, "το Jupyter Notebook είναι μια εφαρμογή web ανοιχτού κώδικα που επιτρέπει τον χρήστη να δημιουργεί και να μοιράζει έγγραφα που περιέχουν "ζωντανό" κώδικα, εξισώσεις, οπτικοποιήσεις και κείμενο αφήγησης. Οι χρήσεις του περιλαμβάνουν: καθαρισμό και μετασχηματισμό δεδομένων, αριθμητική προσομοίωση, στατιστική μοντελοποίηση, οπτικοποίηση δεδομένων, μηχανική μάθηση και πολλά άλλα" [29]. Συνδυάζοντας το PySpark με το Jupyter Notebook, μπορεί ο εκάστοτε χρήστης να υλοποιήσει όλες τις παραπάνω ροές εργασίας.

Υπάρχουν δύο τρόποι για να έχει ο χρήστης το PySpark διαθέσιμο σε ένα Jupyter Notebook:

- Ρυθμίζοντας το πρόγραμμα οδήγησης PySpark να χρησιμοποιεί το Jupyter Notebook (βλέπε μέθοδος 1 παρακάτω), μπορεί ο χρήστης εκτελώντας την εντολή pyspark να ανοίγει αυτόματα στον προεπιλεγμένο φυλλομετρητή του ένα Jupyter Notebook.
- Ανοίγοντας ένα Jupyter Notebook και ακολούθως φορτώνοντας το PySpark χρησιμοποιώντας το πακέτο findSpark.

Η πρώτη επιλογή είναι πιο γρήγορη αλλά η υλοποίησή της είναι συγκεκριμένη για το Jupyter Notebook, ενώ η δεύτερη επιλογή είναι μια ευρύτερη προσέγγιση για να καταστήσει ο χρήστης το PySpark διαθέσιμο στο IDE της επιλογής του.

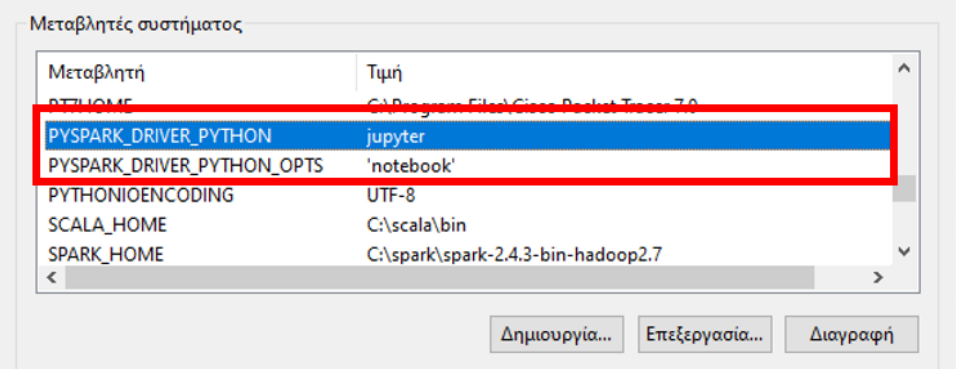
Σημείωση: Ο χρήστης θα πρέπει να επιλέξει μια από τις δυο μεθόδους. Αν για παράδειγμα έχει υλοποιήσει την πρώτη μέθοδο και στη συνέχεια θέλει στη συνέχεια να υλοποιήσει την δεύτερη, θα πρέπει να αφαιρέσει τις μεταβλητές περιβάλλοντος που είχε δημιουργήσει στο βήμα της μεθόδου 1 (PYSPARK_DRIVER_PYTHON και

PYSPARK_DRIVER_PYTHON_OPTS) και ακολούθως να ανοίξει ένα νέο παράθυρο τερματικού και να ακολουθήσει τα βήματα της μεθόδου 2.

Μέθοδος 1 - Ρύθμιση του προγράμματος οδήγησης PySpark

Θα πρέπει ο χρήστης να καθορίσει τις μεταβλητές περιβάλλοντος του προγράμματος οδήγησης PySpark (PYSPARK_DRIVER_PYTHON και PYSPARK_DRIVER_OPTS), μέσα από το παράθυρο μεταβλητές περιβάλλοντος (στον *file explorer* των *Windows* επιλέγει, "***Αυτός ο υπολογιστής***" -> δεξί κλικ ποντικιού, επιλογή **"ιδιότητες"** -> "**Ρυθμίσεις συστήματος για προχωρημένους**" -> "**Μεταβλητές περιβάλλοντος**") (βλέπε Σχήμα 6.14):

```
PYSPARK_DRIVER_PYTHON = jupyter
PYSPARK_DRIVER_PYTHON_OPTS = 'notebook'
```



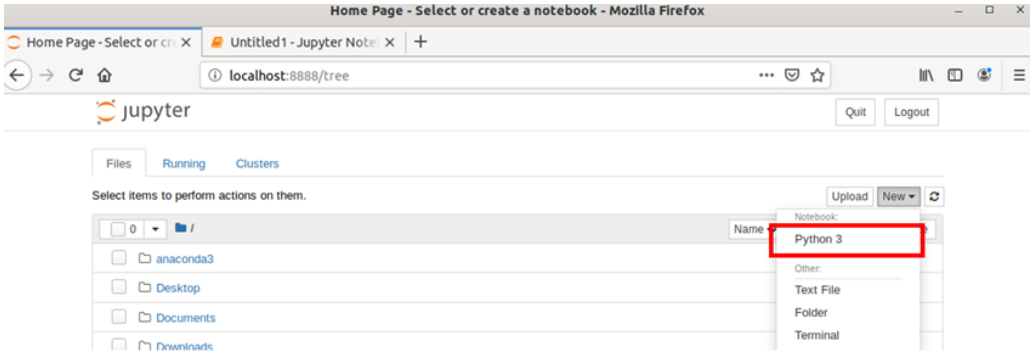
Σχήμα 6-14: Καθορισμός μεταβλητών περιβάλλοντος σε επίπεδο συστήματος

Αφού κλείσει και στην συνέχεια ανοίξει εκ νέου την γραμμή εντολών, θα πρέπει να πληκτρολογήσει *pyspark* για να ανοίξει το jupyter notebook.

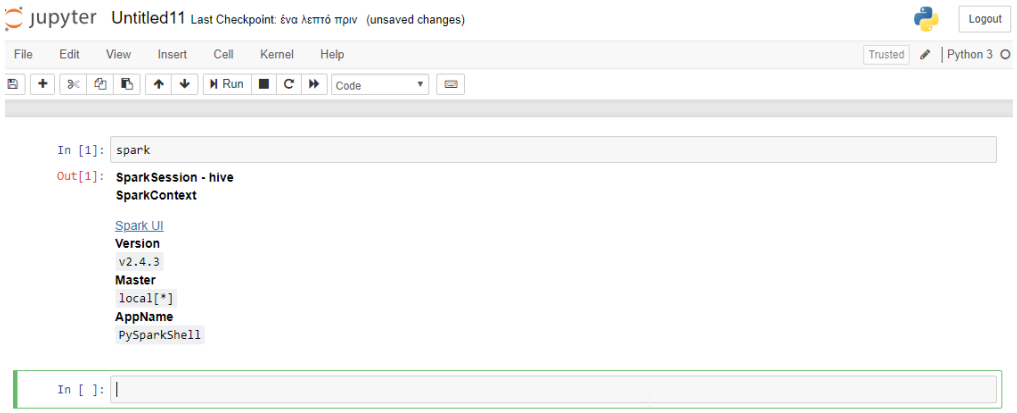
```
conda activate pyspark_env
pyspark
```

Μέσα στη νέα καρτέλα με ονομασία Home Page που θα ανοίξει στον φυλλομετρητή, θα πρέπει να επιλέξει την καρτέλα *Files* (βλέπε Σχήμα 6.15α). Στο δεξί μέρος της υπάρχει η αναδυόμενη λίστα *New* στην οποία επιλέγει τον πυρήνα *python 3* για να δημιουργήσει ένα νέο python σημειωματάριο. Στο νέο σημειωματάριο που θα δημιουργηθεί (βλέπε Σχήμα 6.15β) θα πρέπει να πληκτρολογήσει τον ακόλουθο κώδικα:

```
spark
```



(α)



(β)

Σχήμα 6-15: (α) Δημιουργία ενός νέου python σημειωματάριου, (β) Έλεγχος της διαθεσιμότητας του Spark μέσω του Jupyter Notebook (1η μέθοδος - καθορισμός μεταβλητών περιβάλλοντος PYSPARK_DRIVER_PYTHON και PYSPARK_DRIVER_OPTS)

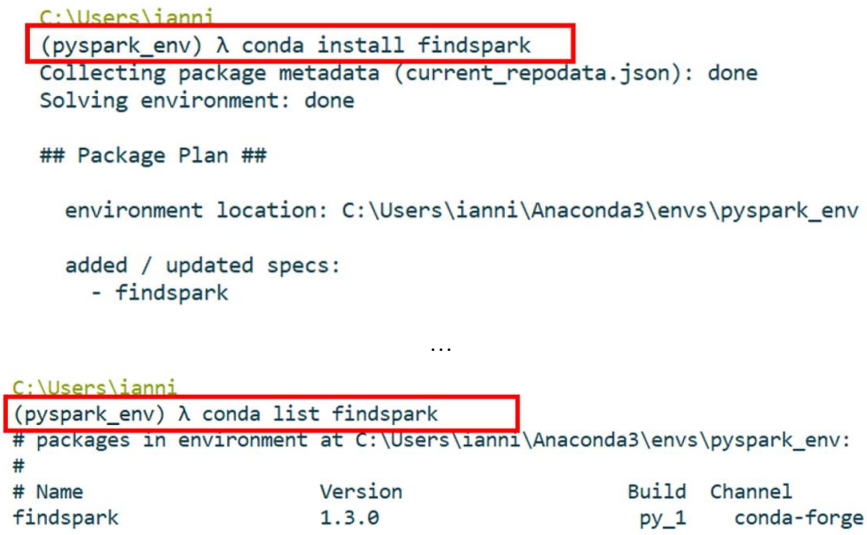
Μέθοδος 2 - Πακέτο FindSpark

Για να χρησιμοποιήσει ο χρήστης τη μέθοδο αυτή θα πρέπει να εγκαταστήσει το *jupyter* και ακολούθως το *findspark*, για να αποκτήσει πρόσβαση σε ένα στιγμιότυπο spark από το notebook jupyter (βλέπε Σχήμα 4.16):

```
conda activate pyspark_env
conda install jupyter
# Εγκατάσταση του findspark από το αποθετήριο conda-forge
conda install -c conda-forge findspark
conda list findspark
```

```
C:\Users\ianni
λ conda activate pyspark_env

C:\Users\ianni
(pyspark_env) λ conda install jupyter
```

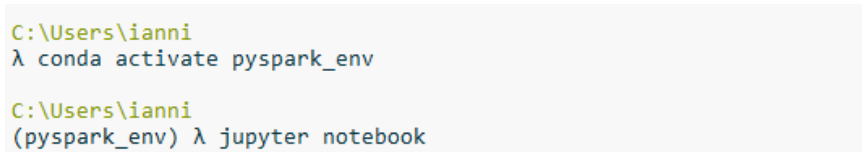
…



Σχήμα 6-16: Ενεργοποίηση του conda περιβάλλοντος pyspark_env - Εγκατάσταση των πακέτων jupyter και findspark – προβολή του πακέτου findspark

Για να μπορέσει ο χρήστης να χρησιμοποιήσει το jupyter μέσα από το περιβάλλον του θα πρέπει να κλείσει και να ανοίξει εκ νέου τη γραμμή εντολών και ακολούθως αφού ενεργοποιήσει το περιβάλλον conda, που δημιούργησε σε προγενέστερο βήμα, με την εντολή *conda activate pyspark_env* να ανοίξει το *jupyter* πληκτρολογώντας την εντολή *jupyter notebook* (βλέπε Σχήμα 6.17):

```
conda activate pyspark_env
jupyter notebook
```



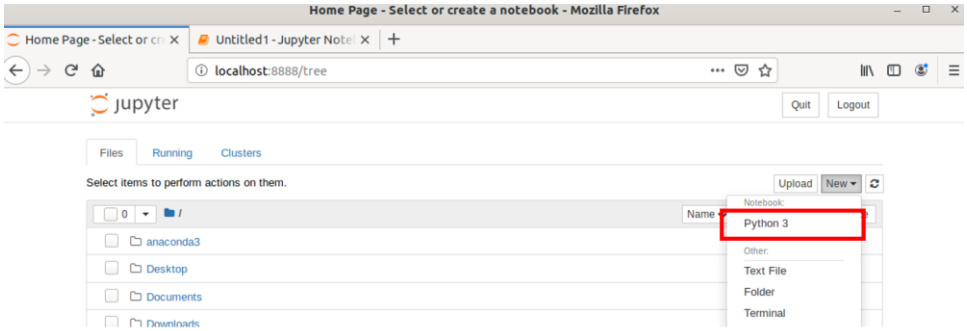
Σχήμα 6-17: Ενεργοποίηση του conda περιβάλλοντος pyspark_env – Εκκίνηση του jupyter notebook

Μέσα στη νέα καρτέλα με ονομασία Home Page που θα ανοίξει στον φυλλομετρητή θα πρέπει να επιλέξει την καρτέλα *files* (βλέπε Σχήμα 6.18α). Στο δεξί μέρος της υπάρχει η αναδυόμενη λίστα *New* στην οποία θα πρέπει να επιλέξει τον πυρήνα python 3 για να δημιουργήσει ένα νέο python σημειωματάριο. Στο νέο σημειωματάριο που θα δημιουργηθεί (βλέπε Σχήμα 6.18β) πληκτρολογεί τον ακόλουθο κώδικα:

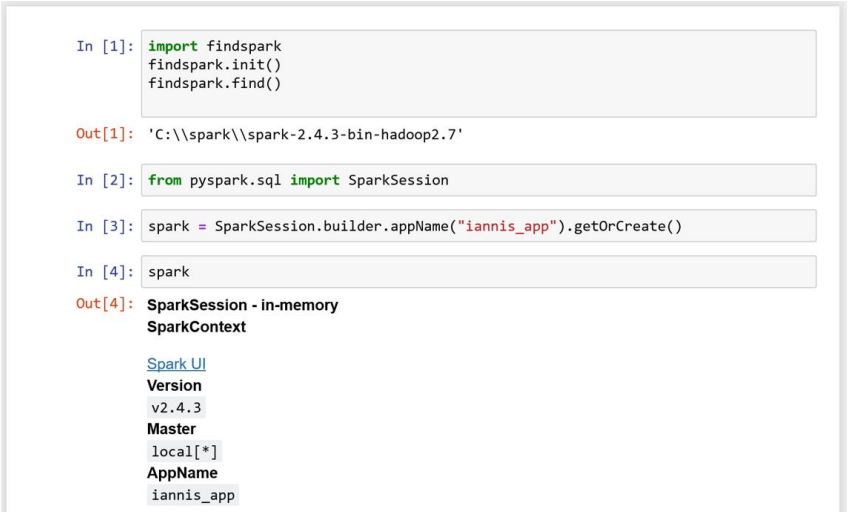
```
import findspark
findspark.init()
findspark.find()
```

και στη συνέχεια τις παρακάτω εντολές (βλέπε Σχήμα 6.18β):

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("iannis_app").getOrCreate()
spark
```



(α)



(β)

Σχήμα 6-18: (α) Δημιουργία ενός νέου python σημειωματάριου, (β) Έλεγχος της διαθεσιμότητας του Spark μέσω του Jupyter Notebook (2η μέθοδος - findspark)

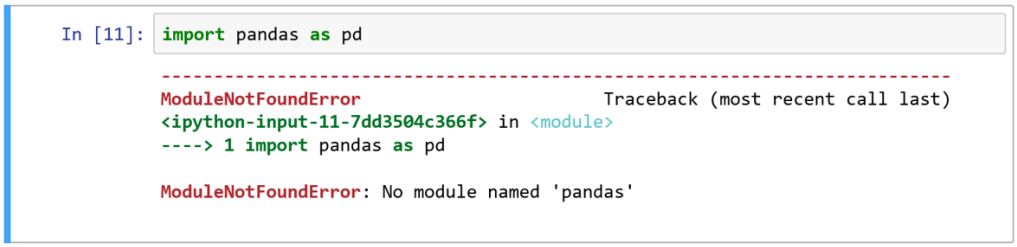
Σε αυτό το σημείο η εγκατάσταση και ο έλεγχος του pyspark μέσω του jupyter ολοκληρώθηκε με επιτυχία.

Εκκίνηση / Σταμάτημα του Jupyter Notebook

Η εκκίνηση / σταμάτημα του διακομιστή Jupyter Notebook γίνεται από τη γραμμή εντολών:

```
# Εκκίνηση του διακομιστή Jupyter Notebook:
#  * Για την μέθοδο 1, μέσω του pyspark
pyspark
# * Για την μέθοδο 2,  μέσω του πακέτου “findspark”
jupyter notebook

# Σταμάτημα του διακομιστή Jupter Notebook
Ctrl+c
```

Εάν παρατηρήσει ο χρήστης κάποιο λάθος στο σημειωματάριό (notebook) του που διαμαρτύρεται για το ότι δεν βρέθηκε το *pandas* (βλέπε Σχήμα 6.19), παρόλο που το είχε εγκαταστήσει ήδη στο περιβάλλον *conda*, θα πρέπει να βεβαιωθεί ότι έχει επιλέξει τον σωστό περιβάλλον.



Σχήμα 6-19: Σε περίπτωση μηνύματος λάθους κατά την εισαγωγή ενός πακέτου στο jupyter, ενώ είναι σίγουρος ο χρήστης ότι το έχει εγκαταστήσει, θα πρέπει να βεβαιωθεί ο χρήστης ότι έχει επιλέξει το σωστό περιβάλλον conda.

Βήμα 7ο (προαιρετικό): Εγκατάσταση της επέκτασης nb_conda *για πρόσβαση στο περιβάλλον* conda *και εγκατάσταση πακέτων μέσα από το Jupyter.*

Το *nb_conda* επιτρέπει στους χρήστες να έχουν πρόσβαση σε πυρήνες από άλλα περιβάλλοντα *conda* μέσα από το *Jupyter*. Για το λόγο αυτό θα πρέπει ο χρήστης να ενεργοποιήσει πρώτα το *conda* περιβάλλον του και ακολούθως να εγκαταστήσει το *nb_conda* (βλέπε Σχήμα 6.20):

```
conda activate pyspark_env
conda install nb_conda
```

```
C:\Users\ianni
(pyspark_env) λ conda install nb_conda
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\ianni\Anaconda3\envs\pyspark_env

  added / updated specs:
    - nb_conda
```

…

```
C:\Users\ianni
(pyspark_env) λ conda list nb_conda
# packages in environment at C:\Users\ianni\Anaconda3\envs\pyspark_env:
#
# Name                    Version                   Build  Channel
nb_conda                  2.2.1                     py37_2    conda-forge
nb_conda_kernels          2.2.2                     py37_0    conda-forge
```

Σχήμα 6-20: Εγκατάσταση του πακέτου nb_conda

Στη συνέχεια προκειμένου να εφαρμοστούν οι αλλαγές θα πρέπει να απενεργοποιήσει το περιβάλλον του και κατόπιν να το ενεργοποιήσει εκ νέου.

```
conda deactivate
conda activate pyspark_env
```

Θα πρέπει ακολούθως να εκκινήσει το *Jupyter Notebook* (βλέπε Σχήμα 6.21):

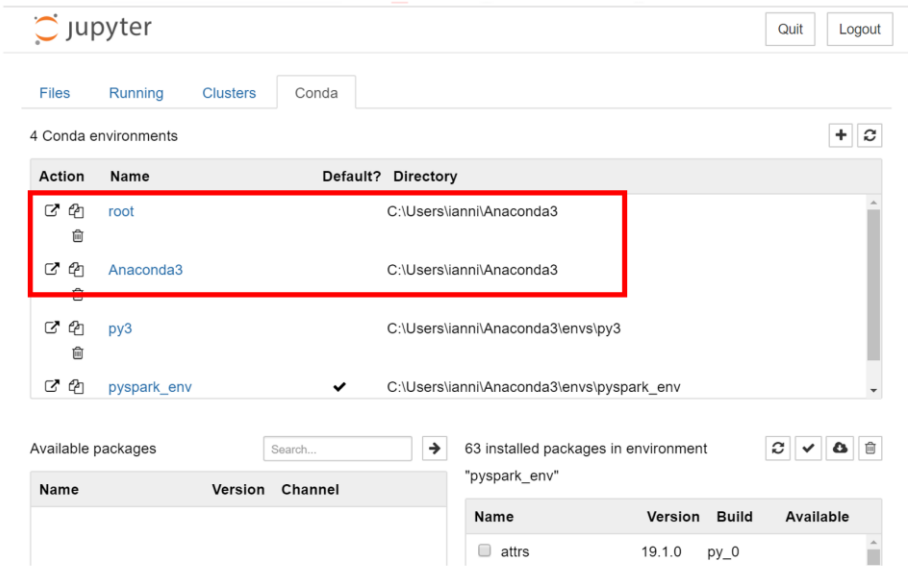
```
jupyter notebook
```

```
C:\Users\ianni
(pyspark_env) λ conda deactivate

C:\Users\ianni
λ conda activate pyspark_env

C:\Users\ianni
(pyspark_env) λ jupyter notebook
[I 03:36:23.236 NotebookApp] [nb_conda_kernels] enabled, 3 kernels found
[W 03:36:25.113 NotebookApp] Error loading server extension jupyter_nbextensions_configurator
    Traceback (most recent call last):
      File "C:\Users\ianni\Anaconda3\envs\pyspark_env\lib\site-packages\notebook\notebookapp.py", line 1615, in init_serv
er_extensions
        mod = importlib.import_module(modulename)
      File "C:\Users\ianni\Anaconda3\envs\pyspark_env\lib\importlib\__init__.py", line 127, in import_module
        return _bootstrap._gcd_import(name[level:], package, level)
      File "<frozen importlib._bootstrap>", line 1006, in _gcd_import
      File "<frozen importlib._bootstrap>", line 983, in _find_and_load
      File "<frozen importlib._bootstrap>", line 965, in _find_and_load_unlocked
    ModuleNotFoundError: No module named 'jupyter_nbextensions_configurator'
[I 03:36:25.142 NotebookApp] [nb_conda] enabled
[I 03:36:25.144 NotebookApp] Serving notebooks from local directory: C:\Users\ianni
[I 03:36:25.144 NotebookApp] The Jupyter Notebook is running at:
[I 03:36:25.144 NotebookApp] http://localhost:8888/?token=f7c7277f96ef6605f22d67aa5f59d28681b3cf45162d0835
[I 03:36:25.145 NotebookApp]  or http://127.0.0.1:8888/?token=f7c7277f96ef6605f22d67aa5f59d28681b3cf45162d0835
[I 03:36:25.145 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 03:36:26.894 NotebookApp]
```

Σχήμα 6-21: Μετά την εγκατάσταση του πακέτου nb_conda σε ένα περιβάλλον conda, θα πρέπει ο χρήστης να απενεργοποιήσει το περιβάλλον και κατόπιν να το ενεργοποιήσει εκ νέου για να είναι διαθέσιμο.

Στο παραπάνω σχήμα παρατηρεί κανείς ότι κατά την εκκίνησή του το *Jupyter Notebook* εντόπισε τρείς *nb_conda* πυρήνες. Επίσης όπως φαίνεται στο Σχήμα 6.22 στο περιβάλλον του *jupyter* δημιουργήθηκε μια επιπλέον καρτέλα με ονομασία *conda* όπου φαίνονται ποια είναι τα εικονικά περιβάλλοντα *conda* που έχει δημιουργήσει ο χρήστης, ποιο περιβάλλον είναι ενεργό και επιτρέπει στον χρήστη να αλλάζει κατά βούληση το ενεργό περιβάλλον. Υπάρχει όμως ένα μικρό πρόβλημα που πρέπει να διορθωθεί (το πρόβλημα δεν είναι μόνο οπτικό αλλά και λειτουργικό). Η καρτέλα αυτή εμφανίζει 4 περιβάλλοντα *conda*, απ' τα οποία τα δυο (root, Anaconda3) παραπέμπουν στην ίδια διαδρομή (*C:\Users\Anaconda3*). Το πρόβλημα αυτό οφείλεται σε αλλαγή της εξόδου *conda info --json* στις νέες εκδόσεις του *conda* (error#66) και απαιτεί μια μικρή παρέμβαση από μέρους του χρήστη προκειμένου να διορθωθεί το πρόβλημα (αν στη εγκατάσταση του χρήστη δεν εμφανίζεται το πρόβλημα αυτό, σημαίνει ότι θα έχει διορθωθεί με κάποια μεταγενέστερη ενημέρωση, οπότε μπορεί να συνεχίσει με τα βήματα μετά το Σχήμα 6.24). Ο ποιο εύκολος τρόπος είναι να ανοίξει τη γραμμή εντολών και να αναζητήσει που βρίσκεται το αρχείο *envmanager.py* (βλέπε Σχήμα 6.23) πληκτρολογώντας την παρακάτω εντολή (πιθανόν να αργήσει λίγο η εντολή αλλά τελικά θα βρει τα αρχεία). Βλέπει κανείς ότι στον υπολογιστή της εγκατάστασης υπάρχουν τρία αρχεία, ο χρήστης θα πρέπει να ενδιαφερθεί για αυτό που βρίσκεται στον υποκατάλογο *nbconda* στη διαδρομή *...envs\pyspark_env\site-packages\nb_conda* μέσα στο περιβάλλον του (στην παρούσα εγκατάσταση το *pyspark_env*).

```
where /r c:\ envmanager.py
```

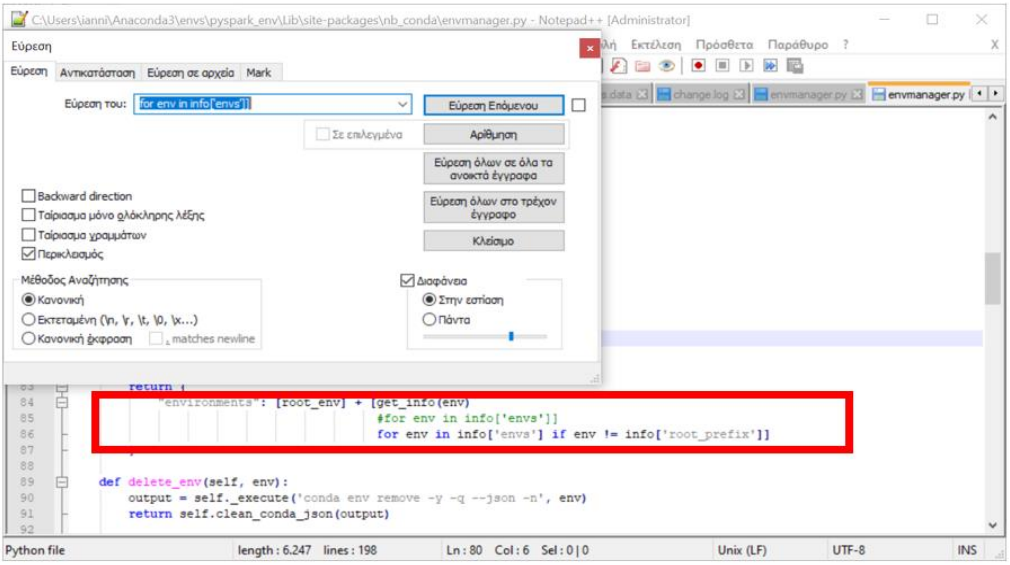


Σχήμα 6-22: Στην καρτέλα conda φαίνονται ποια είναι τα εγκατεστημένα εικονικά περιβάλλοντα

```
C:\Users\ianni
(base) λ where /r c:\ envmanager.py
c:\Users\ianni\Anaconda3\envs\py3\Lib\site-packages\nb_conda\envmanager.py
c:\Users\ianni\Anaconda3\envs\pyspark_env\Lib\site-packages\nb_conda\envmanager.py
c:\Users\ianni\Anaconda3\Lib\site-packages\nb_conda\envmanager.py
c:\Users\ianni\Anaconda3\pkgs\nb_conda-2.2.1-py37_2\Lib\site-packages\nb_conda\envmanager.py
c:\Users\ianni\Anaconda31\pkgs\nb_conda-2.2.1-py37_2\Lib\site-packages\nb_conda\envmanager.py
```

Σχήμα 6-23: Αναζήτηση του προγράμματος envmanager.py με την εντολή where

Στη συνέχεια αφού το ανοίξει με κάποιον επεξεργαστή κώδικα όπως τον *Notepad++*, θα πρέπει να πατήσει *CTR+F* και να εισάγει στο παράθυρο αναζήτησης : *for env in info['envs']]* (βλέπε Σχήμα 6.24). Θα βρει μια αναφορά την οποία θα πρέπει να αντικαταστήσει με την: *for env in info['envs'] if env != info['root_prefix']]*.



Σχήμα 6-24: Εύρεση της έκφρασης 'for env in info['envs']]' και αντικατάστασή της με την 'for env in info['envs'] if env != info['root_prefix']]'

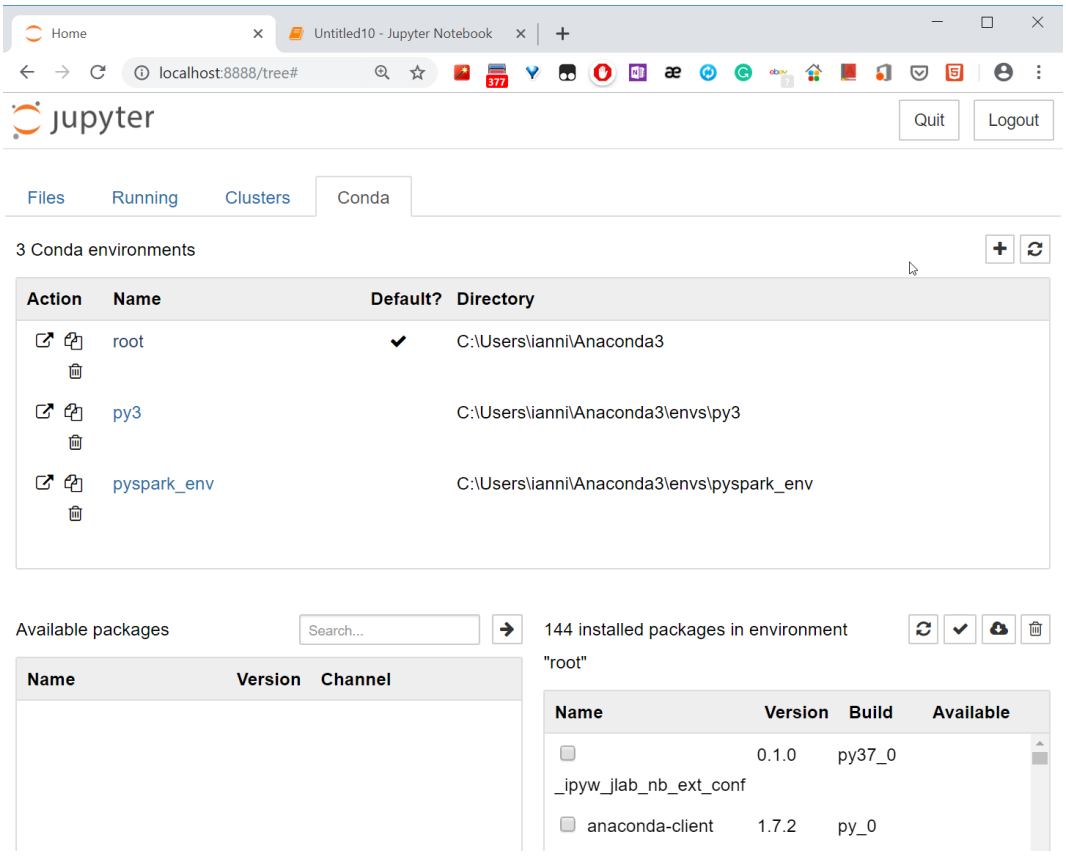
Αφού σώσει το αρχείο, θα πρέπει να κλείσει τη γραμμή εντολών εάν ήταν ήδη ανοικτή, στη συνέχεια να ανοίξει μια νέα και να εκκινήσει εκ νέου το *jupyter*:

```
jupyter notebook
```

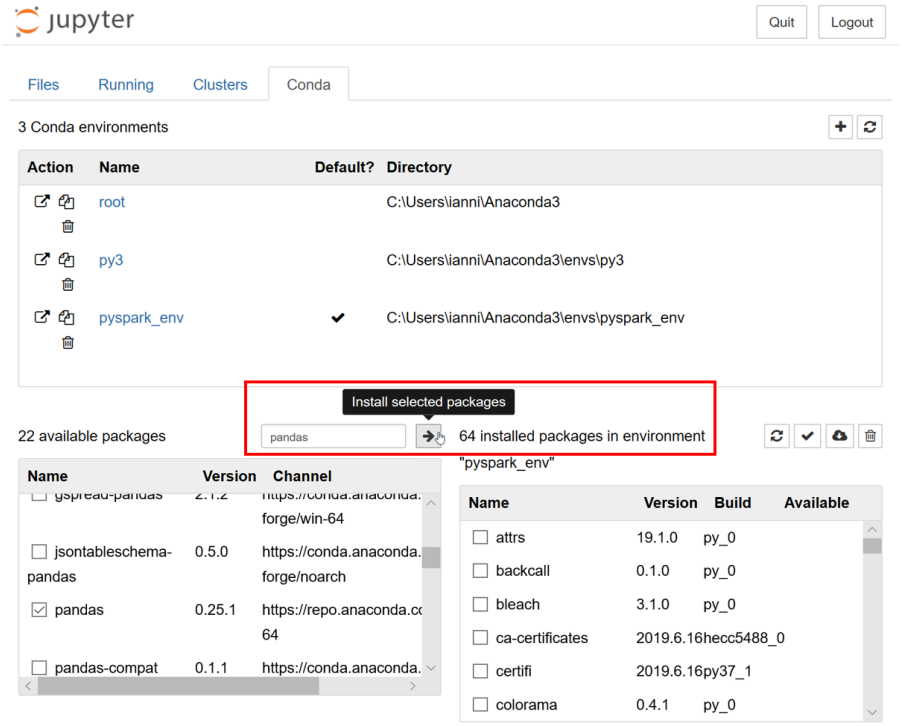
Όπως φαίνεται στο Σχήμα 6.25, το πρόβλημα αποκαταστάθηκε. Από την καρτέλα conda μπορεί να εγκαθιστά και να απεγκαθιστά πακέτα σε ένα περιβάλλον *conda*. Μπορεί επίσης να διαχειρίζεται τα διάφορα περιβάλλοντα (προσθήκη / διαγραφή / κλωνοποίηση).

Για να προσθέσει ο χρήστης στο περιβάλλον *conda* με ονομασία *pyspark_env* το πακέτο που ήθελε (*pandas*) θα πρέπει να ακολουθήσει τα παρακάτω βήματα:

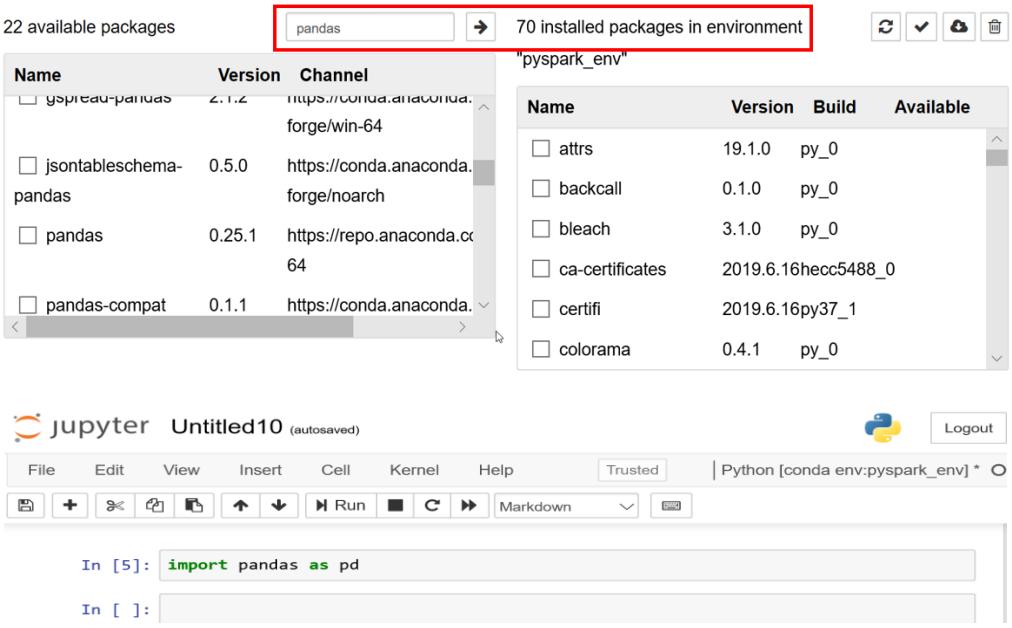
- Θα πρέπει πρώτα να επιλέξει το εν λόγω περιβάλλον conda
- Στη συνέχεια να κάνει ανανέωση των πακέτων με το πλήκτρο
- Ακολούθως θα πρέπει να επιλέξει το πακέτο ή τα πακέτα που θέλει να εγκαταστήσει και να πατήσει το πλήκτρο (βλέπε Σχήμα 6.26).



Σχήμα 6-25: Το πρόβλημα (βλέπε Σχήμα 6.22) όπου στην καρτέλα conda δυο περιβάλλοντα τα root και Anaconda3 παρέπεμπαν στην ίδια διαδρομή αποκαταστάθηκε



Σχήμα 6-26: Διαδικασία εγκατάστασης του πακέτου pandas μέσα από την καρτέλα conda

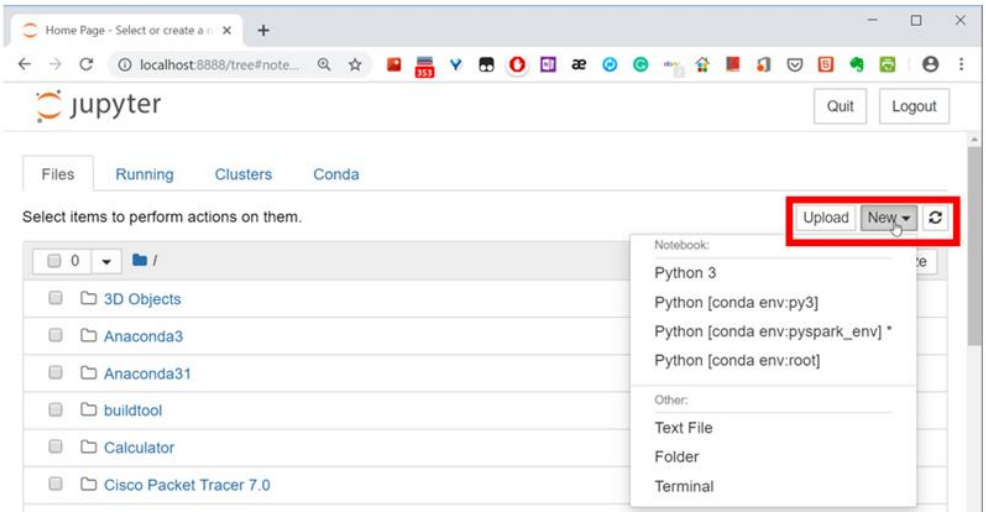


Σχήμα 6-27: Το πακέτο pandas εγκαταστάθηκε με επιτυχία – Επιτυχής εισαγωγή του πακέτου pandas στο jupyter

Όπως παρατηρεί κανείς στο παραπάνω σχήμα η εντολή εκτελέστηκε κανονικά (ο αριθμός μέσα της αγκύλες In [5] αριστερά της εντολής υποδηλώνει την ορθή εκτέλεσή της) μετά την εγκατάσταση του πακέτου.

Από την καρτέλα *files*, μέσω της αναδυόμενης λίστα *New* μπορεί να επιλέξει ο χρήστης τον πυρήνα που επιθυμεί (όσες επιλογές έχουν την ένδειξη *conda env:env_name*, είναι

συνδυασμός πυρήνα-περιβάλλοντος conda). Οι λειτουργίες είναι διαισθητικές και μπορεί να της ανακαλύψει κανείς πειραματιζόμενος με το περιβάλλον του *jupyter*.



Σχήμα 6-28: Από την καρτέλα files, μέσω της αναδυόμενης λίστα New μπορεί να επιλέξει ο χρήστης τον πυρήνα που επιθυμεί

Εγκατάσταση του PySpark στο Ubuntu 19.04 σε τοπική λειτουργία (local mode)

Για περισσότερες πληροφορίες σχετικά με την τοπική λειτουργία του Spark θα πρέπει να ανατρέξει ο χρήστης στην ενότητα "Τρέξιμο της εφαρμογής Spark τοπικά (local mode)". Τα περισσότερα ζητήματα στην εγκατάσταση του Pyspark προκαλούνται από μη κατάλληλα καθορισμένες μεταβλητές περιβάλλοντος, για το λόγω αυτό θα πρέπει να είναι κανείς ιδιαίτερα προσεκτικός κατά τον καθορισμό τους.

Βήμα 1ο : Εγκατάσταση της Java 8

Μπορεί ο χρήστης να εγκαταστήσει είτε την έκδοση *Oracle JDK* είτε αυτή του *OpenJDK*. Η προσωπική άποψη του γράφοντα, έχοντας δοκιμάσει και τις δυο εκδόσεις, είναι ότι λειτουργούν εξίσου καλά.

Έλεγχος της εγκατεστημένης έκδοσης

Είναι απόλυτα εξακριβωμένο ότι η έκδοση 8 (openjdk8 ή java 1.8) θα απαλλάξει τον χρήστη από πολλά προβλήματα που δημιουργούν οι μεταγενέστερες εκδόσεις, γι' αυτό θα πρέπει να βεβαιωθεί ότι είναι εγκατεστημένη στο σύστημά του, πληκτρολογώντας στη γραμμή εντολών:

```
java -version
```

Θα πρέπει να δει ένα μήνυμα παρόμοιο με το παρακάτω:

```
openjdk version "1.8.0_222"
OpenJDK Runtime Environment (build 1.8.0_222-8u222-b10-1ubuntu1~19.04.1-b10)
OpenJDK 64-Bit Server VM (build 25.222-b10, mixed mode)
```

ή

```
java version "1.8.0_221"
Java(TM) SE Runtime Environment (build 1.8.0_221-b11)
```

```
Java HotSpot(TM) 64-Bit Server VM (build 25.221-b11, mixed mode)
```

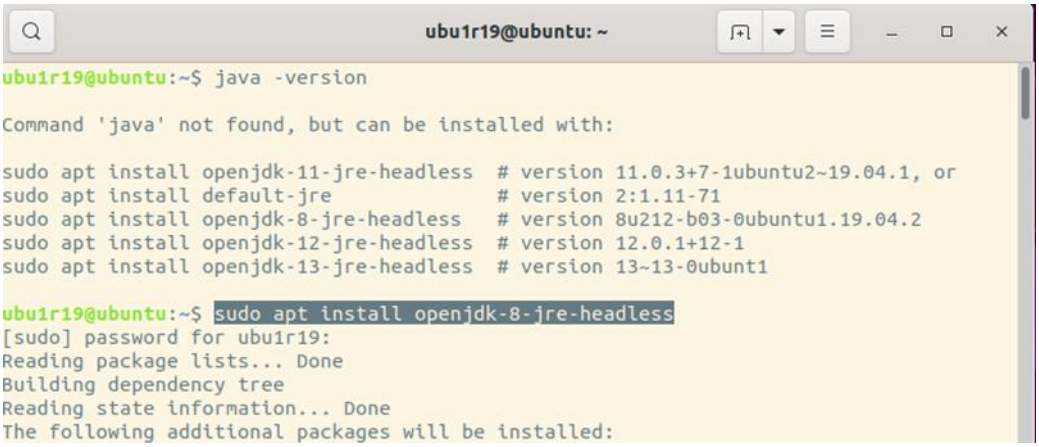
Αν δεν είναι εγκατεστημένη θα δει ένα μήνυμα παρόμοιο με το ακόλουθο:

```
Command 'java' not found  …
```

Αν έχει ήδη εγκατεστημένη την έκδοση αυτή, θα πρέπει να ελέγξει τις μεταβλητές περιβάλλοντος όπως περιγράφονται στο τέλος του επόμενου βήματος, σε διαφορετική περίπτωση θα πρέπει να μεταβεί στο επόμενο βήμα.

Openjdk version

Το Ubuntu όπως είναι φυσικό προτείνει στον χρήστη να εγκαταστήσει το openjdk καθότι και τα δυο είναι λογισμικά ανοικτού κώδικα (βλέπε Σχήμα 6.29).



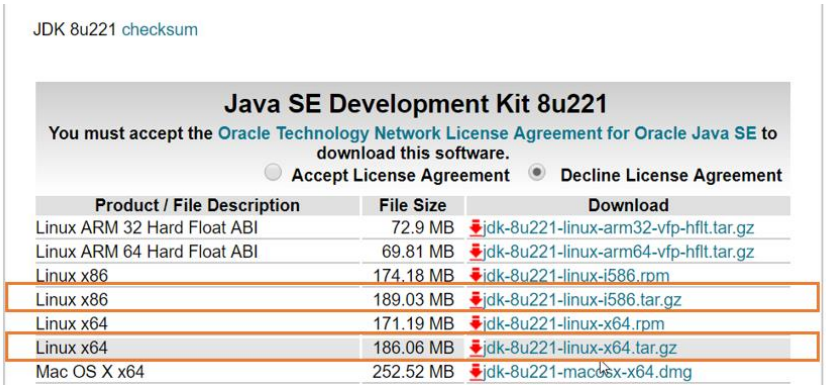
Σχήμα 6-29:Εγκατάσταση του openjdk-8

Θα πρέπει να ανοίξει το τερματικό πατώντας Alt+Ctrl+T και να τρέξει στη γραμμή εντολών την ακόλουθη εντολή:

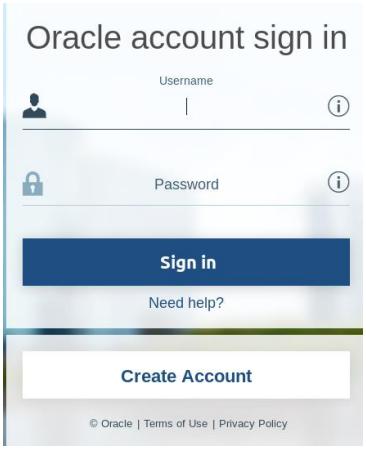
```
sudo apt install openjdk-8-jre-headless
```

Oracle JDK version

Για το Oracle JDK θα πρέπει να μεταβεί ο χρήστης στην ιστοσελίδα λήψεων του επίσημου ιστότοπου της Oracle, και αφού αποδεχτεί την άδεια χρήσης της Oracle να κατεβάσει την έκδοση του Java JDK 8, που είναι κατάλληλη για το σύστημά του (θα χρειαστεί να κάνει πρώτα εγγραφή, σύμφωνα με τους νέους όρους της Oracle, για να του επιτραπεί η μεταφόρτωση – βλέπε Σχήματα 6.30, 6.31).



Σχήμα 6-30: Ο χρήστης πρέπει να αποδεχθεί πρώτα την άδεια χρήσης πριν την μεταφόρτωση του αρχείου που τον ενδιαφέρει (Windows x86 / Windows x64)



Σχήμα 6-31: θα χρειαστεί ο χρήστης να δημιουργήσει πρώτα ένα λογαριασμό στην Oracle πριν του επιτραπεί η μεταφόρτωση του αρχείου

Στη συνέχεια θα πρέπει να εξάγει το JDK στη διαδρομή */usr/lib/jvm*. Εάν ο φάκελος */usr/lib/jvm* δεν υπάρχει, αυτή η εντολή θα δημιουργήσει τον κατάλογο. Αν έχει ήδη ο χρήστης αυτόν τον φάκελο, μπορεί να αγνοήσει αυτό το βήμα και να προχωρήσει στο επόμενο βήμα. Για παράδειγμα:

```
sudo mkdir /usr/lib/jvm
sudo tar xvfz ~/Downloads/jdk-8u221-linux-x64.tar.gz -C /usr/lib/jvm
sudo update-alternatives --install /usr/bin/java java
/usr/lib/jvm/jdk1.8.0_221/bin/java 1
```

Διαχείριση της Java

Μπορεί ο χρήστης να έχει πολλές εγκατεστημένες εκδόσεις της Java σε έναν διακομιστή. Εδώ για παράδειγμα, ακολουθώντας τα παραπάνω βήματα ο γράφων εγκατέστησε δύο διαφορετικές εκδόσεις της Java (openjdk-8, Oracle jdk-8). Του δίνεται η δυνατότητα να ρυθμίσει ποια έκδοση θα είναι η προεπιλεγμένη για χρήση, χρησιμοποιώντας την εντολή *update-alternatives --config java* (βλέπε Σχήμα 6.32).

```
sudo update-alternatives --config java
```

```
ubu1r19@ubuntu:~$ sudo update-alternatives --config java
[sudo] password for ubu1r19:
There are 2 choices for the alternative java (providing /usr/bin/java).

  Selection    Path                                            Προτε Status
------------------------------------------------------------
* 0            /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java   1081      auto mode
  1            /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java   1081      manual mode
  2            /usr/lib/jvm/jdk1.8.0_221/bin/java               1         manual mode

Press <enter> to keep the current choice[*], or type selection number: 2
update-alternatives: using /usr/lib/jvm/jdk1.8.0_221/bin/java to provide /usr/bin/java (java) in manual mode
ubu1r19@ubuntu:~$ sudo update-alternatives --config java
There are 2 choices for the alternative java (providing /usr/bin/java).

  Selection    Path                                            Προτε Status
------------------------------------------------------------
  0            /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java   1081      auto mode
  1            /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java   1081      manual mode
* 2            /usr/lib/jvm/jdk1.8.0_221/bin/java               1         manual mode

Press <enter> to keep the current choice[*], or type selection number: 0
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java to provide /usr/bin/java (java) in auto mo
ubu1r19@ubuntu:~$
```

Σχήμα 6-32: Επιλογή της επιθυμητής έκδοσης java μέσω της εντολής update-alternatives --config java

Παρατηρώντας το παραπάνω σχήμα διαπιστώνει κανείς ότι την πρώτη φορά που εκτελέστηκε η εντολή η προεπιλεγμένη έκδοση της java (αυτή με το *) ήταν αυτή του openjdk και ο γράφων επέλεξε να ορίσει την java της Oracle, πληκτρολογώντας 2 και πατώντας enter. Τη δεύτερη φορά όρισε εκ νέου ως προεπιλεγμένη έκδοση την java openjdk, πληκτρολογώντας 0 και πατώντας enter.

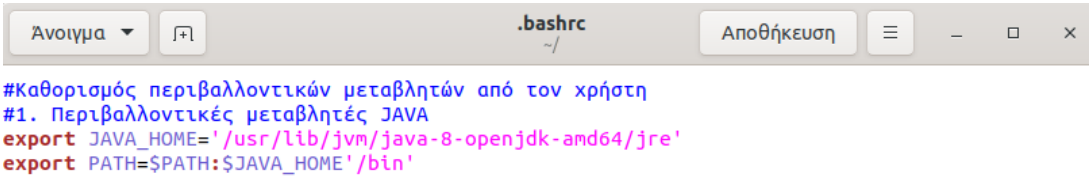
Καθορισμός των μεταβλητών περιβάλλοντος:

Επίσης, συνιστάται να ορίσει ο χρήστης τη μεταβλητή περιβάλλοντος JAVA_HOME και να προσθέσει τα JAVA binaries αρχεία στη μεταβλητή PATH. Μπορεί να κάνει τις αλλαγές αυτές είτε στο αρχείο *.bashrc* είτε στο */etc/environment*. Παρακάτω ο χρήστης θα πρέπει να επιλέξει να ανοίξει το αρχείο *.bashrc* με το gedit, τον επίσημο Επεξεργαστή Κειμένου του Ubuntu:

```
gedit ~/.bashrc
```

Και κατόπιν να προσθέσει τις ακόλουθες γραμμές στο αρχείο:

```
export JAVA_HOME='/usr/lib/jvm/java-8-openjdk-amd64/jre'
export PATH=$PATH:$JAVA_HOME'/bin'
```



Σχήμα 6-33: Καθορισμός των απαραίτητων μεταβλητών περιβάλλοντος για την Java στο αρχείο .bashrc

Για να εφαρμοστούν οι αλλαγές, θα πρέπει να πληκτρολογήσει:

```
source ~/.bashrc
```

Ακολούθως θα πρέπει να ελέγξει την έκδοση της java:

```
java -version
```

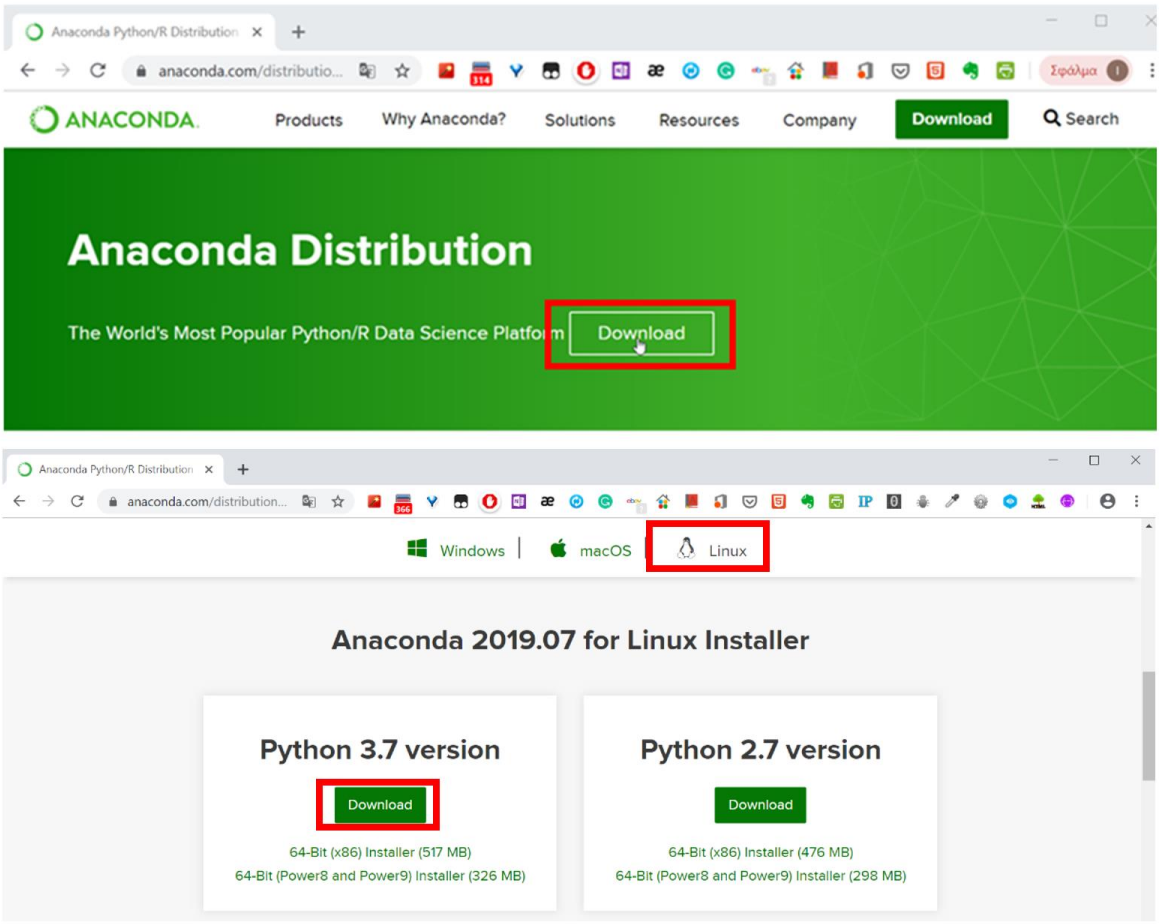
Βήμα 2ο : Λήψη και Εγκατάσταση του Anaconda (για python)

Ανεξάρτητα από το αν ο χρήστης έχει εγκατεστημένη ή όχι την επίσημη διανομή CPython που διατίθεται από το python.org (έλεγχος με εντολές *whereis python*, *python --version*), θα ήταν προτιμότερο να εγκαταστήσει και το Anaconda, μια ελεύθερη και ανοικτού κώδικα διανομή των γλωσσών προγραμματισμού Python και R με πολλά πλεονεκτήματα που αποσκοπεί στην απλοποίηση της διαχείρισης και της ανάπτυξης πακέτων. Η διανομή Anaconda περιλαμβάνει περισσότερα από 1.500 πακέτα, καθώς και τον διαχειριστή πακέτων και εικονικών περιβαλλόντων conda, που χειρίζεται την εγκατάσταση πακέτων Python αλλά και εξωτερικών απαιτήσεων λογισμικού τρίτων κατασκευαστών. Περιλαμβάνει επίσης ένα γραφικό περιβάλλον, το Anaconda Navigator, ως γραφική εναλλακτική λύση στη διεπαφή γραμμής εντολών (CLI) [28]. Τα εικονικά περιβάλλοντα επιτρέπουν στον χρήστη να εγκαταστήσει συγκεκριμένες εκδόσεις πακέτων για ένα συγκεκριμένο έργο χωρίς να ανησυχεί για συγκρούσεις εκδόσεων.

Με τη λήψη του Anaconda, ο χρήστης αποκτά πρόσβαση στα **conda**, **Python**, **Jupyter Notebook** και εκατοντάδες άλλα πακέτα ανοιχτού κώδικα.

Συνιστάται στον χρήστη η μεταφόρτωση και εκτύπωση του Anaconda Cheatsheet.

Για τη λήψη του Anaconda θα πρέπει ο χρήστης να μεταβεί στην ιστοσελίδα λήψεων του επίσημου ιστότοπου του Anaconda και να κατεβάσει την πιο πρόσφατη έκδοση (τη στιγμή που γράφονταν η διπλωματική η πιο πρόσφατη έκδοση ήταν η 3.7).

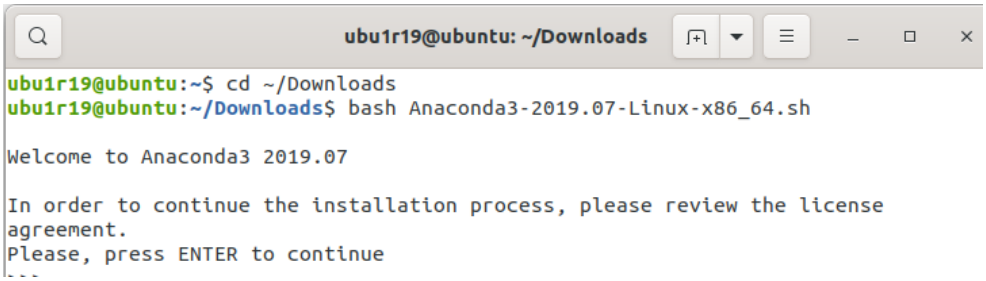


Σχήμα 6-34: Βήματα για την μεταφόρτωση του αρχείου Anaconda 2019.07 (Python 3.7 version) για Linux OS

Στη συνέχεια αφού ανοίξει το τερματικό και μεταβεί στον φάκελο Downloads, που βρίσκεται στη διαδρομή *~/Downloads*, θα πρέπει να εκτελέσει το script εγκατάστασης του Anaconda:

```
bash Anaconda3-2019.07-Linux-x86_64.sh
```

Θα λάβει την ακόλουθη έξοδο, που τον προτρέπει να ελέγξει τη σύμβαση παραχώρησης άδειας χρήσης (licence agreement). Θα πρέπει να πατήσει το πλήκτρο *enter* για να συνεχίσει.



Σχήμα 6-35: Για να συνεχιστεί η εγκατάσταση του Anaconda η εφαρμογή εγκατάστασης προτρέπει τον χρήστη να πατήσει Enter και να ελέγξει τη σύμβαση παραχώρησης άδειας χρήσης (licence agreement)

Όταν φτάσει στο τέλος της σύμβασης παραχώρησης άδειας χρήσης, θα πρέπει να πληκτρολογήσει *yes*, εφόσον συμφωνεί με αυτήν και ακολούθως να επιβεβαιώσει την

προεπιλεγμένη διαδρομή εγκατάστασης (*~/anaconda3*) πατώντας *enter* για να ολοκληρωθεί η εγκατάσταση.

Μόλις ολοκληρωθεί η εγκατάσταση, θα λάβει την ακόλουθη έξοδο:

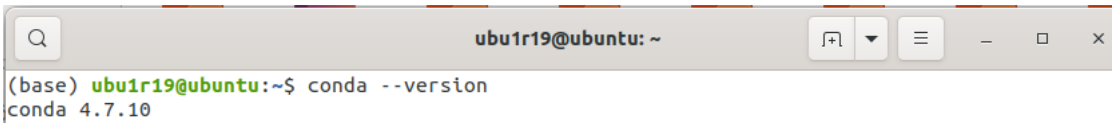
```
installation finished.
Do you wish the installer to initialize Anaconda3
by running conda init? [yes|no]
[no] >>> yes
```

Σχήμα 6-36: Για να αρχικοποιηθεί το Anaconda3 (επιτρέπει τη χρήση του conda μέσα από το τερματικό) θα πρέπει ο χρήστης να πληκτρολογήσει 'yes' στο εικονιζόμενο μήνυμα.

Συνιστάται να πληκτρολογήσει *yes* για να μπορεί να χρησιμοποιήσει την εντολή *conda* μέσα από το τερματικό.

Στη συνέχεια αφού κλείσει το τερματικό και ανοίξει ένα νέο θα πρέπει να εκτελέσει την εντολή *conda --version*. Παρατηρώντας κανείς το παρακάτω σχήμα θα διαπιστώσει ότι η γραμμή εντολών του χρήστη έχει πρόθεμα με το όνομα του ενεργού περιβάλλοντος conda (base – το βασικό περιβάλλον conda). Για περισσότερες πληροφορίες σχετικά με τα περιβάλλοντα conda θα πρέπει να ανατρέξει ο χρήστης στην ενότητα "Διαχείριση έργων".

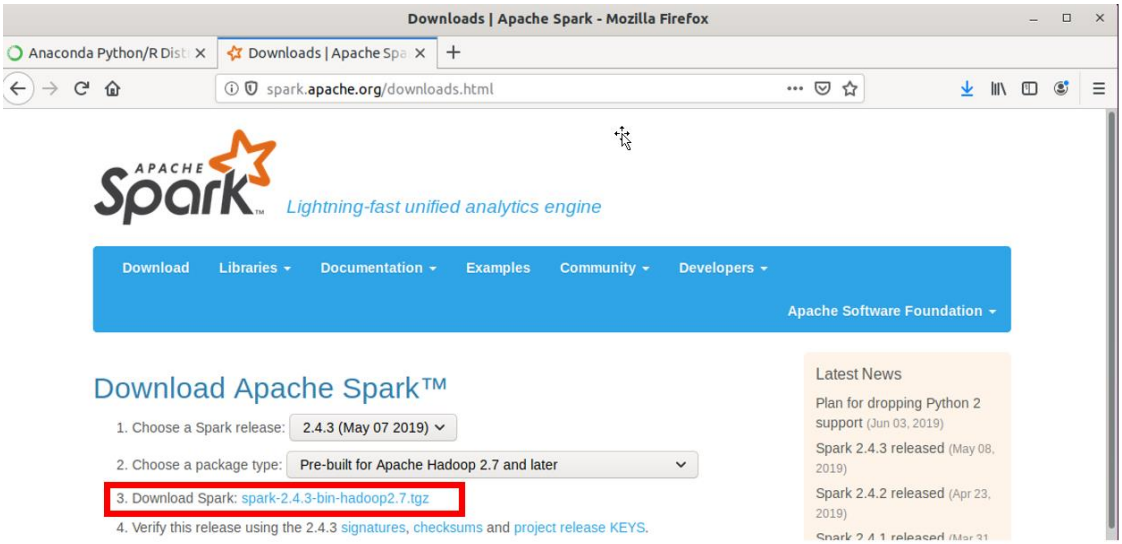
```
conda --version
```



Σχήμα 6-37: Μετά την αρχικοποίηση του Anaconda3 θα διαπιστώσει κανείς ότι η γραμμή εντολών του χρήστη έχει πρόθεμα με το όνομα του ενεργού περιβάλλοντος conda (base – το βασικό περιβάλλον conda). Ο έλεγχος της εγκατάστασης του Anaconda3 γίνεται με την εντολή conda --version

Βήμα 3ο : Λήψη και εγκατάσταση του Spark

Πηγαίνοντας στην ιστοσελίδα λήψεων του επίσημου ιστότοπου Spark μπορεί ο χρήστης να κατεβάσει την πιο πρόσφατη έκδοση (τη στιγμή που γράφονταν η διπλωματική, ήταν η 2.4.3) του αρχείου με επέκταση .tgz (βλέπε Σχήμα 6.38).



Σχήμα 6-38: Επιλογή της έκδοσης Spark (prebuilt for Apache Hadoop 2.7 and later) που επιθυμεί ο χρήστης

Εναλλακτικά μπορεί να αντιγράψει τον σύνδεσμο του αρχείου και να το κατεβάσει από την γραμμή εντολών με την *curl* ή την *wget* στη διαδρομή ~/Downloads :

```
cd ~/Downloads
curl      -O      https://archive.apache.org/dist/spark/spark-2.4.3/spark-2.4.3-bin-hadoop2.7.tgz
```

Αν δεν είναι εγκατεστημένη η εντολή *curl* την εγκαθιστά με την εντολή:

```
sudo apt install curl
```

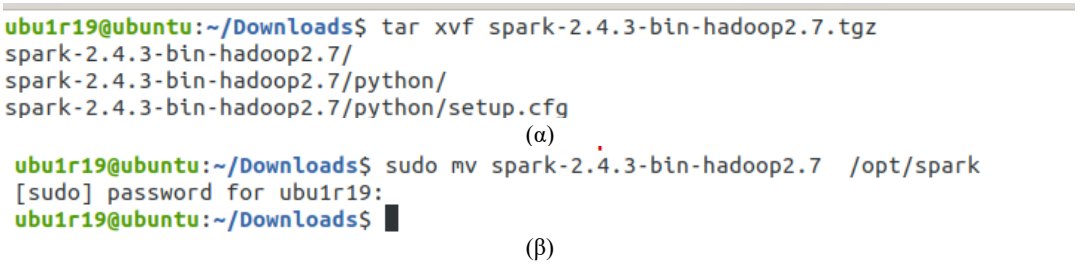
Ένα αρχείο με επέκταση .tgz ή .tar.gz είναι ένα TAR Archive αρχείο που έχει συμπιεστεί χρησιμοποιώντας το λογισμικό Gnu Zip (gzip). Με την εντολή *tar* μπορεί ο χρήστης να το αποσυμπιέσει (.tgz -> .tar) και να εξάγει τη συλλογή αρχείων ή καταλόγων από το TAR στον κατάλογο που έχει επιλέξει. Με τις παρακάτω εντολές μπορεί ο χρήστης να εξαγάγει το αρχείο *spark-2.4.3-bin-hadoop2.7.tgz* στον ομώνυμο κατάλογο *spark-2.4.3-bin-hadoop2.7* που βρίσκεται στη διαδρομή *~/Downloads*. Στη συνέχεια μπορεί, αν το επιθυμεί, να διαγράψει το αρχείο με επέκταση .tgz (εντολή *rm*):

```
tar xvf spark-2.4.3-bin-hadoop2.7.tgz
rm -rf spark-2.4.3-bin-hadoop2.7.tgz
ls
```

Στη συνέχεια θα πρέπει να μετακινήσει τον κατάλογο *spark-2.4.3-bin-hadoop2.7* μέσα στον κατάλογο */opt/* μετονομάζοντας τον ταυτόχρονα σε *spark:*

```
sudo mv spark-2.4.3-bin-hadoop2.7 /opt/spark
```

Τα παραπάνω βήματα φαίνονται στο παρακάτω σχήμα.



Σχήμα 6-39: (α) Αποσυμπίεση και εξαγωγή του αρχείου spark-2.4.3-bin-hadoop2.7.tgz στον ομώνυμο φάκελο spark-2.4.3-bin-hadoop2.7 μέσα , στον τρέχοντα κατάλογο, (β) Μετακίνηση του καταλόγου spark-2.4.3-bin-hadoop2.7 μέσα στον κατάλογο /opt/, μετονομάζοντας τον ταυτόχρονα σε spark

Αφού ανοίξει ο χρήστης το αρχείο *.bashrc* με το *gedit*:

```
gedit ~/.bashrc
```

θα πρέπει να καθορίσει τη μεταβλητή περιβάλλοντος SPARK_HOME και να προσθέσει τα SPARK binaries αρχεία στη μεταβλητή PATH προσθέτοντας στο αρχείο τις ακόλουθες γραμμές (βλέπε Σχήμα 4.40):

```
export SPARK_HOME= '/opt/spark'
export PATH=$PATH:$SPARK_HOME/bin
```

Σχήμα 6-40: Καθορισμός των απαραίτητων μεταβλητών περιβάλλοντος για το Spark στο αρχείο .bashrc

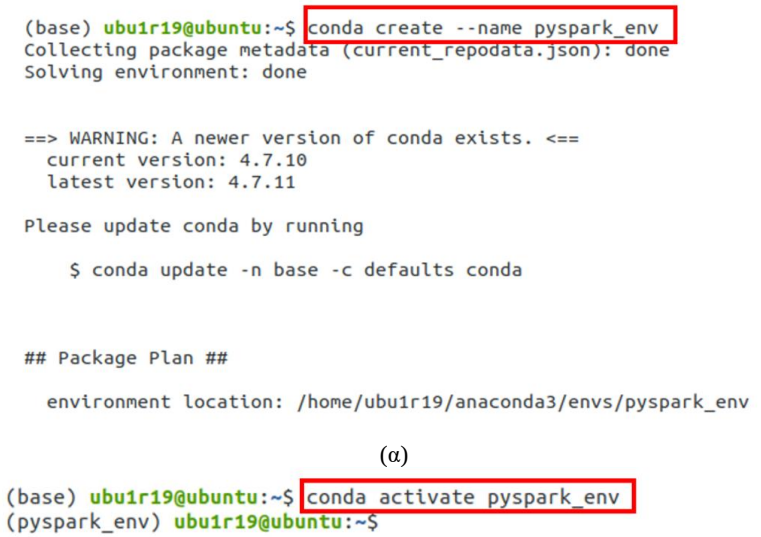
Τέλος για να εφαρμοστούν οι αλλαγές θα πρέπει να τρέξει την παρακάτω εντολή:

```
source ~/.bashrc
```

Βήμα 4ο: Έλεγχος της εγκατάστασης του PySpark

Θα πρέπει ο χρήστης να δημιουργήσει πρώτα ένα περιβάλλον conda για το έργο του, από το οποίο θα τρέχει το Jupyter Notebook (στην περίπτωση του γράφοντα το *pyspark_env*). Στην ενότητα "Διαχείριση έργων" δίδονται αναλυτικές πληροφορίες σχετικά με τα περιβάλλοντα *conda*. Για το λόγο αυτό θα πρέπει να ανοίξει τη γραμμή εντολών και να πληκτρολογήσει (βλέπε Σχήμα 6.41):

```
#Δημιουργία περιβάλλοντος conda με ονομασία pyspark_env και εγκατάσταση της python 3.7.3
conda create --name pyspark_env python=3.7.3

#Ενεργοποίηση του ανωτέρω περιβάλλοντος conda
conda activate pyspark_env
```



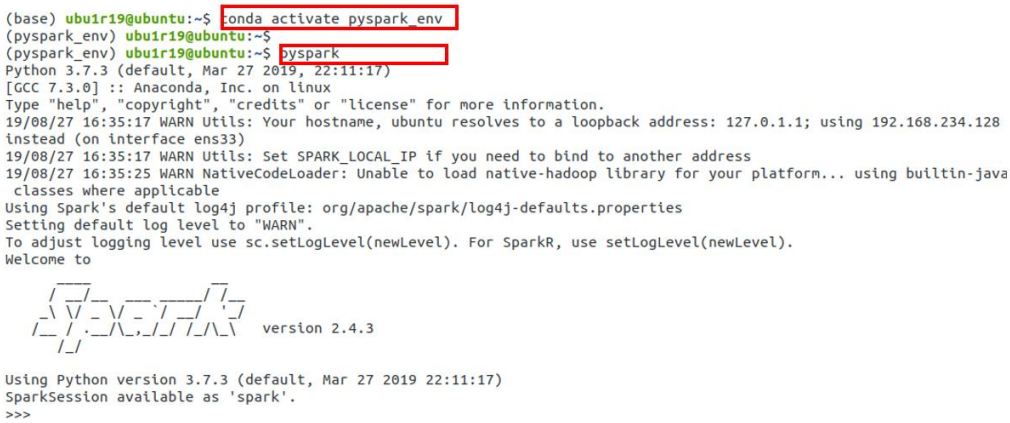
(α)

(β)

Σχήμα 6-41: (α) Δημιουργία περιβάλλοντος conda με ονομασία pyspark_env, (β) Ενεργοποίηση του περιβάλλοντος pyspark_env

Στη συνέχεια θα πρέπει να πληκτρολογήσει *pyspark*, για να μπει στο *shell* (κέλυφος) του *pyspark*. Θα πρέπει να δει τα εξής:

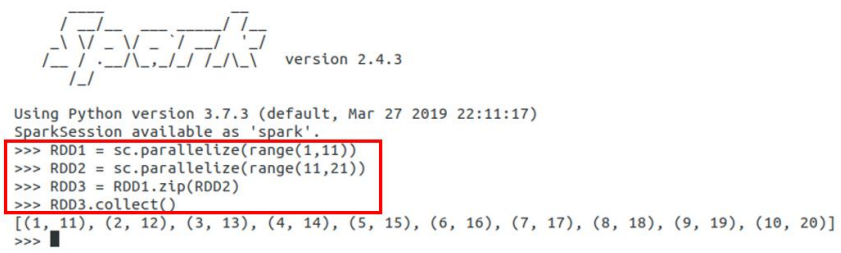
```
pyspark
```



Σχήμα 6-42: Ενεργοποίηση του περιβάλλοντος pyspark_env και ακολούθως έναρξη του κελύφους pyspark

Εκτελώντας τις εντολές που φαίνονται στο Σχήμα 6.43, η έξοδος πρέπει να είναι:

```
[(1, 11), (2, 12), (3, 13), (4, 14), (5, 15), (6, 16), (7, 17), (8, 18), (9, 19), (10,
20)].
```



Σχήμα 6-43: Έλεγχος του Spark με ένα σύντομο πρόγραμμα με RDDs

Για να βγει κανείς από το κέλυφος του pyspark θα πρέπει να πληκτρολογήσει:

```
exit()
```

Μπορεί επίσης να αξιοποιήσει τα χαρακτηριστικά αυτόματης συμπλήρωσης κώδικα (code completion) μέσα στο κέλυφος του pyspark, για παράδειγμα αν πληκτρολογήσει:

```
spark.<tab><tab>
```

θα εμφανιστούν οι μέθοδοι του στιγμιότυπου spark. Αν υπάρχουν πολλές μέθοδοι που αρχίζουν από το ίδιο γράμμα θα χρειαστεί επίσης να πατήσει δυο φορές το *tab* για να του δείξει τις μεθόδους που αρχίζουν από αυτό το γράμμα, διαφορετικά πατώντας το πλήκτρο *tab* συμπληρώνεται η μέθοδος αυτόματα (βλέπε Σχήμα 6.44):

Σχήμα 6-44: Αυτόματη συμπλήρωση κώδικα με το πλήκτρο <tab> - για παράδειγμα πατώντας ο χρήστης το <tab> μετά το spark.B

Βήμα 5ο: Το PySpark μέσω του Jupyter Notebook

Σύμφωνα με τον επίσημο ιστότοπό του, "το Jupyter Notebook είναι μια εφαρμογή web ανοιχτού κώδικα που επιτρέπει τον χρήστη να δημιουργεί και να μοιράζει έγγραφα που περιέχουν "ζωντανό" κώδικα, εξισώσεις, οπτικοποιήσεις και κείμενο αφήγησης. Οι χρήσεις του περιλαμβάνουν: καθαρισμό και μετασχηματισμό δεδομένων, αριθμητική προσομοίωση, στατιστική μοντελοποίηση, οπτικοποίηση δεδομένων, μηχανική μάθηση και πολλά άλλα" [29]. Συνδυάζοντας το PySpark με το Jupyter Notebook, μπορεί ο εκάστοτε χρήστης να υλοποιήσει όλες τις παραπάνω ροές εργασίας.

Υπάρχουν δύο τρόποι για να έχει ο χρήστης το PySpark διαθέσιμο σε ένα Jupyter Notebook:

- Ρυθμίζοντας το πρόγραμμα οδήγησης PySpark να χρησιμοποιεί το Jupyter Notebook, μπορεί ο χρήστης εκτελώντας την εντολή pyspark να ανοίγει αυτόματα στον προεπιλεγμένο φυλλομετρητή του ένα Jupyter Notebook.
- Ανοίγοντας ένα Jupyter Notebook και ακολούθως φορτώνοντας το PySpark χρησιμοποιώντας το πακέτο findSpark.

Η πρώτη επιλογή είναι πιο γρήγορη αλλά η υλοποίησή της είναι συγκεκριμένη για το Jupyter Notebook, ενώ η δεύτερη επιλογή είναι μια ευρύτερη προσέγγιση για να καταστήσει ο χρήστης το PySpark διαθέσιμο στο IDE της επιλογής του. Μερικές φορές χρειάζεται ο χρήστης ένα πλήρες IDE για να δημιουργήσει πιο σύνθετο κώδικα και το PySpark δεν είναι εξ ορισμού στο sys.path, αλλά αυτό δεν σημαίνει ότι δεν μπορεί να χρησιμοποιηθεί ως κανονική βιβλιοθήκη. Το πακέτο findspark προσθέτει το pyspark στο sys.path κατά το χρόνο εκτέλεσης.

Σημείωση: Ο χρήστης θα πρέπει να επιλέξει μια από τις δυο μεθόδους. Αν για παράδειγμα έχει υλοποιήσει την πρώτη μέθοδο και στη συνέχεια θέλει να υλοποιήσει την δεύτερη, θα πρέπει να αφαιρέσει τις μεταβλητές περιβάλλοντος που είχε δημιουργήσει στο βήμα της μεθόδου 1 (PYSPARK_DRIVER_PYTHON και PYSPARK_DRIVER_PYTHON_OPTS) από το αρχείο *.bashrc* και ακολούθως να ανοίξει ένα νέο παράθυρο τερματικού και να ακολουθήσει τα βήματα της μεθόδου 2.

Μέθοδος 1 - Ρύθμιση του προγράμματος οδήγησης PySpark

Θα πρέπει ο χρήστης να καθορίσει τις μεταβλητές περιβάλλοντος του προγράμματος οδήγησης PySpark (PYSPARK_DRIVER_PYTHON και PYSPARK_DRIVER_OPTS). Για το λόγο αυτό θα πρέπει να ανοίξει το *.bashrc* με το *gedit* (βλέπε Σχήμα 6.45):

```
gedit ~/.bashrc
```

Στη συνέχεια θα πρέπει να προσθέσει στο αρχείο τις ακόλουθες γραμμές:

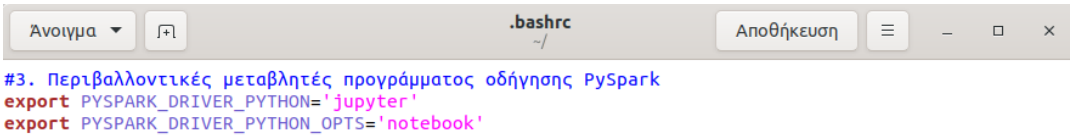
```
export PYSPARK_DRIVER_PYTHON='jupyter'
export PYSPARK_DRIVER_PYTHON_OPTS='notebook'
```

Για να εφαρμοστούν οι αλλαγές θα πρέπει να εκτελέσει την εντολή:

```
source ~/.bashrc
```

Ακολούθως θα πρέπει να ενεργοποιήσει το περιβάλλον *conda* του έργου του (εδώ το *pyspark_env*) και να ανοίξει το *jupyter* με την εντολή *pyspark* (βλέπε Σχήμα 6.46):

```
conda activate pyspark_env
pyspark
```



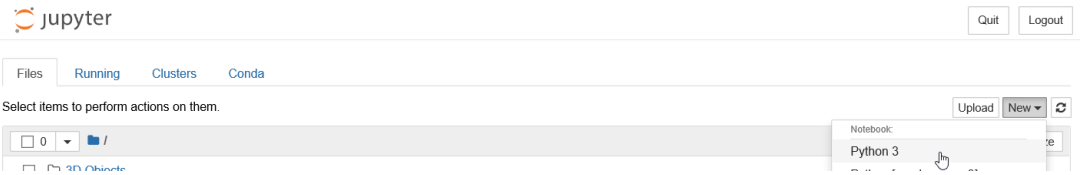
Σχήμα 6-45: Καθορισμός μεταβλητών περιβάλλοντος στο αρχείο .bashrc, έτσι ώστε να εκκινεί το Jupyter notebook όταν ο χρήστης πληκτρολογεί την εντολή *pyspark* στη γραμμή εντολών

Σχήμα 6-46: Εκκίνηση του Jupyter notebook πληκτρολογώντας ο χρήστης την εντολή *pyspark* στην γραμμή εντολών

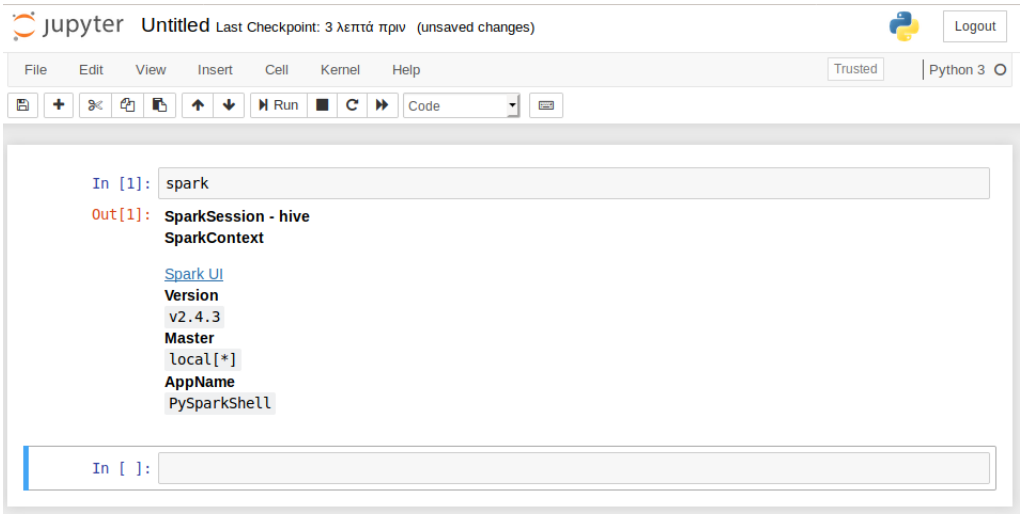
Μέσα στη νέα καρτέλα με ονομασία Home Page που θα ανοίξει στον φυλλομετρητή, θα πρέπει να επιλέξει την καρτέλα *Files* (βλέπε Σχήμα 6.47). Στο δεξί μέρος της υπάρχει η αναδυόμενη λίστα *New* στην οποία επιλέγει τον πυρήνα *python 3* για να δημιουργήσει ένα νέο σημειωματάριο. Στο νέο σημειωματάριο που θα δημιουργηθεί (βλέπε Σχήμα 4.48) θα πρέπει να πληκτρολογήσει τον ακόλουθο κώδικα:

```
spark
```

Θα διαπιστώσει ότι η εγκατάσταση του pyspark λειτουργεί πλέον κανονικά.



Σχήμα 6-47: Δημιουργία ενός νέου python σημειωματάριου μέσα στο jupyter notebook



Σχήμα 6-48: Έλεγχος της διαθεσιμότητας του Spark μέσω του Jupyter Notebook (1η μέθοδος – καθορισμός μεταβλητών περιβάλλοντος PYSPARK_DRIVER_PYTHON και PYSPARK_DRIVER_OPTS μέσα στο αρχείο *.bashrc)*

Μέθοδος 2 - Πακέτο FindSpark

Θα πρέπει ο χρήστης να εγκαταστήσει το *findspark*, μέσα στο εικονικό περιβάλλον του conda (π.χ στο *pyspark_env* – οδηγίες σχετικά με τα περιβάλλοντα conda παρέχονται στην ενότητα Διαχείριση έργων) για να αποκτήσει πρόσβαση σε ένα στιγμιότυπο spark από το jupyter notebook (βλέπε Σχήμα 4.49):

```
conda activate pyspark_env
conda list findspark
# Εγκατάσταση του findspark από το αποθετήριο conda-forge
conda install -c conda-forge findspark
conda list findspark
```

```
(pyspark_env) ubu1r19@ubuntu:~$ conda list findspark
# packages in environment at /home/ubu1r19/anaconda3/envs/pyspark_env:
#
# Name                    Version                   Build  Channel
(pyspark_env) ubu1r19@ubuntu:~$ conda install -c conda-forge findspark
Collecting package metadata (current_repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 4.7.10
  latest version: 4.7.11

Please update conda by running

    $ conda update -n base -c defaults conda

...
(pyspark_env) ubu1r19@ubuntu:~$ conda list findspark
# packages in environment at /home/ubu1r19/anaconda3/envs/pyspark_env:
#
# Name                    Version                   Build  Channel
findspark                 1.3.0                      py_1    conda-forge
(pyspark_env) ubu1r19@ubuntu:~$
```

Σχήμα 6-49: - Εγκατάσταση του πακέτου findspark – προβολή του πακέτου findspark

Ακολούθως θα πρέπει να εκτελέσει της παρακάτω εντολές. Η πρώτη κάνει ενημέρωση του πακέτου conda στο βασικό (base) περιβάλλον *conda* και η δεύτερη εγκαθιστά το *jupyter* μέσα στο περιβάλλον *pyspark_env*:

```
conda update -n base -c defaults conda
conda install jupyter
```

Για να ανοίξει το jupyter θα πρέπει να πληκτρολογήσει:

```
jupyter notebook
```

Στη συνέχεια στην αναδυόμενη λίστα New της καρτέλας files, θα πρέπει να επιλέξει τον πυρήνα python 3 για να δημιουργήσει ένα νέο python σημειωματάριο. Στο νέο σημειωματάριο που θα δημιουργηθεί (βλέπε Σχήμα 6.50) θα πρέπει να πληκτρολογήσει τον ακόλουθο κώδικα:

```
import findspark
findspark.init()
findspark.find()
```

Σχήμα 6-50: Εισαγωγή του πακέτου findspark στο Jupyter Notebook και εύρεση της διαδρομής στην οποία είναι εγκατεστημένο (2η μέθοδος – findspark)

Με τις παρακάτω εντολές ο χρήστης δημιουργεί ένα στιγμιότυπο του SparkSession (βλέπε Σχήμα 4.51):

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('iannis_ap').getOrCreate()
spark
```

Σχήμα 6-51: Έλεγχος της διαθεσιμότητας του Spark μέσω του Jupyter Notebook (2η μέθοδος - findspark)

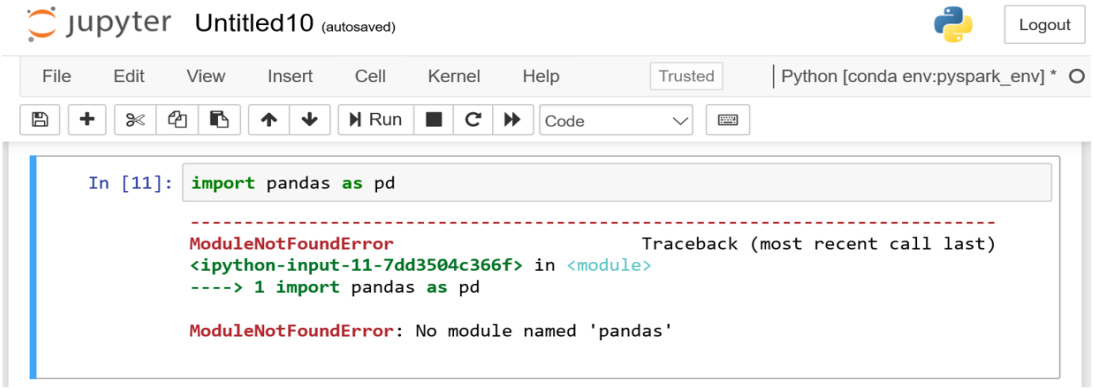
Σημείωση: Θα πρέπει να επιλέξει και να υλοποιήσει κανείς την μέθοδο που τον εξυπηρετεί καλύτερα.

Εκκίνηση / Σταμάτημα του Jupyter Notebook

Η εκκίνηση / σταμάτημα του διακομιστή Jupyter Notebook γίνεται από τη γραμμή εντολών:

```
# Εκκίνηση του διακομιστή Jupyter Notebook:
# * Για την μέθοδο 1, μέσω του pyspark
pyspark

# * Για την μέθοδο 2
jupyter notebook

# Σταμάτημα του διακομιστή Jupyter Notebook
Ctrl+c
```

Εάν παρατηρήσει ο χρήστης κάποιο λάθος στο σημειωματάριό (notebook) του που διαμαρτύρεται για το ότι δεν βρέθηκε το *pandas* (βλέπε Σχήμα 6.52), παρόλο που το εγκατέστησε ήδη στο περιβάλλον *conda*, θα πρέπει να βεβαιωθεί ότι έχει επιλέξει τον σωστό περιβάλλον.



Σχήμα 6-52: Σε περίπτωση μηνύματος λάθους κατά την εισαγωγή της πακέτου στο jupyter, ενώ είναι σίγουρος ο χρήστης ότι το έχει εγκαταστήσει, θα πρέπει να βεβαιωθεί ότι έχει επιλέξει το σωστό περιβάλλον conda.

Βήμα 6ο (προαιρετικό): Εγκατάσταση της επέκτασης nb_conda για πρόσβαση στο περιβάλλον conda και εγκατάσταση πακέτων μέσα από το Jupyter.

Το *nb_conda* επιτρέπει στους χρήστες να έχουν πρόσβαση σε πυρήνες από άλλα περιβάλλοντα *conda* μέσα από το *Jupyter*. Για το λόγο αυτό θα πρέπει ο χρήστης να ενεργοποιήσει πρώτα το *conda* περιβάλλον του και ακολούθως να εγκαταστήσει το *nb_conda* (βλέπε Σχήμα 6.53):

```
conda activate pyspark_env
conda install nb_conda
```

```
(base) ubu1r19@ubuntu:~$ conda activate pyspark_env
(pyspark_env) ubu1r19@ubuntu:~$ conda install nb_conda
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/ubu1r19/anaconda3/envs/pyspark_env

  added / updated specs:
    - nb_conda

...
```

Σχήμα 6-53: Εγκατάσταση της επέκτασης nb_conda

Στη συνέχεια προκειμένου να εφαρμοστούν οι αλλαγές θα πρέπει να απενεργοποιήσει το περιβάλλον του και κατόπιν να το ενεργοποιήσει εκ νέου.

```
conda deactivate
conda activate pyspark_env
```

Θα πρέπει ακολούθως να εκκινήσει το *Jupyter Notebook* (βλέπε Σχήμα 6.54):

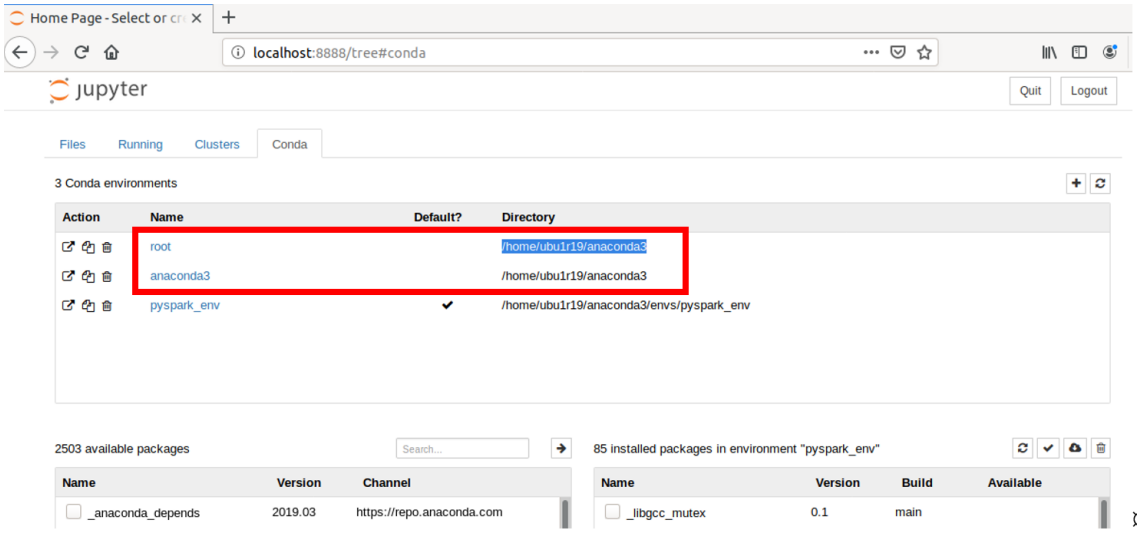
```
jupyter notebook
```

```
(pyspark_env) ubu1r19@ubuntu:~$ conda deactivate
(base) ubu1r19@ubuntu:~$ conda activate pyspark_env
(pyspark_env) ubu1r19@ubuntu:~$ jupyter notebook
[I 17:36:31.120 NotebookApp] [nb_conda_kernels] enabled, 2 kernels found
[I 17:36:33.449 NotebookApp] [nb_conda] enabled
[I 17:36:33.449 NotebookApp] Serving notebooks from local directory: /home/ubu1r19
[I 17:36:33.449 NotebookApp] The Jupyter Notebook is running at:
[I 17:36:33.450 NotebookApp] http://localhost:8888/?token=2d8f5b29b0338b0e8dcc150130e69e3c8d4258f67dc9fb29
[I 17:36:33.450 NotebookApp]  or http://127.0.0.1:8888/?token=2d8f5b29b0338b0e8dcc150130e69e3c8d4258f67dc9fb29
[I 17:36:33.450 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmat
[C 17:36:33.674 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///home/ubu1r19/.local/share/jupyter/runtime/nbserver-3202-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=2d8f5b29b0338b0e8dcc150130e69e3c8d4258f67dc9fb29
     or http://127.0.0.1:8888/?token=2d8f5b29b0338b0e8dcc150130e69e3c8d4258f67dc9fb29
```

Σχήμα 6-54: Εκκίνηση του Jupyter Notebook

Στο παραπάνω σχήμα παρατηρεί κανείς ότι κατά την εκκίνησή του το Jupyter Notebook εντόπισε δυο *nb_conda* πυρήνες. Επίσης όπως βλέπει κανείς στο Σχήμα 6.55 στο περιβάλλον του jupyter δημιουργήθηκε μια επιπλέον καρτέλα με ονομασία *conda* όπου φαίνονται ποια είναι τα εικονικά περιβάλλοντα *conda* που έχει δημιουργήσει ο χρήστης, ποιο περιβάλλον είναι ενεργό και επιτρέπει στον χρήστη να αλλάξει το ενεργό περιβάλλον. Υπάρχει όμως ένα μικρό πρόβλημα που πρέπει να διορθωθεί (το πρόβλημα δεν είναι μόνο οπτικό αλλά και λειτουργικό). Η καρτέλα αυτή εμφανίζει 3 περιβάλλοντα *conda*, απ' τα οποία τα δυο (*root, Anaconda3*) παραπέμπουν στην ίδια διαδρομή (*/home/ubu1r19/anaconda3*). Το πρόβλημα αυτό οφείλεται σε αλλαγή της εξόδου *conda info --json* στις νέες εκδόσεις του conda (error#66) και απαιτεί μια μικρή παρέμβαση από μέρους του χρήστη για να διορθωθεί το πρόβλημα (αν στη εγκατάσταση του χρήστη δεν εμφανίζεται το πρόβλημα αυτό, σημαίνει ότι θα έχει διορθωθεί με κάποια μεταγενέστερη ενημέρωση, οπότε μπορεί να συνεχίσει με τα βήματα μετά το Σχήμα 6.58). Ο πιο εύκολος τρόπος είναι να ανοίξει ο χρήστης τη γραμμή εντολών και να αναζητήσει που βρίσκεται το αρχείο *envmanager.py* (βλέπε Σχήμα 4.56) πληκτρολογώντας την παρακάτω εντολή (πιθανόν να αργήσει λίγο αλλά θα βρει τα αρχεία). Βλέπει κανείς ότι στον υπολογιστή της εγκατάστασης υπάρχουν τρία αρχεία, ο χρήστης ενδιαφέρεται για αυτό που βρίσκεται στον υποκατάλογο *nb_conda* στη διαδρομή *.../envs/pyspark_env/lib/python3.7/site-packages/nb_conda/envmanager.py* μέσα στο περιβάλλον του (στην παρούσα εγκατάσταση το *pyspark_env*).

```
find ~/ -name 'envmanager.py'
```

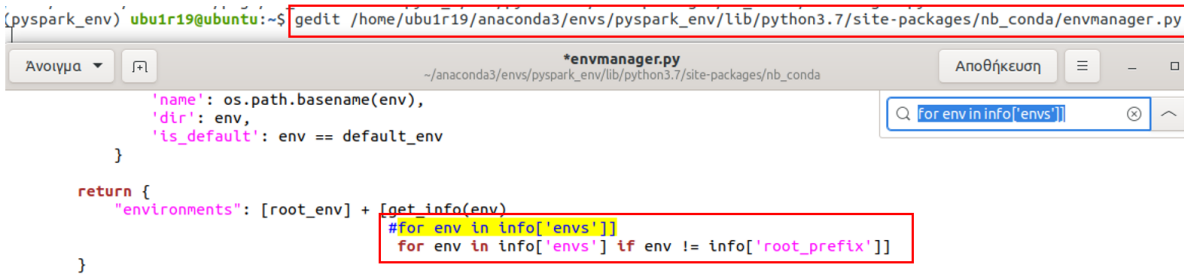


Σχήμα 6-55: Μετά την εγκατάσταση του πρόσθετου nb_conda δημιουργείται στο jupyter notebook η καρτέλα conda, στην οποία φαίνονται τα εικονικά περιβάλλοντα conda που έχει δημιουργήσει ο χρήστης, το ενεργό περιβάλλον και που επιτρέπει στον χρήστη την αλλαγή ενεργού περιβάλλοντος και την προσθήκη νέων πακέτων μέσω του jupyter.

```
(pyspark_env) ubu1r19@ubuntu:~$ find ~/ -name 'envmanager.py'
/home/ubu1r19/anaconda3/envs/pyspark_env/lib/python3.7/site-packages/nb_conda/envmanager.py
/home/ubu1r19/anaconda3/pkgs/nb_conda-2.2.1-py37_0/lib/python3.7/site-packages/nb_conda/envmanager.py
(pyspark_env) ubu1r19@ubuntu:~$
```

Σχήμα 6-56: Αναζήτηση του προγράμματος *envmanager.py* με την εντολή *find*

Στη συνέχεια αφού το ανοίξει με κάποιον επεξεργαστή κώδικα όπως το gedit, θα πρέπει να πατήσει *CTR+F* και να εισάγει στο παράθυρο αναζήτησης : *for env in info['envs']]* (βλέπε Σχήμα 6.57). Θα βρει μια αναφορά την οποία θα πρέπει να αντικαταστήσει με την: *for env in info['envs'] if env != info['root_prefix']]*.



Σχήμα 6-57: Εύρεση της έκφρασης 'for env in info['envs']]' και αντικατάστασή της με την 'for env in info['envs'] if env != info['root_prefix']]'

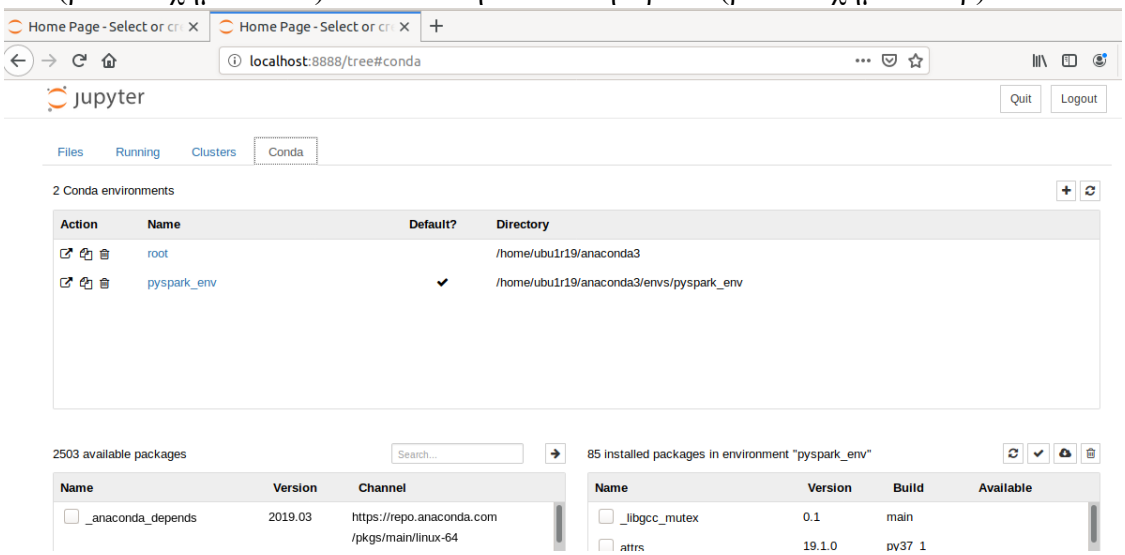
Αφού σώσει το αρχείο θα πρέπει να κλείσει τη γραμμή εντολών εάν ήταν ήδη ανοικτή και στη συνέχεια να ανοίξει μια νέα όπου θα εκκινήσει εκ νέου το jupyter:

```
jupyter notebook
```

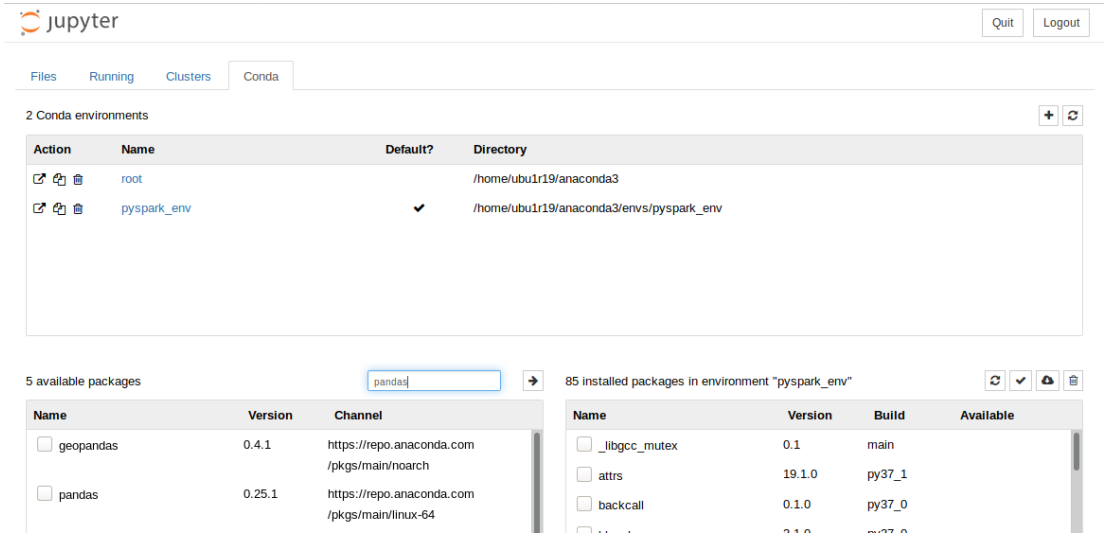
Όπως φαίνεται στο Σχήμα 6.58, το πρόβλημα αποκαταστάθηκε. Από την καρτέλα *conda* μπορεί ο χρήστης να εγκαθιστά και να απεγκαθιστά πακέτα σε ένα περιβάλλον *conda*. Μπορεί επίσης να διαχειρίζεται τα διάφορα περιβάλλοντα (προσθήκη / διαγραφή / κλωνοποίηση).

Για να προσθέσει ο χρήστης στο περιβάλλον *conda* με ονομασία *pyspark_env* το πακέτο που ήθελε (*pandas*) θα πρέπει να ακολουθήσει τα παρακάτω βήματα:

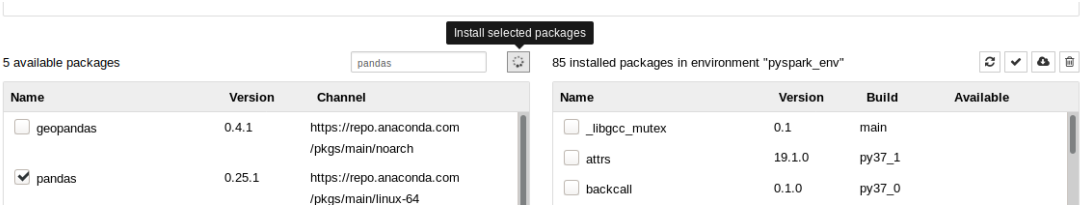
- Θα πρέπει πρώτα να επιλέξει το εν λόγω περιβάλλον (βλέπε Σχήμα 6.59).
- Στη συνέχεια να κάνει ανανέωση των πακέτων με το πλήκτρο
- Ακολούθως θα πρέπει να επιλέξει το πακέτο ή τα πακέτα που θέλει να εγκαταστήσει (βλέπε Σχήμα 6.60α) και να πατήσει το πλήκτρο (βλέπε Σχήμα 6.60β).



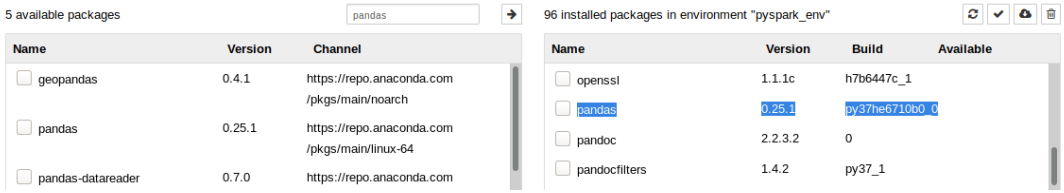
Σχήμα 6-58: Το πρόβλημα (βλέπε Σχήμα 6.55) όπου στην καρτέλα conda δυο περιβάλλοντα τα root και Anaconda3 παρέπεμπαν στην ίδια διαδρομή αποκαταστάθηκε



Σχήμα 6-59: Επιλογή του pyspark_env ως ενεργό περιβάλλον και αναζήτηση του πακέτου pandas μέσα από την καρτέλα conda

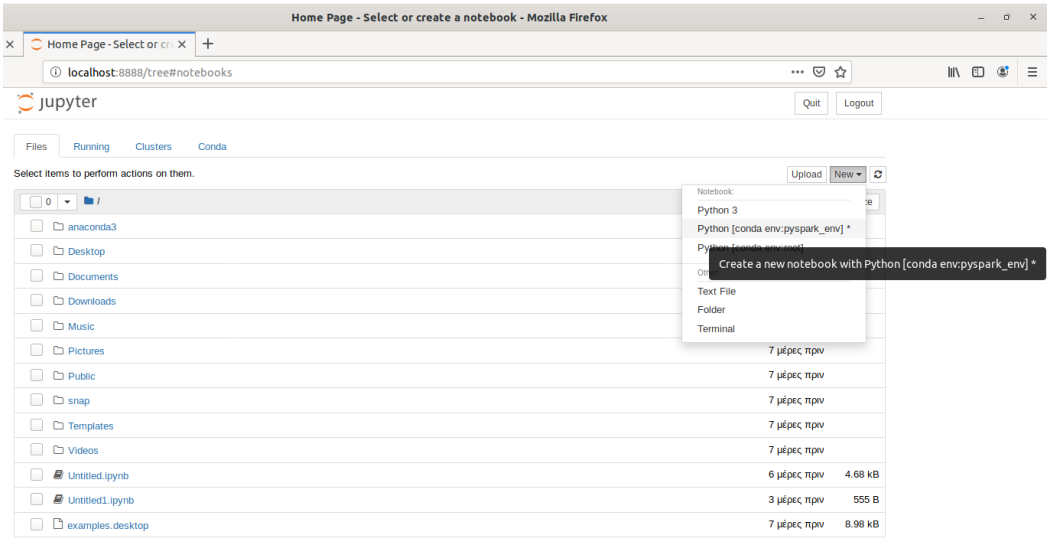


(α)

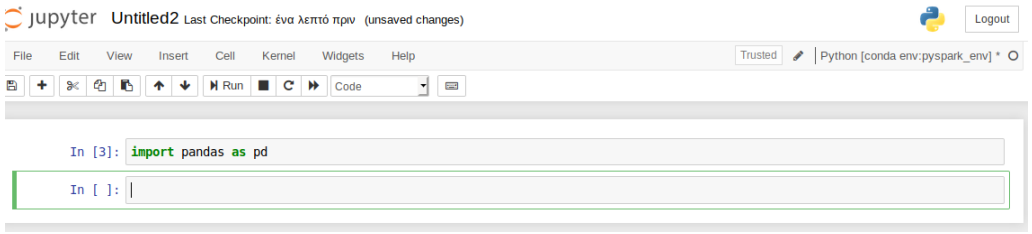


(β)

Σχήμα 6-60: Εγκατάσταση του πακέτου pandas μέσα από την καρτέλα conda



(α)



(β)

Σχήμα 6-61: (α) Από την καρτέλα files, μέσω της αναδυόμενης λίστα New μπορεί να επιλέξει ο χρήστης τον πυρήνα που επιθυμεί για το νέο jupyter notebook, (β) Επιτυχής εισαγωγή του πακέτου pandas στο νέο σημειωματάριο jupyter (ο αριθμό μέσα στις αγκύλες In [3]) αριστερά της εντολής υποδεικνύει την επιτυχή εκτέλεση της εντολής)

Από την καρτέλα files, μέσω της αναδυόμενης λίστα New μπορεί να επιλέξει ο χρήστης τον πυρήνα που επιθυμεί (όσες επιλογές έχουν την ένδειξη conda env:env_name, είναι συνδυασμός πυρήνα-περιβάλλοντος conda) για το σημειωματάριο (notebook) που θέλει να δημιουργήσει (conda env:pyspark_env στην περίπτωσή μας – βλέπε Σχήμα 6.61α). Στη

συνέχεια μέσα στο σημειωματάριο που δημιουργήθηκε εισάγουμε το pandas με την εντολή (βλέπε Σχήμα 6.61β):

```
import pandas as pd
```

Και παρατηρούμε ότι εκτελέστηκε κανονικά (όταν έχει αριθμό μέσα στις αγκύλες π.χ. In [3]) μετά την εγκατάσταση του πακέτου.

Τέλος μια καλή συνήθεια είναι να αποθηκεύει ο προγραμματιστής το περιβάλλον conda σε ένα αρχείο yaml (αρχείο με επέκταση .yml, βλέπε ενότητα "Αποθήκευση περιβάλλοντος"). Για τον λόγο αυτό θα πρέπει ο χρήστης να δημιουργήσει έναν κατάλογο μέσα στον οποίο θα αποθηκεύσει το αρχείο yaml (*YAML Ain't Markup Language*):

```
# Ενεργοποίηση του περιβάλλοντος conda με ονομασία pyspark_env
conda activate pyspark_env
# Δημιουργία φακέλου για την αποθήκευση του περιβάλλοντος conda
mkdir ~/Downloads/pyspark_env_dir
# Αποθήκευση του τρέχοντος περιβάλλοντος conda
conda env export > ~/Downloads/pyspark_env_dir/pyspark_env.yml
ls -al ~/Downloads/pyspark_env_dir
# Το yaml είναι ένα φιλικό προς τον άνθρωπο πρότυπο σειριοποίησης δεδομένων για όλες τις γλώσσες προγραμματισμού.
sudo gedit ~/Downloads/pyspark_env_dir/pyspark_env.yml
```

Εγκατάσταση του Spark σε μια συστάδα Hadoop YARN

Σε προηγούμενη ενότητα παρουσιάστηκε ο τρόπος εγκατάστασης του Spark σε τοπική λειτουργία εκτέλεσης (local mode) τόσο σε λειτουργικό Σύστημα Windows όσο και σε Linux. Στη συνέχεια παρουσιάζεται ο τρόπος εγκατάστασης του Spark σε μια συστάδα Hadoop YARN αποτελούμενη από εικονικές μηχανές με εγκατεστημένο λειτουργικό σύστημα Linux (UBUNTU 19.04). Οι όροι "κόμβος" και "εικονική μηχανή" χρησιμοποιούνται εναλλάξιμα σε αυτή την διπλωματική εργασία και αναφέρονται σε μια ΕΜ (Εικονική Μηχανή) που περιέχει μια ή περισσότερες διεργασίες Hadoop.

Πλάνο εγκατάστασης του Spark σε συστάδα Hadoop YARN και σε Standalone συστάδα

Θα πρέπει σε πρώτη φάση να δημιουργηθεί η συστάδα Hadoop YARN με τις εικονικές μηχανές και στην συνέχεια να ακολουθήσει η εγκατάσταση του Spark. Θα πρέπει να τονιστεί ότι δεν είναι απαραίτητη η εγκατάσταση του Spark σε όλους τους κόμβους της συστάδας (αρκεί σε έναν κόμβο) παρόλα αυτά θα εγκατασταθεί σε όλους τους κόμβους πρώτον για να μπορεί να χρησιμοποιηθεί η εγκατάσταση της συστάδας και για την λειτουργία του Spark με τον εγγενή διαχειριστή συστάδας του Spark (Spark Standalone Cluster Manager) και δεύτερον με αυτό τον τρόπο απλοποιείται η διαδικασία της εγκατάστασης καθώς οι κλώνοι δε θα χρειαστούν πολλές τροποποιήσεις.

Για τη δημιουργία των εικονικών μηχανών θα χρησιμοποιηθεί το λογισμικό ανοικτού κώδικα *Oracle VirtualBox* που είναι μια ανεξαρτήτου πλατφόρμας εφαρμογή εικονικοποίησης. Το πλάνο εγκατάστασης προβλέπει την εγκατάσταση , σε μια *ΕΜ - Εικονική Μηχανή (VM - Virtual Machine)*, όλων των προαπαιτούμενων προγραμμάτων που απαριθμούνται στον παρακάτω πίνακα και την πραγματοποίηση όλων των απαραίτητων ρυθμίσεων στα αρχεία διαμόρφωσης των προγραμμάτων αυτών και του λειτουργικού συστήματος και ακολούθως την δημιουργία τριών κλώνων (*slave1, slave2 και spark-client*) της ΕΜ όπου θα πραγματοποιηθούν οι επιμέρους τροποποιήσεις στα αρχεία διαμόρφωσης. Ακολουθώντας τις οδηγίες εγκατάστασης ο εκάστοτε χρήστης μπορεί αν το επιθυμεί να δημιουργήσει μια συστάδα n-κόμβων (όπου n = ο αριθμός των κόμβων που ικανοποιεί τις ανάγκες του).

Πίνακας 6-1: Προαπαιτούμενο λογισμικό για την εγκατάσταση του Spark σε συστάδα Hadoop/YARN και σε Standalone Mode

Α/Α	**Προαπαιτούμενο Λογισμικό**	**Αρχεία διαμόρφωσης**
1	Λειτουργικό **UBUNTU** 19.04 και πρόσθετα επισκέπτη VirtualBox:	▪ /etc/profile ▪ /etc/hosts ▪ ~/.bashrc
2	**Java**: openjdk-8-jdk	
3	**Anaconda3**: Anaconda3-2019.07-Linux-x86_64.sh *Home path: /opt/anaconda*	
4	**Hadoop**: hadoop-3.2.1.tar.gz[24] *Home path: /usr/local/hadoop*	▪ /etc/profile.d/hadoop.sh Στον κατάλογο **/usr/local/hadoop/etc/hadoop/**: ▪ hadoop-env.sh ▪ workers ▪ core-site.xml ▪ hdfs-site.xml ▪ mapred-site.xml ▪ yarn-site.xml
5	**Spark**: *Home path: /opt/spark*	Στον κατάλογο **/opt/spark/conf**: ▪ slaves[25] ▪ spark-defaults.conf ▪ spark-env.sh

[24] *Τόσο σε συστάδα Hadoop YARN όσο και σε συστάδα Spark σε Standalone mode που τρέχει παράλληλα με το Hadoop HDFS για να εκμεταλλεύεται τις δυνατότητες του κατανεμημένου συστήματος αρχείων.*
[25] *Σε Spark Standalone mode*

Προκαταρκτικές Εργασίες πριν την δημιουργία των Εικονικών Μηχανών

Πριν την δημιουργία της ΕΜ θα πρέπει ο χρήστης να κάνει κάποιες επιπλέον προκαταρκτικές εργασίες. Θα πρέπει να προαποφασίσει σε ποια θέση (βλέπε *home path* στον Πίνακα 6-1) θα εγκαταστήσει τα Anaconda, Hadoop και Spark, ποιες στατικές διευθύνσεις θα δώσει στις ΕΜ του, ποιο hostname και πόσους επεξεργαστές θα έχει κάθε ΕΜ, πόση συνολική μνήμη σε GB, καθώς και ποια θα είναι η χωρητικότητα του σκληρού δίσκου σε GB κάθε ΕΜ .

Παρακάτω παρατίθεται ο Πίνακας 6-2 με τις επιμέρους ρυθμίσεις που επιλέχτηκαν για την εκάστοτε ΕΜ στην παρούσα διπλωματική εργασία καθώς και ο Πίνακας 6-3 με το Όνομα χρήστη και το Όνομα Ομάδας για τις ΕΜ.

Πίνακας 6-2: Επιμέρους ρυθμίσεις ΕΜ

Όνομα ΕΜ	Διεύθυνση IP	Όνομα υπολογιστή (Hostname)	HDFS Daemons	YARN Daemons	Spark Daemons & Processes	Αριθμός CPU	Μνήμη (GB)	Χωρητικότητα HDD (GB)
hadoop-master	192.168.1.100	master	NameNode Secondary NameNode	ResourceManager	Master**[26]	1	3	40
hadoop-slave1	192.168.1.101	slave1	DataNode	NodeManager	Worker** executors*	2[27]	4	40
hadoop-slave2	192.168.1.102	slave2	DataNode	NodeManager	Worker** executors*[28]	2	4	40
spark-client	192.168.1.110	spark-client			driver*[29]	1	2	40

Πίνακας 6-3: Όνομα χρήστη και Όνομα Ομάδας για τις ΕΜ

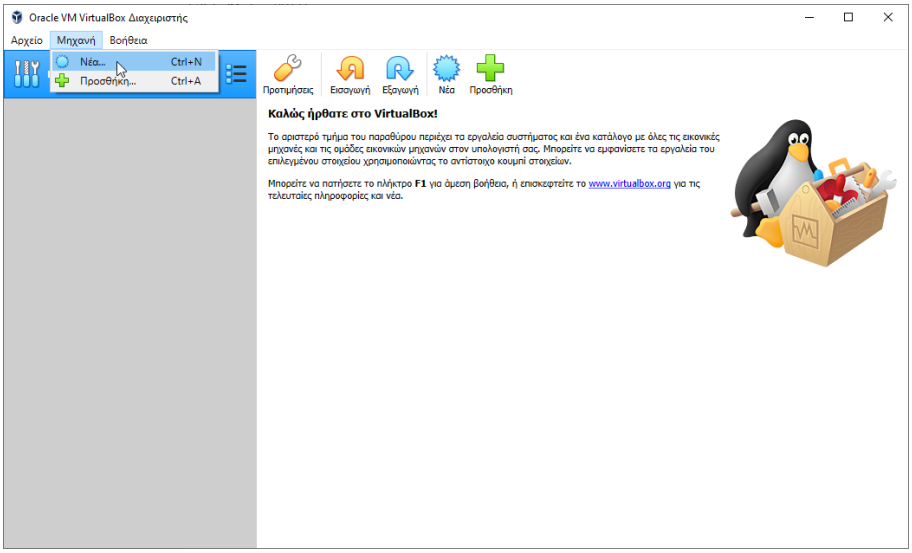
username	groupname
hadoopuser	hadoopgroop

Δημιουργία Εικονικής Μηχανής με την εφαρμογή Oracle VM VirtualBox

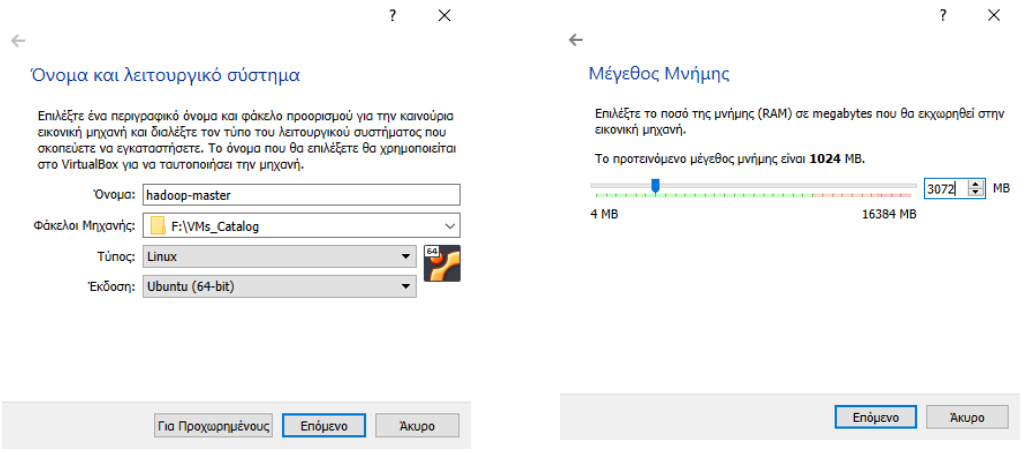
Για να δημιουργήσει ο χρήστης μια ΕΜ θα πρέπει να ανοίξει την εφαρμογή Oracle VM VirtualBox και ακολούθως να κάνει κλικ στο κουμπί "Νέα" για να ανοίξει ένα παράθυρο

[26] **: *Σε Spark Standalone mode*
[27] *Οι 2 πυρήνες αντιπροσωπεύουν τον "ελάχιστο" παραλληλισμό.*
[28] *:*Ο εκτελεστής (executor) και ο οδηγός (driver) είναι διεργασίες (processes)*
[29] *: *Σε client mode, τόσο σε συστάδα Hadoop Yarn όσο και σε Spark Standalone mode (επί του παρόντος, η αυτόνομη λειτουργία δεν υποστηρίζει cluster mode για εφαρμογές Python). Σε cluster mode σε συστάδα Hadoop YARN ο ApplicationsManager (ένα συστατικό του ResourceManager) αφού επικυρώσει οτι οι απαιτούμενοι από την υποβληθείσα εφαρμογή πόροι για τον ApplicationMaster είναι διαθέσιμοι, αναπτύσσει τον driver εντός του ApplicationMaster (το κύριο container που ζητά, ξεκινά και παρακολουθεί τους πόρους της εφαρμογής,), σε κάποιον από τους διαθέσιμους slave κόμβους.*

διαλόγου και στη συνέχεια να ακολουθήσει τα καθοδηγούμενα βήματα της εφαρμογής όπως φαίνεται και στα παρακάτω σχήματα.



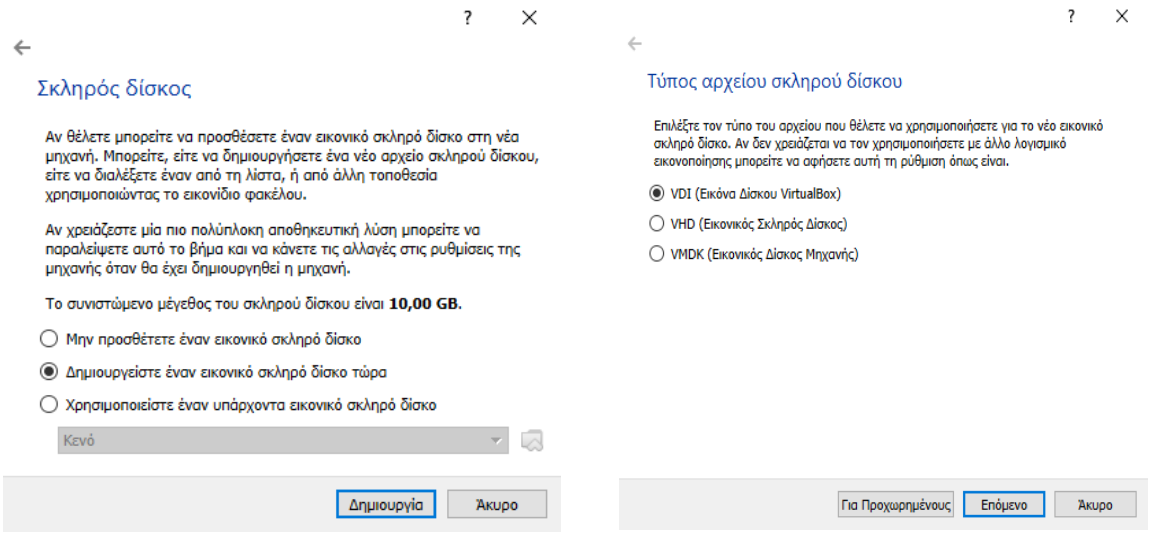
Σχήμα 6-62: Δημιουργία νέας Εικονικής Μηχανής (ΕΜ)



(α) (β)

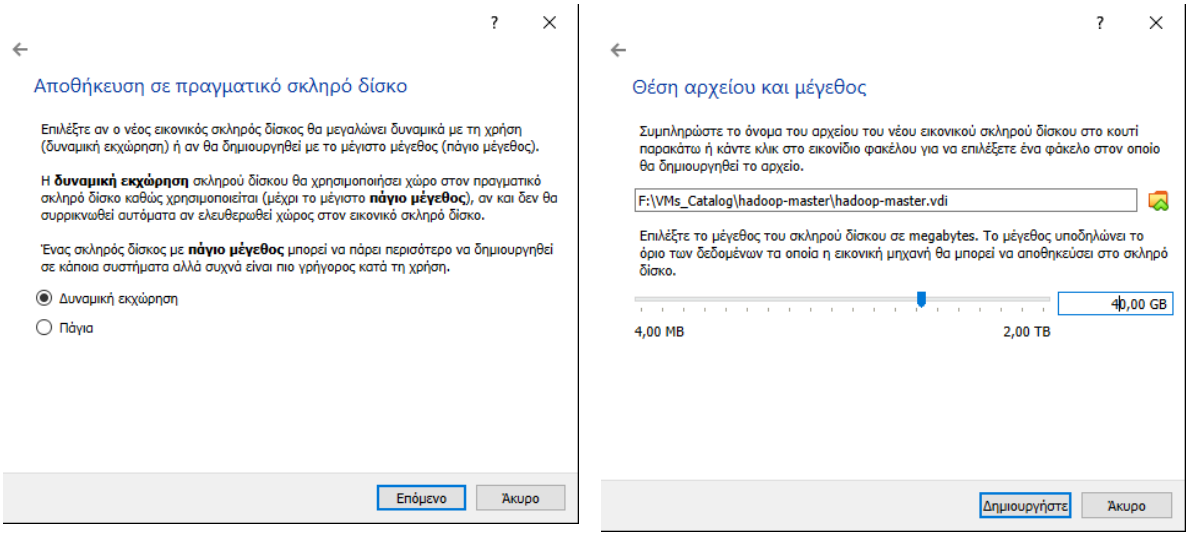
Σχήμα 6-63: (α) Επιλογή ονόματος, φάκελου προορισμού και τύπο Λειτουργικού Συστήματος για την ΕΜ , (β) Εκχώρηση μνήμης RAM στην ΕΜ

Το μέγεθος της μνήμης της ΕΜ (βλέπε Σχήμα 6-63β) που επιλέγει ο χρήστης εξαρτάται από πολλούς παράγοντες, όπως το μέγεθος μνήμης του οικοδεσπότη (host) υπολογιστή, από την προοριζόμενη χρήση της ΕΜ, από τον αριθμό και τη μνήμη που καταναλώνουν συνολικά οι υπόλοιπες ΕΜ που θα τρέχουν ταυτόχρονα, κλπ. Θα πρέπει να σημειωθεί ότι το VirtualBox θα δημιουργήσει μια "κατάτμηση εναλλαγής" (swap) με το ίδιο μέγεθος όπως η μνήμη βάσης που εισάγει εδώ ο χρήστης. Έτσι, αργότερα, όταν επιλέξει το μέγεθος του εικονικού σκληρού δίσκου, θα πρέπει να το λάβει αυτό υπόψη του.



(α) (β)

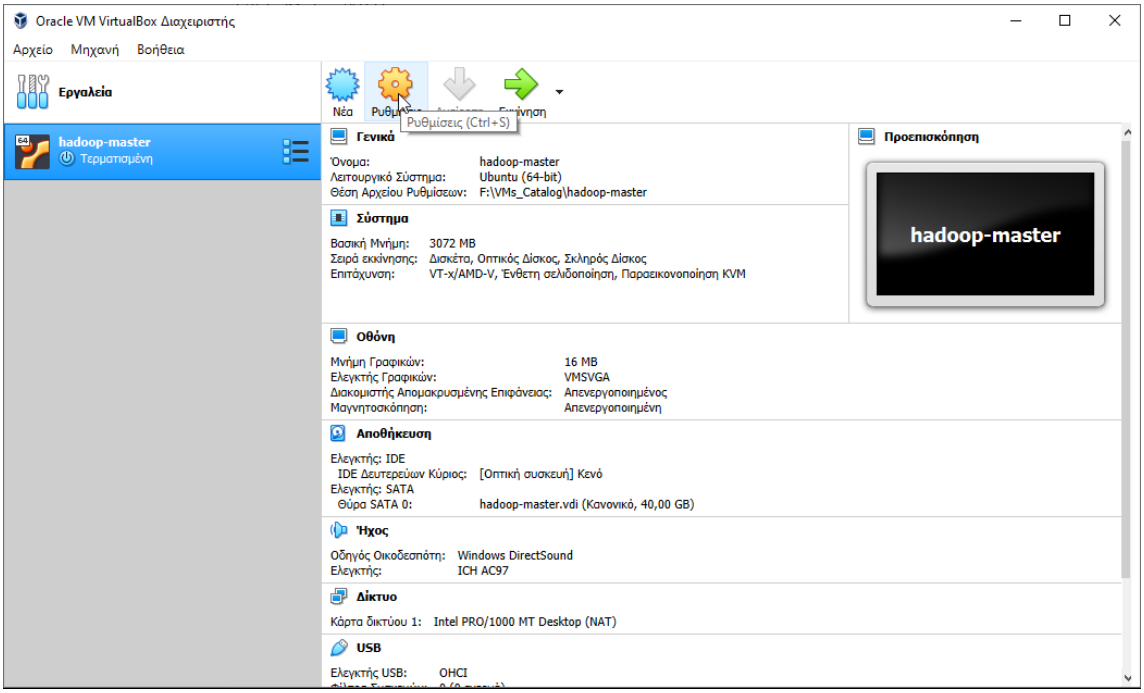
Σχήμα 6-64 (α) Επιλογές για Δημιουργία Εικονικού Σκληρού Δίσκου (ΕΣΔ) στην ΕΜ, (β) Τύπος αρχείου ΕΣΔ



(α) (β)

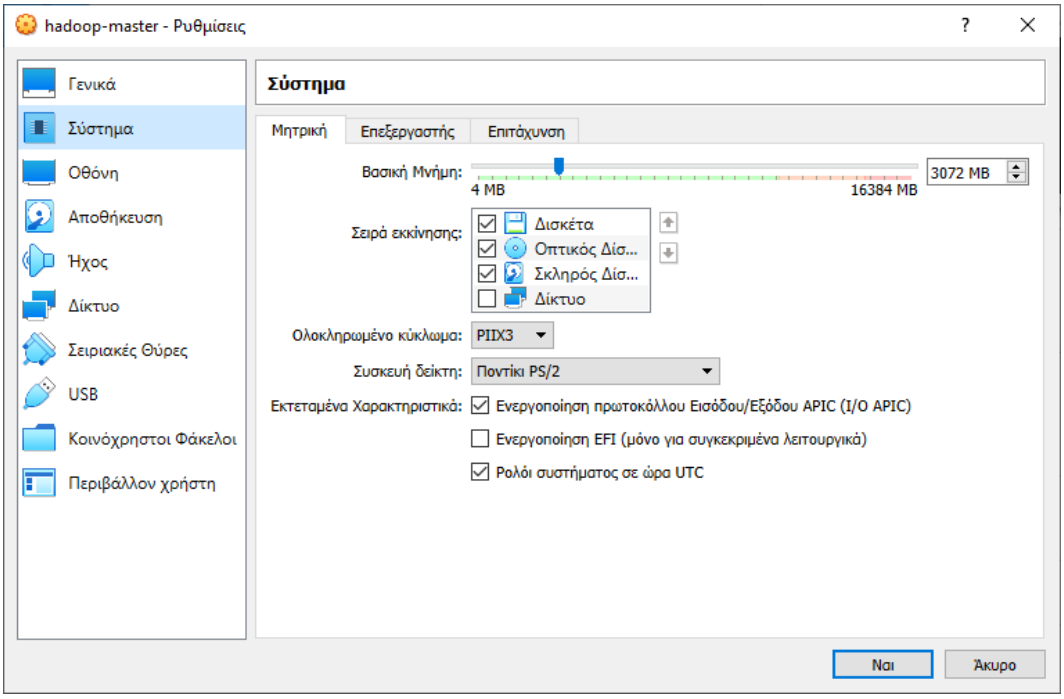
Σχήμα 6-65 (α) Επιλογή Στατικής ή Δυναμικής εκχώρησης για τον ΕΣΔ, (β) Θέση αρχείου ΕΣΔ και μέγεθος ΕΣΔ

Ο χρήστης θα πρέπει να επιλέξει (βλέπε Σχήμα 6.65β) ένα ικανοποιητικό μέγεθος για να αποφύγει την επέκταση του εικονικού σκληρού δίσκου μελλοντικά.

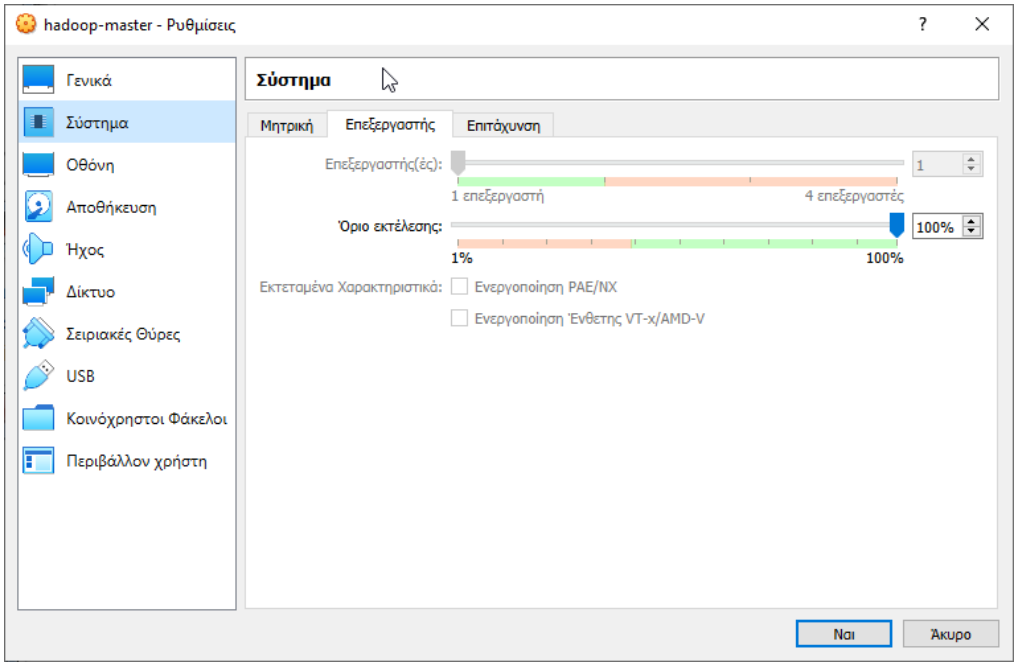


Σχήμα 6-66: Ρυθμίσεις ΕΜ

Αφού επιλέξει τη νέα του ΕΜ θα πρέπει να κάνει κλικ στο κουμπί "Ρυθμίσεις" όπως φαίνεται στην παραπάνω εικόνα.



Σχήμα 6-67: Μνήμη ΕΜ & σειρά εκκίνησης

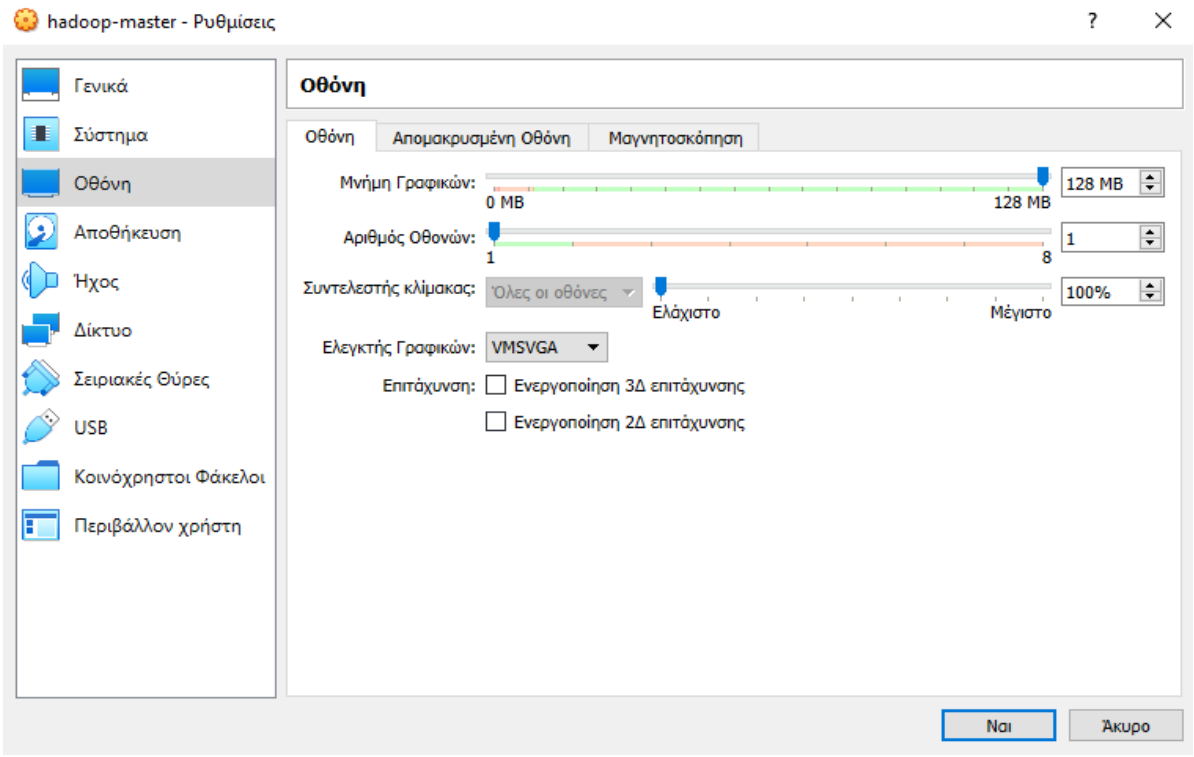


Σχήμα 6-68: Επιλογή αριθμού επεξεργαστών και όριο εκτέλεσης (πόρων) ΕΜ

Ο χρήστης μπορεί να αλλάξει τον αριθμό των επεξεργαστών (βλέπε Σχήμα 6-68) κάνοντας κλικ στην επιλογή 'Σύστημα' και ακολούθως επιλέγοντας την καρτέλα ‘Επεξεργαστής’ στα δεξιά της οθόνης και σύροντας το ρυθμιστικό δίπλα στο "Επεξεργαστής (ες)" προς τα δεξιά μέχρι την τιμή που επιθυμεί. Ο μέγιστος αριθμός που μπορεί να επιλέξει είναι ίσος με τον αριθμό των επεξεργαστών ή πυρήνων επεξεργαστών που είναι εγκατεστημένοι στον υπολογιστή του.

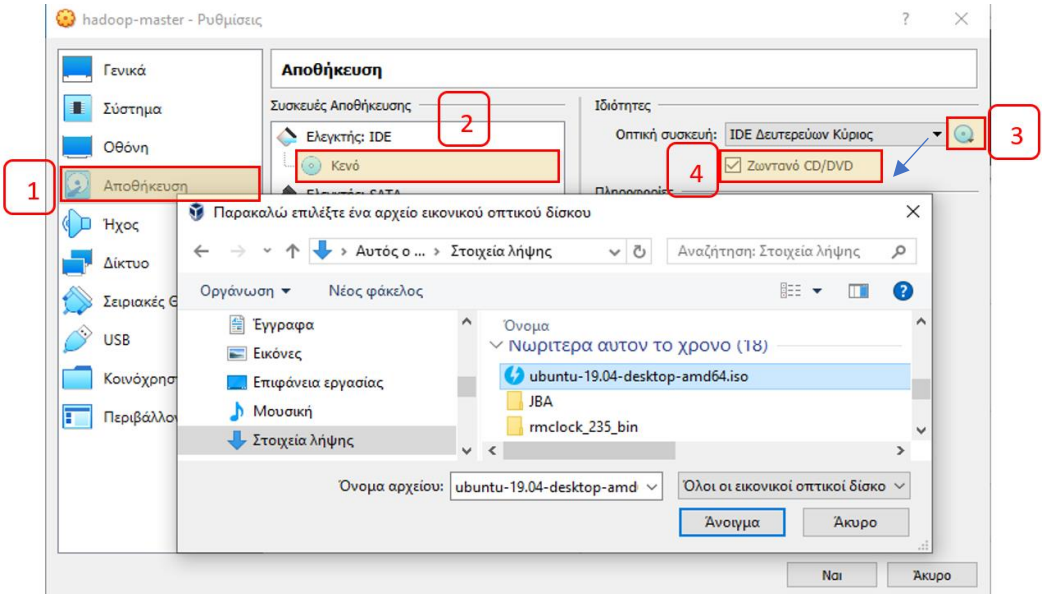
Σύροντας ακολούθως ο χρήστης το ρυθμιστικό δίπλα στο "Όριο εκτέλεσης" προς τα δεξιά μέχρι να εμφανιστεί η τιμή "100" επιτρέπει στο VirtualBox να χρησιμοποιεί όλους τους πόρους του επεξεργαστή του.

Προειδοποίηση: Επιτρέποντας ο χρήστης στο VirtualBox να χρησιμοποιεί όλους τους πόρους του επεξεργαστή του μπορεί να επιβραδύνει το λειτουργικό σύστημα του οικοδεσπότη υπολογιστή.



Σχήμα 6-69: Ρύθμιση μνήμης και λοιπών χαρακτηριστικών κάρτας γραφικών & αριθμού οθονών

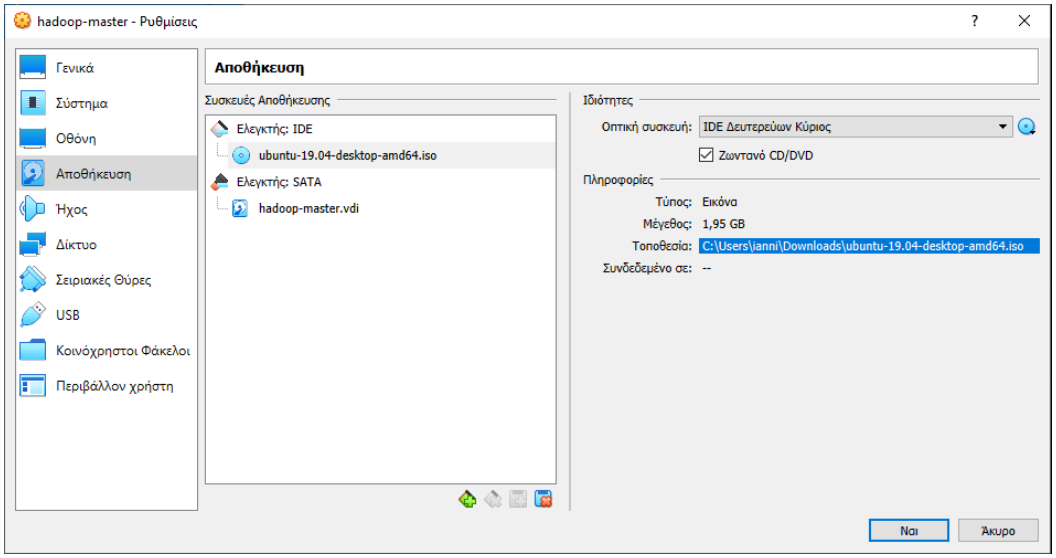
Η επιλογή Οθόνη, επιτρέπει στον χρήστη να επιλέξει το μέγεθος της μνήμης της κάρτας γραφικών, τον συντελεστή κλίμακας, τον ελεγκτή γραφικών καθώς και αν θα χρησιμοποιήσει κάποιο είδος επιτάχυνσης (2Δ ή 3Δ).



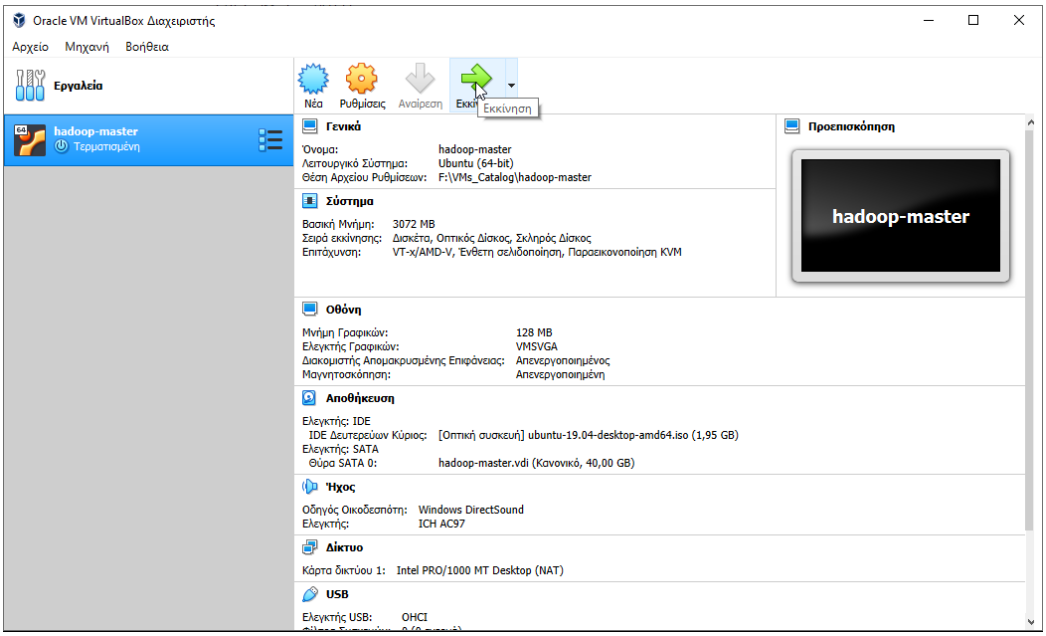
Σχήμα 6-70: Βήματα για την επιλογή αρχείου Εικονικού Οπτικού δίσκου (ΕΟΔ)

Θα πρέπει να κάνει κλικ στην επιλογή 'Αποθήκευση' (1) και ακολούθως στην επιλογή ‘Κενό’ (2) κάτω από τον Ελεγκτή: IDE (βλέπε Σχήμα 6-70) και κατόπιν στο εικονίδιο "CD / DVD"

(3) στη δεξιά πλευρά του παραθύρου και ακολούθως από την επιλογή “Επιλέξτε αρχείο εικονικού οπτικού δίσκου ” του πτυσσόμενο μενού να επιλέξει το αρχείο οπτικού εικονικού δίσκου (το αρχείο ISO ubuntu) που θα φορτωθεί στην οπτική συσκευή. Θα πρέπει επίσης να επιλέξει την επιλογή “Ζωντανό CD/DVD” (4) όπως φαίνεται στην παραπάνω εικόνα.



Σχήμα 6-71: Το παράθυρο Αποθήκευση (στις ρυθμίσεις της ΕΜ) μετά την ρύθμιση του αρχείου Εικονικού Οπτικού δίσκου (ΕΟΔ)



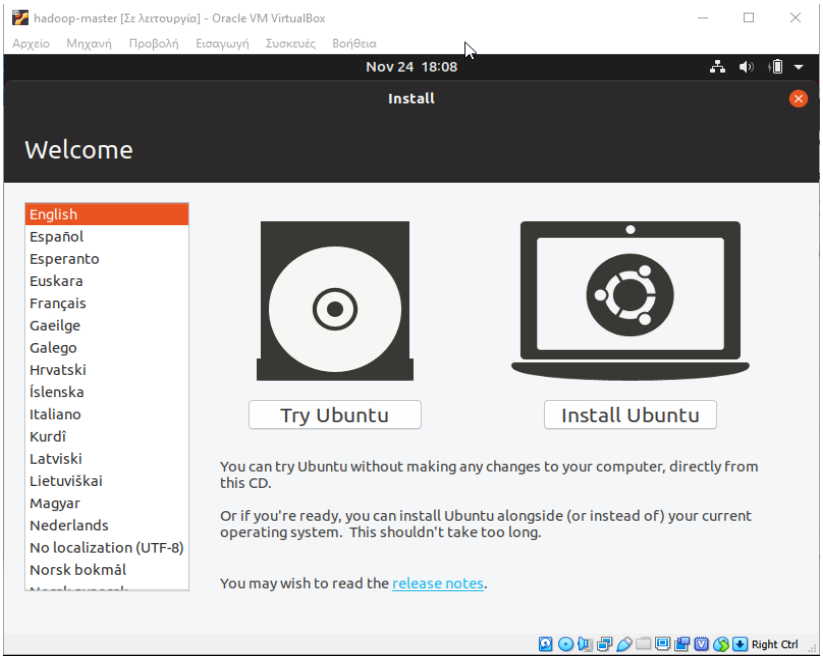
Σχήμα 6-72: Εκκίνηση ΕΜ

Το παράθυρο Αποθήκευση (στις ρυθμίσεις της ΕΜ) μετά την ρύθμιση του αρχείου Εικονικού Οπτικού δίσκου (ΕΟΔ) θα πρέπει να φαίνεται όπως στο Σχήμα 6-71.

Επιστρέφοντας στην οθόνη “Oracle VM VirtualBox Διαχειριστής”, ο χρήστης μπορεί να εκκινήσει την νέα ΕΜ κάνοντας κλικ στη ΕΜ Ubuntu και πατώντας το πλήκτρο 'Εκκίνηση' όπως φαίνεται στο Σχήμα 6-72.

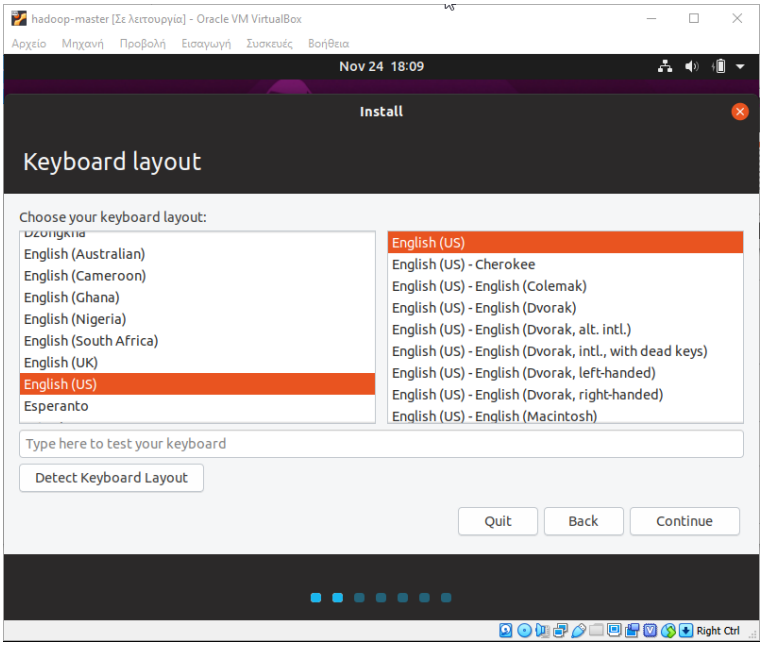
Εγκατάσταση του Ubuntu 19.04

Μόλις το σύστημα εκκινήσει, θα υποδεχτεί τον χρήστη μια οθόνη υποδοχής παρόμοια με την παρακάτω. Σε αυτή την οθόνη του παρέχονται οι επιλογές "Δοκιμάστε το Ubuntu" (Try Ubuntu) ή "Εγκαταστήστε το Ubuntu" (Install Ubuntu). Εάν επιλέξει την επιλογή "Δοκιμάστε το Ubuntu", μπορεί να δοκιμάσει το Ubuntu 19.04 χωρίς να εγκαταστήσει τίποτα στο σκληρό δίσκο του συστήματός του. Για να εγκαταστήσει το Ubuntu Linux στον σκληρό του δίσκο, θα πρέπει να επιλέξει την επιλογή " Εγκαταστήστε το Ubuntu" (Install Ubuntu). Στην αριστερή πλευρά της οθόνης δίνεται στον χρήστη η δυνατότητα επιλογής της γλώσσας εγκατάστασης. Παρέχεται επίσης ένας σύνδεσμος που του επιτρέπει να διαβάσει τις επίσημες σημειώσεις έκδοσης για την παρούσα έκδοση του Ubuntu Linux.

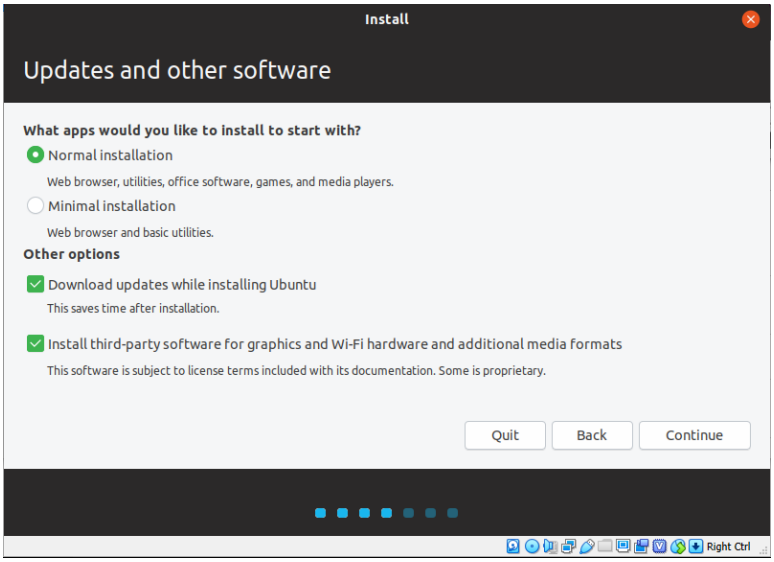


Σχήμα 6-73: Οθόνη υποδοχής (Welcome Screen)

Στην οθόνη που φαίνεται στο Σχήμα 6-74, ο χρήστης επιλέγει τη διάταξη του πληκτρολογίου του. Ο χρήστης μπορεί επίσης να δοκιμάσει τις ρυθμίσεις του πληκτρολογίου του πληκτρολογώντας στην παρεχόμενη περιοχή κειμένου. Αφού ολοκληρώσει την επιλογή του, κάνει κλικ στο κουμπί "Συνέχεια" (Continue).

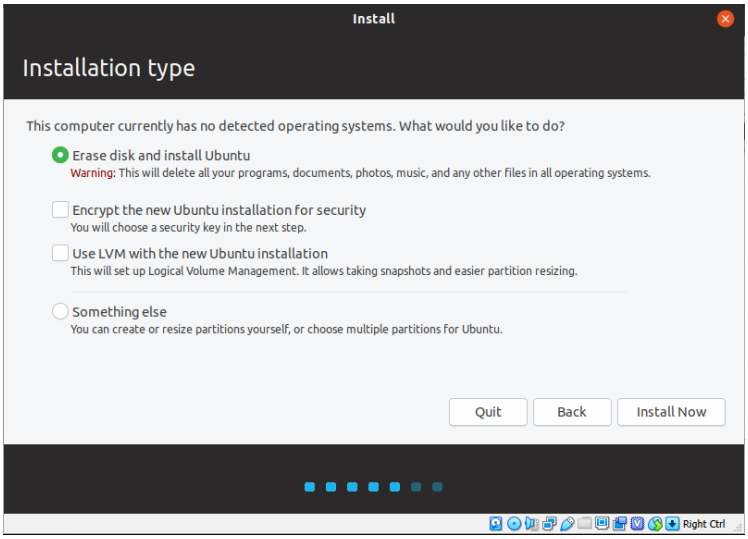


Σχήμα 6-74: Διάταξη πληκτρολογίου (Keyboard layout)



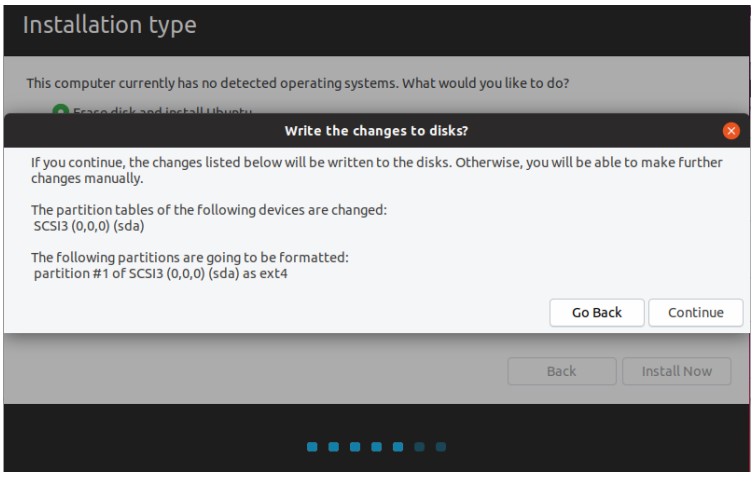
Σχήμα 6-75: Ενημερώσεις και άλλο λογισμικό (Updates and other Software)

Ο χρήστης μπορεί να επιλέξει (Βλέπε Σχήμα 6-75) είτε “Κανονική εγκατάσταση” (Normal Installation) για να εγκαταστήσει όλα τα πρότυπα πακέτα και το λογισμικό που συνοδεύει αυτή την έκδοση του Ubuntu είτε “Ελάχιστη εγκατάσταση” (Minimal Installation) για την εγκατάσταση μόνο των βασικών πακέτων. Μπορεί επίσης να επιλέξει τη λήψη ενημερώσεων (Download updates …) κατά την εγκατάσταση του Ubuntu και την εγκατάσταση λογισμικού τρίτου κατασκευαστή (Install third-party software …) για γραφικά, wi-fi υλικό και πρόσθετους μορφότυπους πολυμέσων. Αφού ολοκληρώσει τις επιλογές του σύμφωνα με την παραπάνω εικόνα, κάνει κλικ στο κουμπί "Συνέχεια".



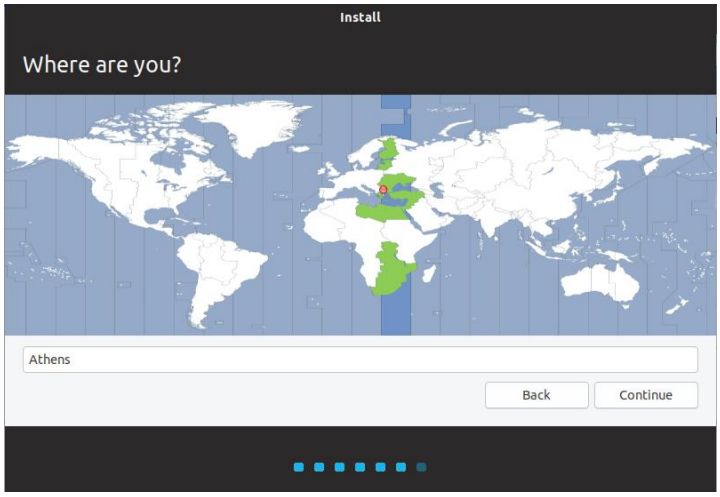
Σχήμα 6-76: Τύπος εγκατάστασης (Installation type)

Στην οθόνη του Σχήματος 6-76 διατίθενται πολλές επιλογές εγκατάστασης για να επιλέξει ο χρήστης. Στη συγκεκριμένη περίπτωση για την εγκατάσταση του λειτουργικού Ubuntu σε ΕΜ ενδείκνυται η πρώτη επιλογή “Διαγραφή δίσκου και εγκατάσταση του Ubuntu ” (Erase disk and …). Μόλις κάνει την επιλογή αυτή, θα πρέπει να κάνει κλικ στην επιλογή "Εγκατάσταση τώρα" (install Now) για να συνεχίσει με την εγκατάσταση.



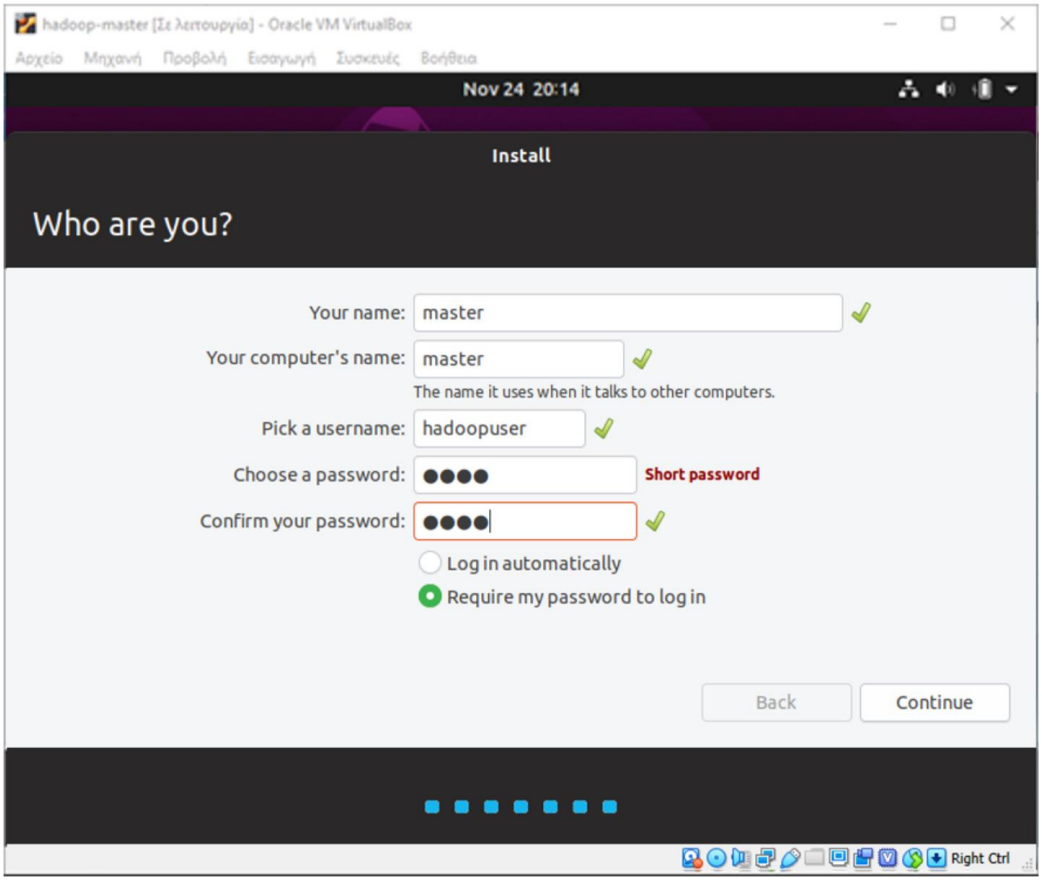
Σχήμα 6-77: Να εγγραφούν οι αλλαγές στους δίσκους; (Write the changes to disks?)

Για να μπορέσει ο χρήστης να ολοκληρώσει την εγκατάσταση (βλέπε Σχήμα 6-77), θα πρέπει να επιβεβαιώσει ότι είναι ικανοποιημένος με τις αλλαγές που πρόκειται να γίνουν στους δίσκους του. Εάν συμφωνεί με τις αλλαγές κάνει κλικ στο κουμπί "Συνέχεια" (Continue) διαφορετικά κάνει κλικ στην επιλογή "Επιστροφή" (Go Back) για να τροποποιήσει τις προηγούμενες επιλογές του. Στην προκειμένη περίπτωση θα πρέπει να κάνει κλικ στο κουμπί "Συνέχεια".



Σχήμα 6-78: Καθορισμός της γεωγραφικής θέσης του χρήστη - Που είσαι; (Where are you?)

Στην οθόνη που απεικονίζεται στο Σχήμα 6-78 πληκτρολογεί ο χρήστης την γεωγραφική του θέση. Αν είναι ενεργοποιημένη η σύνδεση "Internet", η τοποθεσία του χρήστη εντοπίζεται αυτόματα. Αυτή η ρύθμιση τοποθεσίας θα χρησιμοποιηθεί επίσης για τις ρυθμίσεις ζώνης ώρας του συστήματος του χρήστη. Ακολούθως κάνει κλικ στο κουμπί “Συνέχεια”.

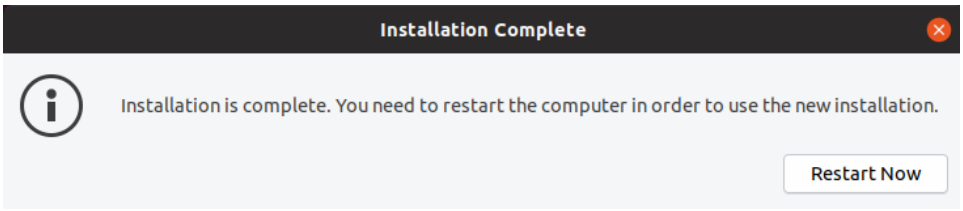


Σχήμα 6-79: Ποιος είσαι; (Who are you?)

Η οθόνη του Σχήματος 6-79, απαιτεί από τον χρήστη την παροχή:

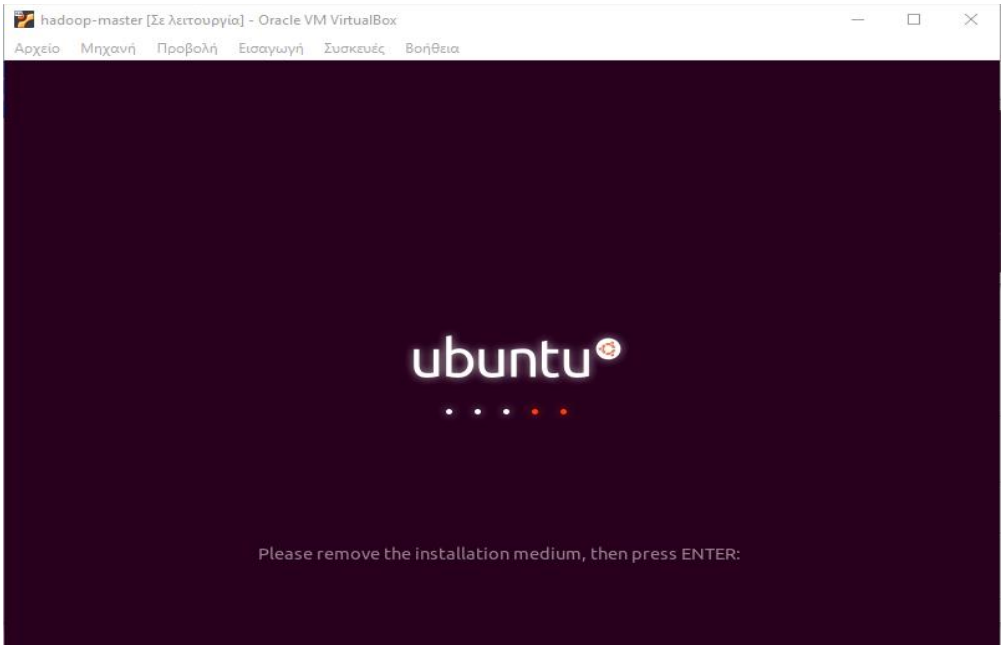
- του ονόματος (your name) που θα εμφανίζεται στη οθόνη σύνδεσης (login screen)
- ενός ονόματος (computer’s name/ hostname) για τον εντοπισμό του υπολογιστή στο δίκτυο που είναι διασυνδεδεμένος
- ενός ονόματος χρήστη (username) και ενός κωδικού πρόσβασης (password).

Συνιστάται ο χρήστης να επιλέξει για πρόσθετη ασφάλεια την επιλογή "Να απαιτείται κωδικός πρόσβασης για να συνδεθώ" (Require my password to login - αυτή είναι η προεπιλεγμένη επιλογή).



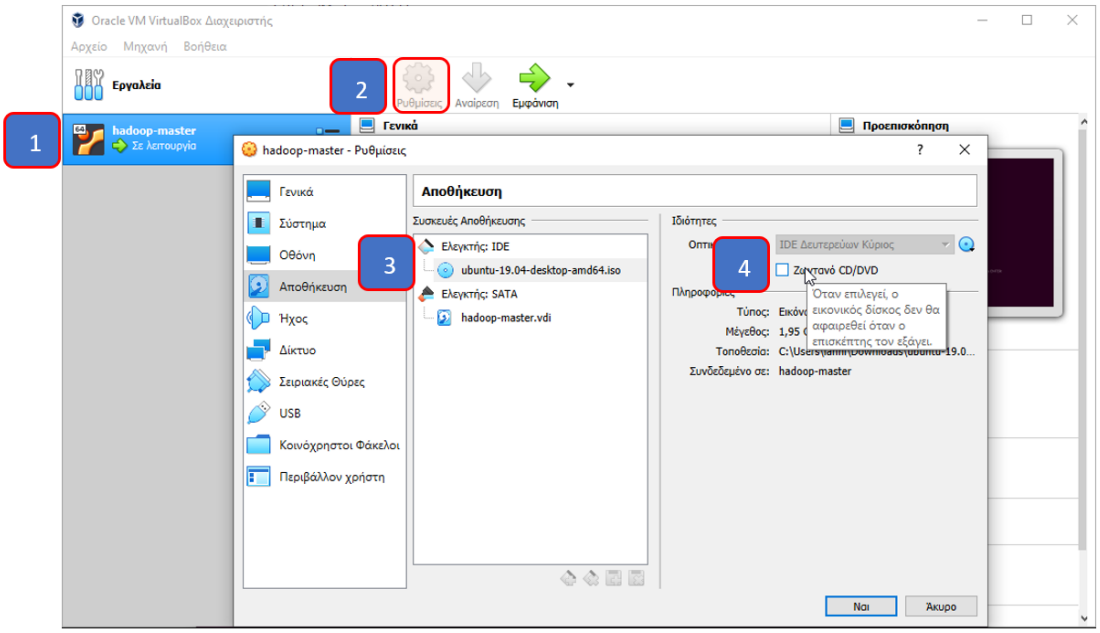
Σχήμα 6-80: Μήνυμα που πληροφορεί τον χρήστη για την ολοκλήρωση της εγκατάστασης και που τον προτρέπει να κάνει επανεκκίνηση της ΕΜ.

Όταν ολοκληρωθεί η εγκατάσταση τουUbuntu θα εμφανιστεί στον χρήστη η οθόνη του Σχήματος 6-80 που θα τον προτρέπει να κάνει επανεκκίνηση του υπολογιστή του / της ΕΜ του για να χρησιμοποιήσει τη νέα του εγκατάσταση. Αφού κάνει κλικ στην επιλογή "Επανεκκίνηση τώρα" (Restart now) θα εμφανιστεί η παρακάτω οθόνη που τον προτρέπει να αφαιρέσει το μέσο εγκατάστασης.



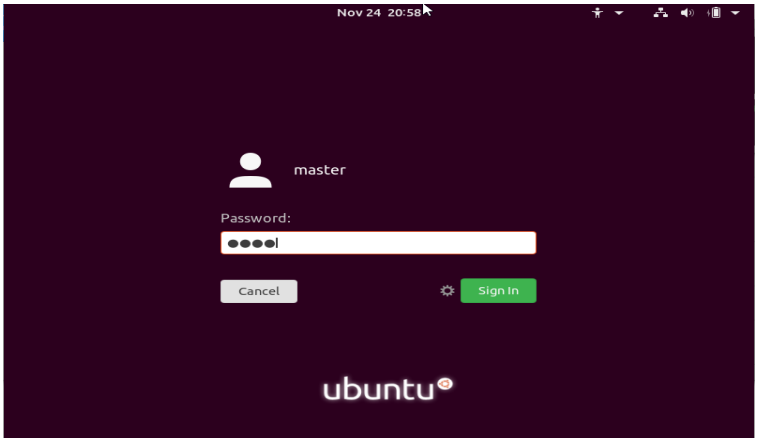
Σχήμα 6-81: Οθόνη μετά την εγκατάσταση του ΛΣ που προτρέπει τον χρήστη να αφαιρέσει το μέσο εγκατάστασης

Για να αφαιρέσει το μέσο εγκατάστασης από την οπτική συσκευή του θα πρέπει να μεταβεί στην οθόνη “Oracle VM VirtualBox Διαχειριστής” (βλέπε Σχήμα 6-82) και να επιλέξει : την EM Hadoop-master (1), Ρυθμίσεις (2) > στο παράθυρο “Ρυθμίσεις” που θα ανοίξει πρέπει να επιλέξει το αρχείο εγκατάστασης: ubuntu-19-04-desktop-amd64.iso (3) και στη οπτική συσκευή να αποεπιλέξει το πλαίσιο ελέγχου “Ζωντανό CD/DVD” (4). Ακολούθως αφού επιβεβαιώσει τις επιλογές του κάνοντας κλικ στο κουμπί “Ναι”, θα πρέπει να επιστρέψει στην οθόνη που εικονίζεται στο προηγούμενο Σχήμα και να πατήσει “Enter” για να χρησιμοποιήσει τη νέα του εγκατάσταση.



Σχήμα 6-82: Αφαίρεση του μέσου εγκατάστασης από την εικονική οπτική συσκευή

Μόλις ξεκινήσει το σύστημα του χρήστη, θα δει μια οθόνη σύνδεσης παρόμοια με αυτήν που φαίνεται Σχήμα 6-83. Σε αυτή την οθόνη πρέπει να εισαγάγει τον κωδικό πρόσβασης χρήστη που δημιούργησε σε προηγούμενο βήμα.

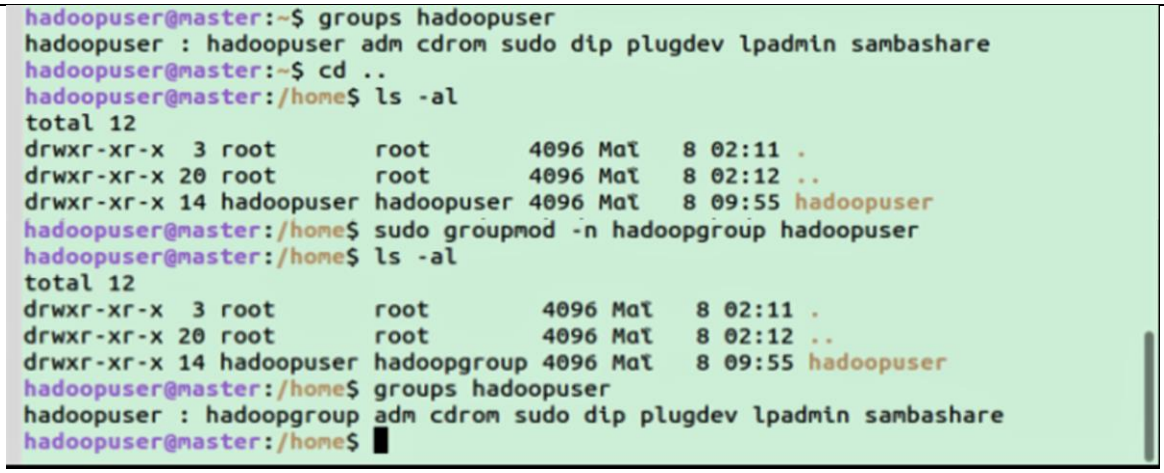


Σχήμα 6-83: Οθόνη σύνδεσης (Login screen)

Αλλαγή του ονόματος της κύριας ομάδας (primary groupname) του χρήστη με όνομα hadoopuser

Σύμφωνα με τον Πίνακα 6-3 θα πρέπει να αλλάξει ο χρήστης το όνομα της κύριας ομάδας του χρήστη με όνομα χρήστη (username) hadoopuser (βλέπε Σχήμα 6-84). Για τον λόγο αυτό πληκτρολογεί στην γραμμή εντολών:

```
groups hadoopuser
cd ..
ls -al
sudo groupmod -n hadoopgroop hadoopuser
ls -al
groups hadoopuser
```

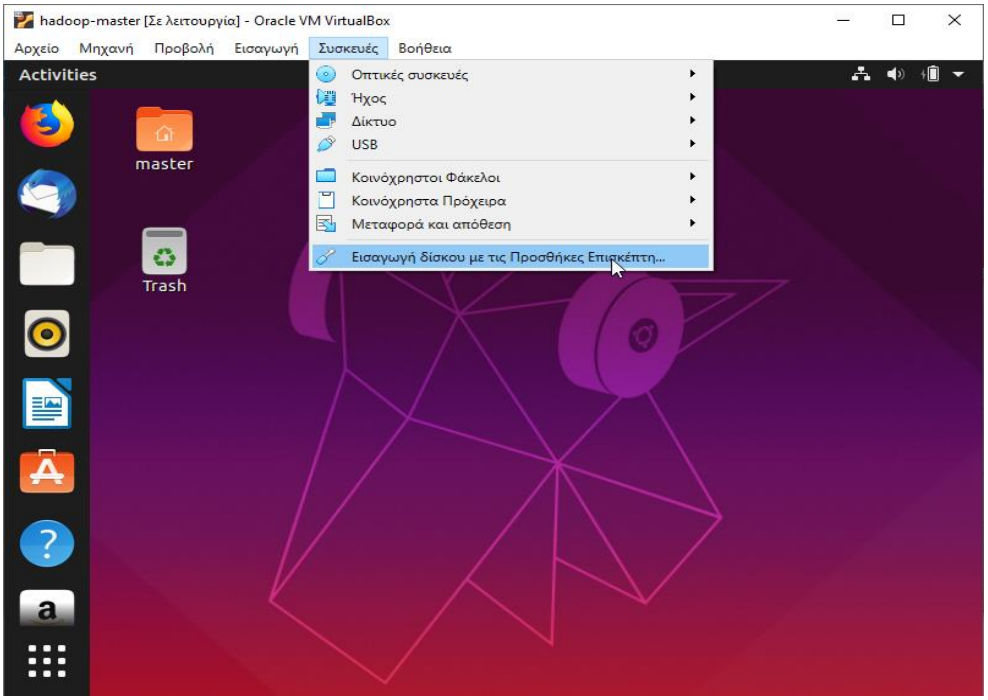


Σχήμα 6-84: Αλλαγή του ονόματος της κύριας ομάδας (primary groupname) του χρήστη με όνομα χρήστη (username) hadoopuser

Εγκατάσταση των προσθηκών επισκέπτη Virtualbox

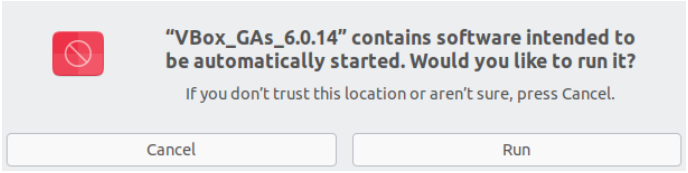
Ακολούθως θα πρέπει ο χρήστης να εγκαταστήσει τις προσθήκες επισκέπτη Virtualbox (guest additions) που επιτρέπουν την καλύτερη ολοκλήρωση μεταξύ των λειτουργικών συστημάτων του οικοδεσπότη και του επισκέπτη. Μετά την εγκατάσταση των προσθηκών επισκέπτη παρέχονται στον χρήστη νέες λειτουργίες όπως η απρόσκοπτη αντιγραφή / επικόλληση μέσω κοινόχρηστου πρόχειρου, η λειτουργία μεταφορά και απόθεση (drag and drop), η αυτό-ρυθμιζόμενη οθόνη επισκέπτη (auto-resize Guest Display) , η κατάσταση πλήρους οθόνης (full screen mode) της οθόνης του επισκέπτη, κλπ.

Για την εγκατάσταση των προσθηκών επισκέπτη θα πρέπει να μεταβεί ο χρήστης στο μενού “Συσκευές” στη γραμμή μενού του VirtualBox (βλέπε Σχήμα 6-85) και στη συνέχεια να επιλέξει "Εισαγωγή δίσκου με τις Προσθήκες επισκέπτη..".



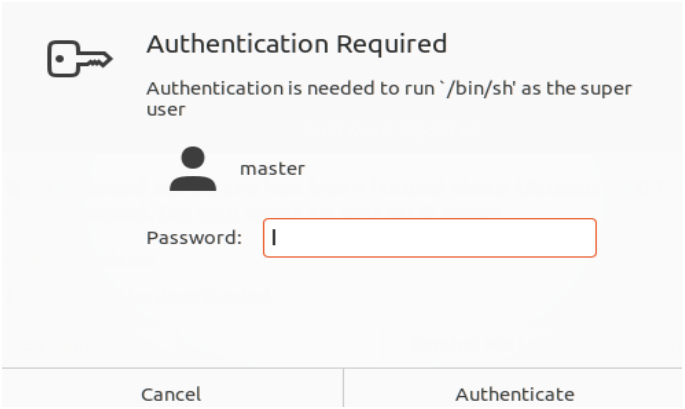
Σχήμα 6-85: Γραμμή μενού VirtualBox: Αναδυόμενο μενού "Συσκευές"> στοιχείο μενού "Προσθήκες Επισκέπτη.."

Θα εμφανιστεί το παράθυρο που φαίνεται στο Σχήμα 6-86 και θα προτρέπει τον χρήστη να πατήσει το πλήκτρο "Run", εάν εμπιστεύεται την πηγή προέλευσης, προκειμένου να εγκατασταθούν οι προσθήκες επισκέπτη.



Σχήμα 6-86: Ο χρήστης για να εγκαταστήσει τις προσθήκες επισκέπτη θα πρέπει να κάνει κλικ στο κουμπί "Run"

Μόλις πατήσει το πλήκτρο "Run" θα εμφανιστεί το παράθυρο αυθεντικοποίησης χρήστη (βλέπε Σχήμα 6-87) όπου θα πρέπει ο χρήστης να εισάγει τον κωδικό του για να προχωρήσει η εγκατάστασή τους.



Σχήμα 6-87: Παράθυρο Αυθεντικοποίησης Χρήστη

Αφού εγκατασταθούν οι προσθήκες επισκέπτη θα πρέπει ο χρήστης να επανεκκινήσει την εικονική μηχανή προκειμένου να φορτωθούν οι προσθήκες στο σύστημα του χρήστη.
Ο έλεγχος εάν οι προσθήκες επισκέπτη είναι εγκατεστημένες ή όχι γίνεται με την εντολή lsmod:

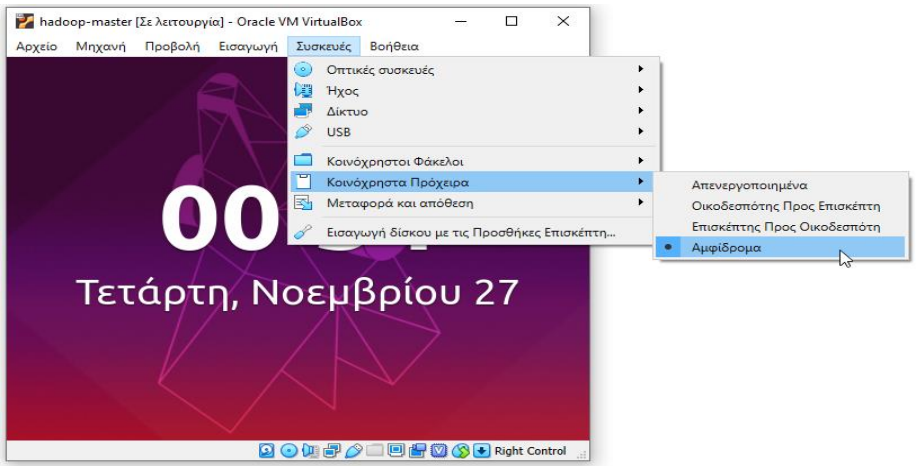
```
lsmod | grep -i vbox
```

Εάν οι προσθήκες vbox είναι εγκατεστημένες και λειτουργούν θα εμφανιστεί στην οθόνη του χρήστη κάποιο αποτέλεσμα σαν αυτό του παρακάτω σχήματος. Αν δεν εμφανιστεί τίποτα τότε πιθανώς οι προσθήκες επισκέπτη δεν εγκαταστάθηκαν επιτυχώς.

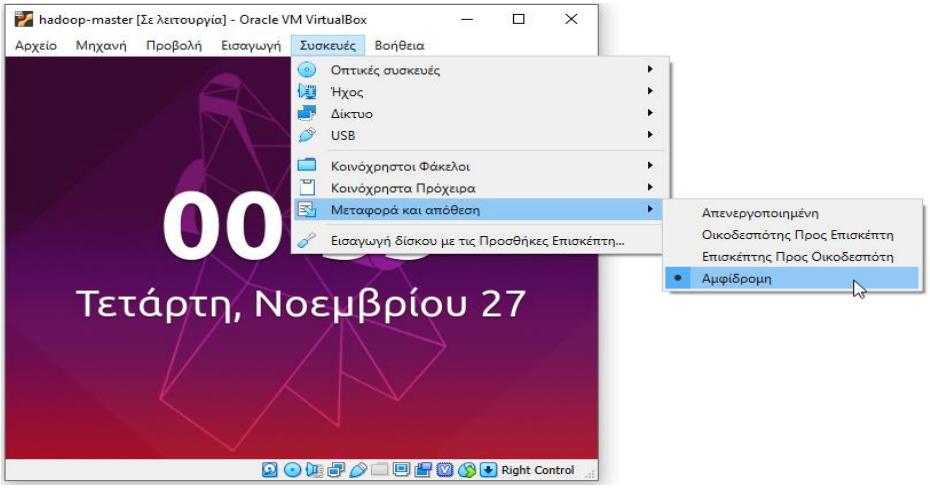
```
hadoopuser@master:~$ lsmod | grep -i vbox
vboxsf                 81920  1
vboxvideo              36864  0
vboxguest             331776  6 vboxsf
ttm                   102400  2 vmwgfx,vboxvideo
drm_kms_helper        184320  2 vmwgfx,vboxvideo
drm                   487424  7 vmwgfx,drm_kms_helper,vboxvideo,ttm
hadoopuser@master:~$
```

Σχήμα 6-88: Έλεγχος για το αν εγκαταστάθηκαν οι "Προσθήκες Επισκέπτη …"

Μετά την επιτυχή εγκατάσταση των προσθηκών επισκέπτη του VirtualBox θα πρέπει ο χρήστης να ενεργοποιήσει το "Κοινόχρηστο Πρόχειρο" (βλέπε Σχήμα 6-89α) και την "Μεταφορά και απόθεση" (βλέπε Σχήμα 6-89β) μέσω των στοιχείων μενού "Κοινόχρηστο Πρόχειρο" και "Μεταφορά και απόθεση" αντίστοιχα του μενού "Συσκευές" της ΕΜ.



(α)



(β)

Σχήμα 6-89: (α) : Ενεργοποίηση " Αμφίδρομων Κοινόχρηστων Πρόχειρων" μεταξύ οικοδεσπότη Η/Υ και ΕΜ
(β) Ενεργοποίηση " Αμφίδρομης Μεταφοράς και Απόθεσης" μεταξύ οικοδεσπότη Η/Υ και ΕΜ

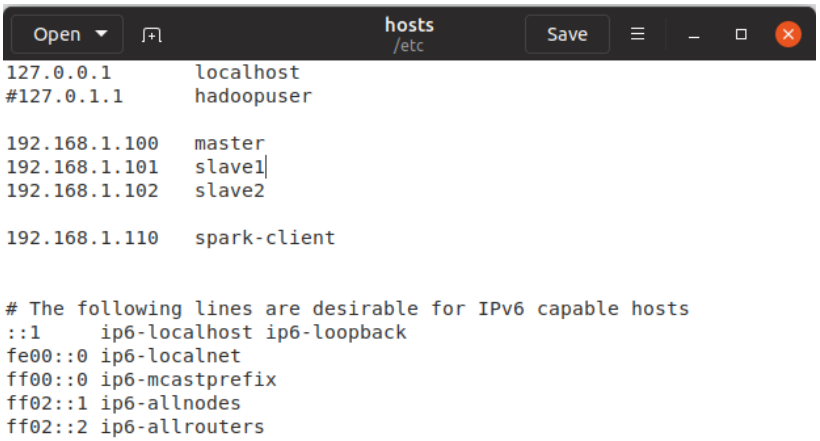
Διαμόρφωση αρχείου hosts

Στη συνέχεια θα πρέπει ο χρήστης να διαμορφώσει κατάλληλα το αρχείο *hosts* (χρησιμοποιείται από τα λειτουργικά συστήματα με υποστήριξη δικτύου για να μεταφράζουν τα ονόματα υπολογιστών (hostnames) σε διευθύνσεις IP). Θα πρέπει να προσθέσει τις IP διευθύνσεις και τα hostnames των ΕΜ της συστάδας που πρόκειται να δημιουργήσει (βλέπε Σχήμα 6-90).

```
sudo gedit /etc/hosts
```

Αρχείο: hosts

```
127.0.0.1       localhost
#127.0.1.1      hadoopuser

192.168.1.100   master
192.168.1.101   slave1
192.168.1.102   slave2
192.168.1.110   spark-client
```



Σχήμα 6-90: Διαμόρφωση Αρχείου hosts

Ρύθμιση της κάρτας Δικτύου στο VirtualBox και της στατικής διεύθυνσης IP στο Ubuntu

Στη συνέχεια θα πρέπει να ρυθμίσει στο VirtualBox την κάρτα δικτύου. Η σύνδεση θα πρέπει να εξασφαλίζει ότι οι ΕΜ:

- Θα μπορούν να επικοινωνούν μεταξύ τους
- Θα πρέπει να είναι προσβάσιμες από τον οικοδεσπότη (host) υπολογιστή
- Θα μπορούν να έχουν πρόσβαση στο διαδίκτυο
- Θα πρέπει να μπορούν να πάρουν στατικές διευθύνσεις IP

Σύμφωνα με τον Πίνακα 6.4 οι παραπάνω απαιτήσεις ικανοποιούνται με τους παρακάτω δυο τρόπους:

- Με γεφυρωμένη κάρτα (bridged adapter)

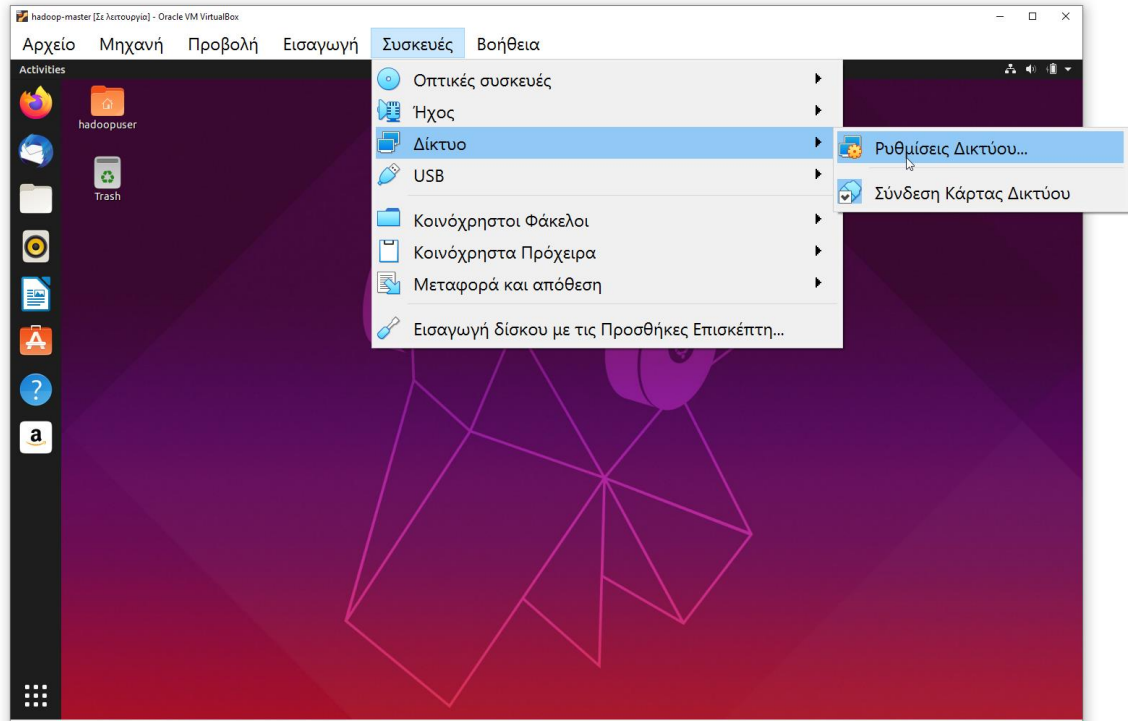
- Με συνδυασμό NAT και Host-Only Adapter

Πίνακας 6-4: Ρύθμιση Δικτύου στο VirtualBox

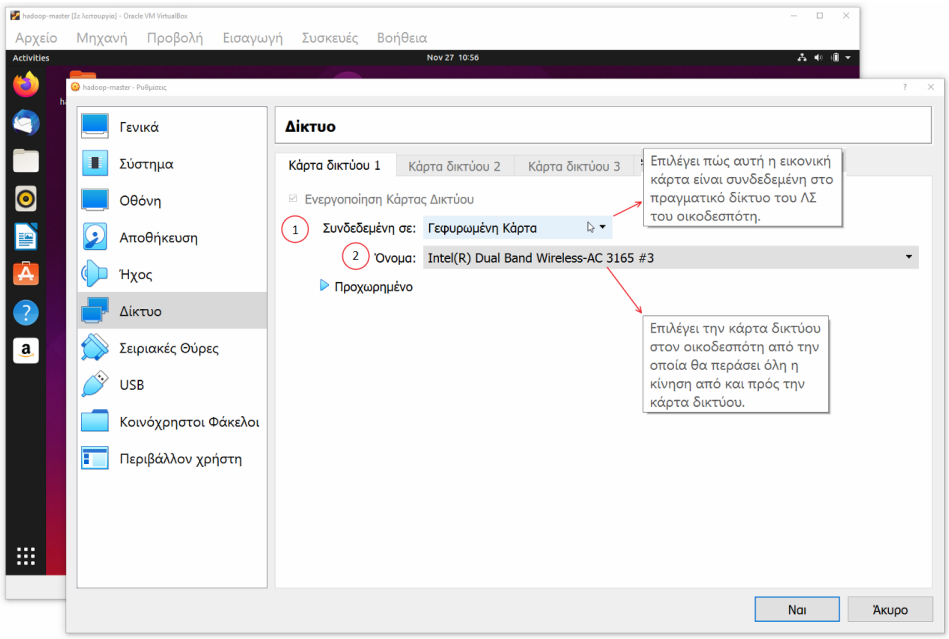
Network type	Access Guest -> other Guests	Access Host -> Guest	Access Guest -> external Network
Not attached	-	-	-
Network Address Translation (NAT)	-	-	✓
Network Address Translation Service	✓	-	✓
Bridged networking	✓	✓	✓
Internal networking	✓	-	-
Host-only networking	✓	✓	-

Πηγή: https://www.thomas-krenn.com/en/wiki/Network_Configuration_in_VirtualBox

Ο γράφων επέλεξε τον πρώτο τρόπο, η υλοποίηση του οποίου παρουσιάζεται στα παρακάτω σχήματα.

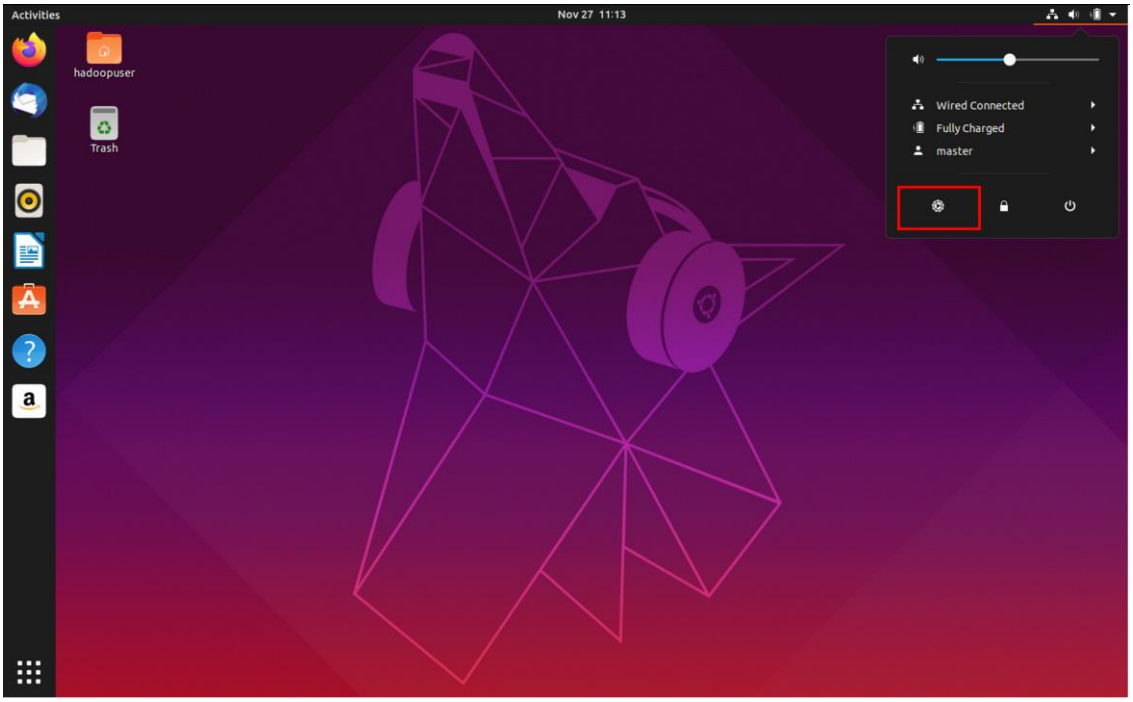


Σχήμα 6-91: "Ρυθμίσεις Δικτύου…" ΕΜ στο VirtualBox



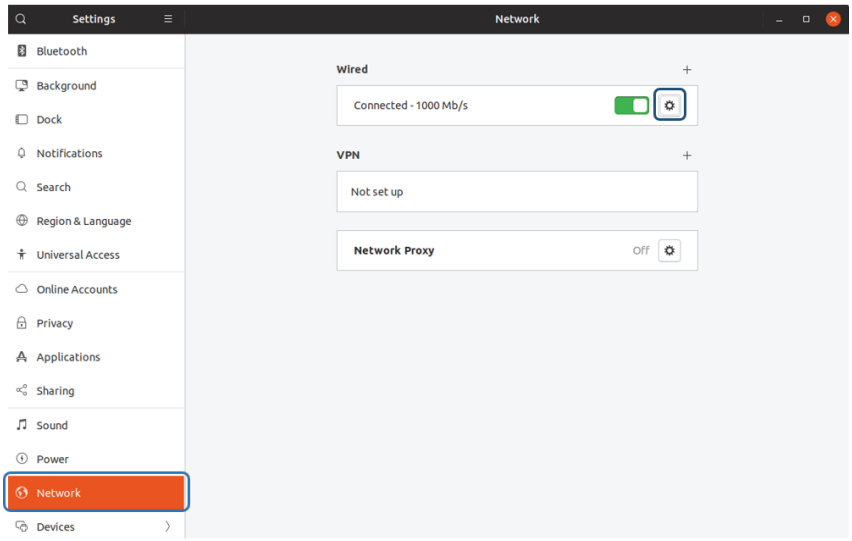
Σχήμα 6-92: Επιλογή κάρτας δικτύου ΕΜ

Ακολούθως θα πρέπει να καθορίσει την στατική IP διεύθυνση της ΕΜ σύμφωνα με τον Πίνακα 6-2. Για το λόγο αυτό θα πρέπει να κάνει κλικ σε οποιοδήποτε από τα εικονίδια στο πάνω δεξί μέρος της οθόνης (βλέπε Σχήμα 6-93) προκειμένου να ανοίξει το αναδυόμενο μενού, απ' όπου θα πρέπει να επιλέξει το γρανάζι για να εμφανιστεί το παράθυρο των ρυθμίσεων (settings).



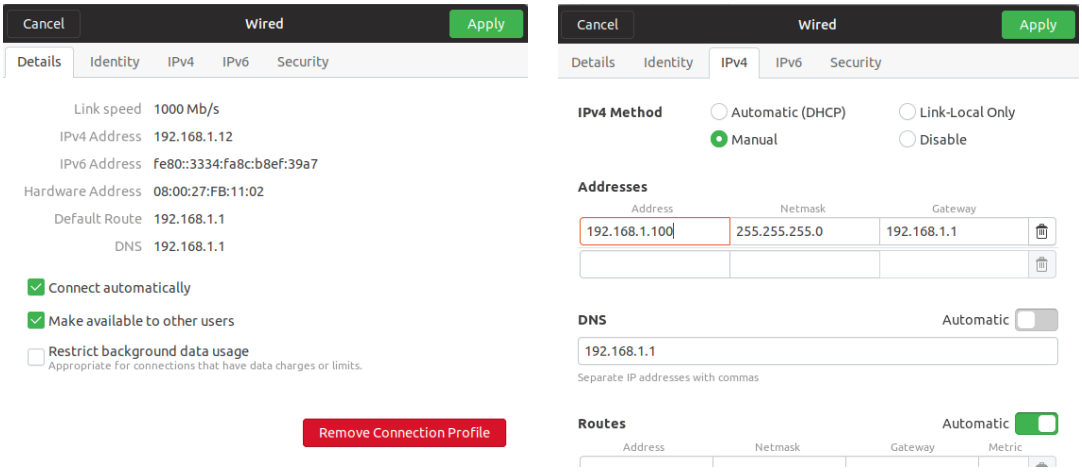
Σχήμα 6-93: Επιλογή "Ρυθμίσεις" του ΛΣ Ubuntu

Από την λίστα επιλογών στο αριστερό μέρος του παραθύρου ρυθμίσεων θα πρέπει να κάνει κλικ στην επιλογή “Δίκτυο” (Network) και στη συνέχεια στο γρανάζι στα δεξιά της ετικέτας “wired” όπως φαίνεται στο Σχήμα 6-94.



Σχήμα 6-94: Επιλογή κάρτας δικτύου στο ΛΣ Ubuntu

Στο νέο παράθυρο που θα ανοίξει (βλέπε Σχήμα 6-95α) θα πρέπει ο χρήστης να επιλέξει την καρτέλα με τίτλο “IPv4” και να πληκτρολογήσει την επιθυμητή στατική διεύθυνση IP, τη μάσκα δικτύου (netmask), την πύλη (gateway) και τις ρυθμίσεις DNS βλέπε Σχήμα 6-95β.



(α) (β)

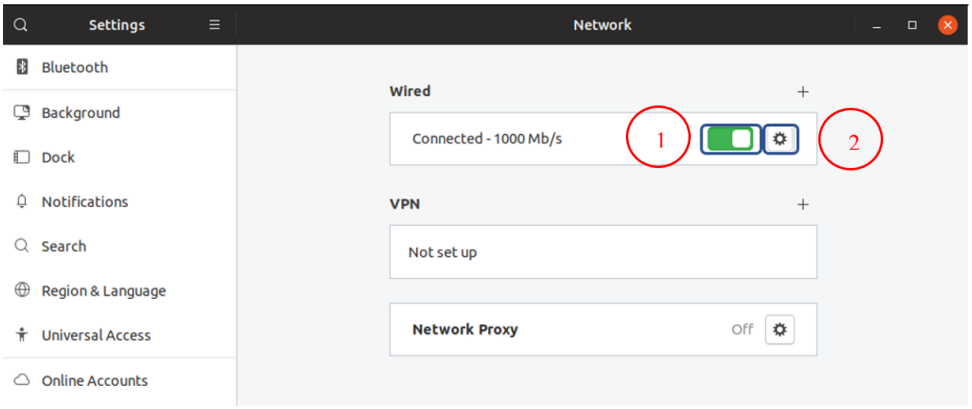
Σχήμα 6-95: Καθορισμός Στατικής Διεύθυνσης IP για την κάρτα δικτύου της ΕΜ

Ακολούθως θα πρέπει να εφαρμόσει τις αλλαγές κάνοντας κλικ στο κουμπί “Apply”. Προκειμένου να εφαρμοστούν οι αλλαγές θα πρέπει να απενεργοποιήσει και να ενεργοποιήσει το συρόμενο κουμπί στα δεξιά της ετικέτας “wired” (βλέπε Σχήμα 6-96) και πατώντας το

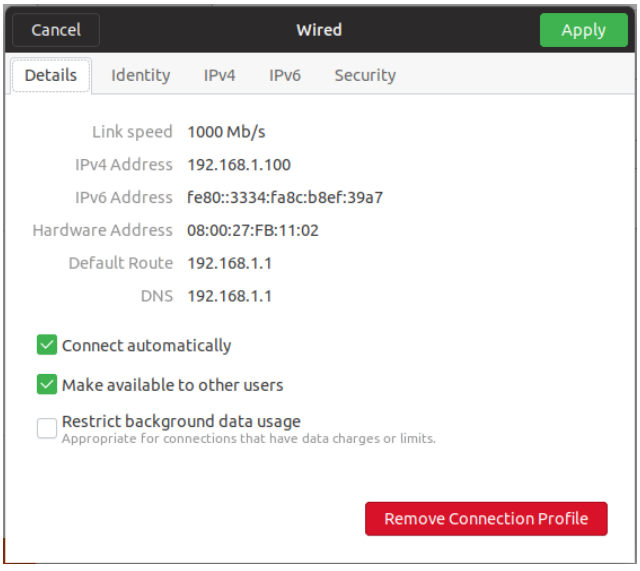
γρανάζι στα δεξιά του συρόμενου κουμπιού μπορεί να επιβεβαιώσει ότι οι αλλαγές υλοποιήθηκαν.

Μπορεί επίσης να επιβεβαιώσει τις αλλαγές στην IP διεύθυνση πληκτρολογώντας στη γραμμή εντολών (βλέπε Σχήμα 6-97β):

```
ip a
```



Σχήμα 6-96: (1) Εφαρμογή αλλαγών (κλείσιμο και άνοιγμα του συρόμενου κουμπιού) & (2) Έλεγχος της νέας στατικής διεύθυνσης IP της EM (κλικ στο γρανάζι)



(α)

```
hadoopuser@master:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:fb:11:02 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.100/24 brd 192.168.1.255 scope global noprefixroute enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::3334:fa8c:b8ef:39a7/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
hadoopuser@master:~$
```

(β)

Σχήμα 6-97: (α) & (β) Επιβεβαίωση των αλλαγών στη στατική διεύθυνση IP της EM

Εγκατάσταση της Java

Ως τώρα ο χρήστης έχει ολοκληρώσει τις παρακάτω εργασίες:

1. Δημιουργία μιας ΕΜ.
2. Εγκατάσταση Λειτουργικού Συστήματος (ΛΣ) ubuntu 19.04 στην ΕΜ.
3. Αλλαγή ονόματος κύριας ομάδας (primary groupname) του χρήστη με όνομα χρήστη (username) hadoopuser
4. Εγκατάσταση των προσθηκών επισκέπτη VirtualBox.
5. Τροποποίηση του αρχείου host.
6. Επιλογή κάρτας δικτύου στο VirtualBox.
7. Καθορισμός στατικής διεύθυνσης IP για την κάρτα δικτύου της ΕΜ.

Το αμέσως επόμενο βήμα που θα πρέπει να υλοποιήσει ο χρήστης είναι η εγκατάσταση της Java. Αν και αρκεί η εγκατάσταση της Java 8 SE Runtime Environment, προτείνεται η εγκατάσταση της Java 8 JDK (Java Development Kit) έτσι ώστε αν στο μέλλον επιθυμήσει να εγκαταστήσει και την Scala να γλυτώσει αυτό το επιπλέον βήμα (βλέπε Σχήμα 6-98).

Εγκατάσταση της Java 8 JDK

```
sudo apt-get install  openjdk-8-jdk
```

```
(base) hadoopuser@master:~$ sudo apt-get install openjdk-8-jdk
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-5.0.0-13 linux-headers-5.0.0-13-generic
  linux-image-5.0.0-13-generic linux-modules-5.0.0-13-generic
  linux-modules-extra-5.0.0-13-generic
Use 'sudo apt autoremove' to remove them.
```

Σχήμα 6-98: Εγκατάσταση Java (JDK) - openjdk-8-jdk

Έλεγχος εγκατεστημένης έκδοσης της Java

Ο έλεγχος της εγκατεστημένης έκδοσης της Java και του μεταγλωττιστή javac της Java που περιλαμβάνεται στο κιτ ανάπτυξης Java (JDK) γίνεται με τις παρακάτω εντολές:

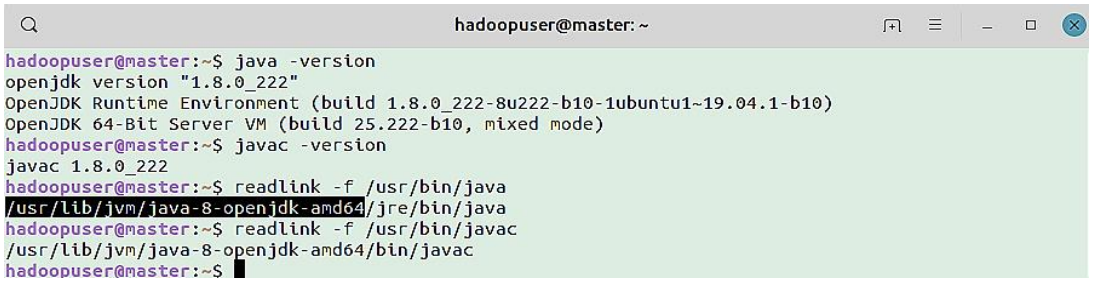
```
java -version
javac -version
```

Με τις παρακάτω εντολές μπορεί να βρει ο χρήστης σε ποιο κατάλογο είναι εγκατεστημένη (μονοπάτι εγκατάστασης) η java και ο μεταγλωττιστής της.

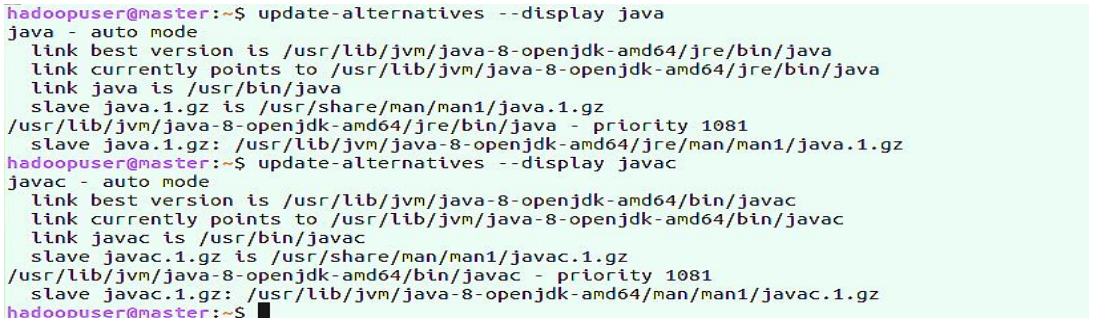
```
readlink -f /user/bin/java
readlink -f /user/bin/javac
```

ή

```
update-alternatives --display java
update-alternatives --display javac
```



Σχήμα 6-99: Εύρεση του μονοπατιού στο οποίο έχει εγκατασταθεί η Java με την εντολή: readlink -f /usr/bin/java



Σχήμα 6-100: Εύρεση του μονοπατιού στο οποίο έχει εγκατασταθεί η Java με την εντολή: update-alternatives --display java

Το συγκεκριμένο μονοπάτι (διαδρομή - */usr/lib/jvm/java-8-openjdk-amd64*) είναι γνωστό ως μεταβλητή περιβάλλοντος JAVA_HOME. Θα πρέπει ο χρήστης να ενημερώσει το αρχείο */etc/profile* με το JAVA_HOME, για το λόγο αυτό θα πρέπει να το ανοίξει με τον επεξεργαστή κειμένου gedit:

```
sudo gedit /etc/profile
```

και να πληκτρολογήσει:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

Σχήμα 6-101: Ενημέρωση του αρχείου /etc/profile με την μεταβλητή περιβάλλοντος JAVA_HOME

Στη συνέχεια θα πρέπει ο χρήστης να ενημερώσει και το αρχείο *~/.bashrc* αφού πρώτα το ανοίξει με τον επεξεργαστή κειμένου gedit:

```
gedit ~/.bashrc
```

Στο αρχείο *.bashrc* σε όλους τους κόμβους και στον spark-client θα πρέπει να προστεθούν οι παρακάτω γραμμές:

```
# >>>>                    JAVA Περιβαλλοντικές Μεταβλητές ΑΡΧΗ                    >>>>
#-------------------------------------------------------------------------------
-
```

```
#    Ορισμός JAVA_HOME και ενημέρωση του PATH
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export PATH=$PATH:$JAVA_HOME/bin
# <<<<                    JAVA Περιβαλλοντικές Μεταβλητές ΤΕΛΟΣ                <<<<
```

Λήψη και Εγκατάσταση του Anaconda (για python)

Αναλυτικές πληροφορίες για το Anaconda και την εγκατάστασή του θα βρεί ο αναγνώστης στην ενότητα "Εγκατάσταση του PySpark στο Ubuntu 19.04 σε τοπική λειτουργία (local mode)". Αφού μεταβεί ο χρήστης στην ιστοσελίδα λήψεων του επίσημου ιστότοπου του Anaconda και αντιγράψει τη διεύθυνση μεταφόρτωσης για την 64-bit έκδοση του Anaconda για Linux Λειτουργικό Σύστημα, ανοίγει ένα τερματικό και πληκτρολογεί (βλέπε Σχήμα 6-102):

```
cd ~/Downloads
wget https://repo.continuum.io/archive/Anaconda3-2019.07-Linux-x86_64.sh
```

Σχήμα 6-102: Μεταφόρτωση του Anaconda3

Στη συνέχεια αφού ολοκληρωθεί η μεταφόρτωση του αρχείου θα πρέπει να εκτελέσει το script εγκατάστασης του Anaconda (βλέπε Σχήμα 6-103α):

```
sudo bash Anaconda3-2019.07-Linux-x86_64.sh
```

Θα πρέπει ο χρήστης αρχικά να συμφωνήσει με τους όρους της εγκατάστασης πληκτρολογώντας *yes* και όταν του ζητηθεί η διαδρομή (μονοπάτι) της εγκατάστασης να ορίσει την διαδρομή: */opt/anaconda* (βλέπε Σχήμα 6-103β):
Μόλις ολοκληρωθεί η εγκατάσταση θα λάβει την έξοδο που φαίνεται στο Σχήμα 6-103γ. Στο ερώτημα που εμφανίζεται για το αν επιθυμεί την αρχικοποίηση του Anaconda3 συνιστάται να πληκτρολογήσει *yes* για να μπορεί να χρησιμοποιήσει την εντολή *conda* μέσα από το τερματικό (κονσόλα / γραμμή εντολών).

```
(base) hadoopuser@master:~/Downloads$ sudo bash Anaconda3-2019.07-Linux-x86_64.sh
[sudo] password for hadoopuser:

Welcome to Anaconda3 2019.07

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>>
```

(α)

```
Do you accept the license terms? [yes|no]
[no] >>>
Please answer 'yes' or 'no':'
>>> yes

Anaconda3 will now be installed into this location:
/home/hadoopuser/anaconda3

  - Press ENTER to confirm the location
  - Press CTRL-C to abort the installation
  - Or specify a different location below

[/home/hadoopuser/anaconda3] >>> /opt/anaconda
```

(β)

```
installation finished.
Do you wish the installer to initialize Anaconda3
by running conda init? [yes|no]
[no] >>> yes
```

(γ)

Σχήμα 6-103: Εγκατάσταση Anaconda3

Στη συνέχεια αφού κλείσει το τερματικό και ανοίξει ένα νέο θα πρέπει να εκτελέσει μια από τις εντολές *conda --version*, *anaconda --version*, *python --version* για να βεβαιωθεί ότι εγκαταστάθηκε σωστά το Anaconda. Παρατηρώντας κανείς το παρακάτω σχήμα θα διαπιστώσει ότι η γραμμή εντολών του χρήστη έχει πρόθεμα με το όνομα του ενεργού περιβάλλοντος conda (base – το βασικό περιβάλλον conda). Για περισσότερες πληροφορίες σχετικά με τα περιβάλλοντα conda θα πρέπει να ανατρέξει ο χρήστης στην ενότητα "Διαχείριση έργων".

```
(base) hadoopuser@master:~/Downloads$ conda --version
conda 4.7.10
(base) hadoopuser@master:~/Downloads$ anaconda --version
anaconda Command line client (version 1.7.2)
(base) hadoopuser@master:~/Downloads$ python --version
Python 3.7.3
(base) hadoopuser@master:~/Downloads$
```

Σχήμα 6-104: Έλεγχος εγκατάστασης Anaconda

Αφού κλείσει και ανοίξει εκ νέου το τερματικό, θα πρέπει να αλλάξει τον ιδιοκτήτη του καταλόγου (βλέπε Σχήμα 6-105):

```
sudo chown -R hadoopuser:hadoopgroup /opt/anaconda
```

```
(base) hadoopuser@master:~$ ls -al /opt
total 16
drwxr-xr-x  4 root root 4096 Νοε  28 18:03 .
drwxr-xr-x 20 root root 4096 Δεκ   2 19:56 ..
drwxr-xr-x 26 root root 4096 Νοε  28 18:09 anaconda
drwxr-xr-x  9 root root 4096 Νοε  24 21:04 VBoxGuestAdditions-6.0.14
(base) hadoopuser@master:~$ sudo chown -R hadoopuser:hadoopgroup /opt/anaconda
[sudo] password for hadoopuser:

(base) hadoopuser@master:~$
(base) hadoopuser@master:~$ ls -al /opt
total 16
drwxr-xr-x  4 root       root        4096 Νοε  28 18:03 .
drwxr-xr-x 20 root       root        4096 Δεκ   2 19:56 ..
drwxr-xr-x 26 hadoopuser hadoopgroup 4096 Νοε  28 18:09 anaconda
drwxr-xr-x  9 root       root        4096 Νοε  24 21:04 VBoxGuestAdditions-6.0.14
(base) hadoopuser@master:~$
```

Σχήμα 6-105: Αλλαγή ιδιοκτήτη στον κατάλογο /opt/anaconda

Λήψη και εγκατάσταση του Hadoop / YARN

Αφού μεταβεί ο χρήστης στην ιστοσελίδα λήψεων του επίσημου ιστότοπου του Apache Hadoop θα πρέπει να πατήσει στον σύνδεσμο binary της έκδοσης 3.2.1 και να αντιγράψει ακολούθως τη διεύθυνση μεταφόρτωσης για την έκδοση 3.2.1 του Apache Hadoop. Κατόπιν ανοίγει ένα τερματικό και πληκτρολογεί (βλέπε Σχήμα 6-106):

```
cd ~/Downloads
wget          http://ftp.cc.uoc.gr/mirrors/apache/hadoop/common/hadoop-3.2.1/hadoop-3.2.1.tar.gz
```

```
(base) hadoopuser@master:~$ clear
(base) hadoopuser@master:~/Downloads$ wget http://ftp.cc.uoc.gr/mirrors/apache/hadoop/common/hadoop-3.2.1/hadoop-3.2.1.tar.gz
--2019-12-03 16:49:34--  http://ftp.cc.uoc.gr/mirrors/apache/hadoop/common/hadoop-3.2.1/hadoop-3.2.1.tar.gz
Resolving ftp.cc.uoc.gr (ftp.cc.uoc.gr)... 147.52.159.12, 2001:648:2c00:6c05::2
Connecting to ftp.cc.uoc.gr (ftp.cc.uoc.gr)|147.52.159.12|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 359196911 (343M) [application/octet-stream]
Saving to: 'hadoop-3.2.1.tar.gz'

hadoop-3.2.1.tar.gz            100%[=====================================================>] 342,56M   293KB/s    in 26m 59s

2019-12-03 17:16:33 (217 KB/s) - 'hadoop-3.2.1.tar.gz' saved [359196911/359196911]

(base) hadoopuser@master:~/Downloads$
```

Σχήμα 6-106: Λήψη του Apache Hadoop

Στη συνέχεια θα πρέπει με την εντολή *tar* να αποσυμπιέσει και εξαγάγει το αρχείο *hadoop-3.2.1.tar.gz* στον ομώνυμο κατάλογο *hadoop-3.2.1* (βλέπε Σχήμα 6-107):

```
tar xvzf hadoop-3.2.1.tar.gz
ls -al
```

```
(base) hadoopuser@master:~/Downloads$ ls -al
total 1633816
drwxr-xr-x  3 hadoopuser hadoopgroup      4096 Δεκ   3 18:08 .
drwxr-xr-x 17 hadoopuser hadoopgroup      4096 Δεκ   3 15:35 ..
-rw-r--r--  1 hadoopuser hadoopgroup 541906131 Ιουλ 25 17:59 Anaconda3-2019.07-Linux-x86_64.sh
-rw-r--r--  1 hadoopuser hadoopgroup 541906131 Ιουλ 25 17:59 Anaconda3-2019.07-Linux-x86_64.sh.1
drwxr-xr-x  9 hadoopuser hadoopgroup      4096 Σεπ  10 19:51 hadoop-3.2.1
-rw-r--r--  1 hadoopuser hadoopgroup 359196911 Σεπ  23 08:16 hadoop-3.2.1.tar.gz
-rw-r--r--  1 hadoopuser hadoopgroup 229988313 Μαΐ   1  2019 spark-2.4.3-bin-hadoop2.7.tgz
(base) hadoopuser@master:~/Downloads$
```

Σχήμα 6-107: Ο κατάλογος ~/Downloads μετά την αποσυμπίεση και εξαγωγή του αρχείου hadoop-3.2.1 με την εντολή tar

και ακολούθως να τον μεταφέρει με την εντολή *mv* στη θέση */usr/local/hadoop* (βλέπε Σχήμα 6-108):

```
sudo mv hadoop-3.2.1 /usr/local/hadoop
ls -al /usr/local/
```

```
(base) hadoopuser@master:~/Downloads$ sudo mv hadoop-3.2.1 /usr/local/hadoop
(base) hadoopuser@master:~/Downloads$ ls -al /usr/local/
total 44
drwxr-xr-x 11 root       root        4096 Δεκ   3 18:17 .
drwxr-xr-x 14 root       root        4096 Απρ  16  2019 ..
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 bin
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 etc
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 games
drwxr-xr-x  9 hadoopuser hadoopgroup 4096 Σεπ  10 19:51 hadoop
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 include
drwxr-xr-x  3 root       root        4096 Απρ  16  2019 lib
lrwxrwxrwx  1 root       root           9 Νοε  24 20:15 man -> share/man
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 sbin
drwxr-xr-x  5 root       root        4096 Απρ  16  2019 share
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 src
(base) hadoopuser@master:~/Downloads$
```

Σχήμα 6-108: Μεταφορά του καταλόγου ~/Downloads/hadoop-3.2.1 στον κατάλογο /usr/local/hadoop

Στη συνέχεια αφού ανοίξει το αρχείο *~/.bashrc*:

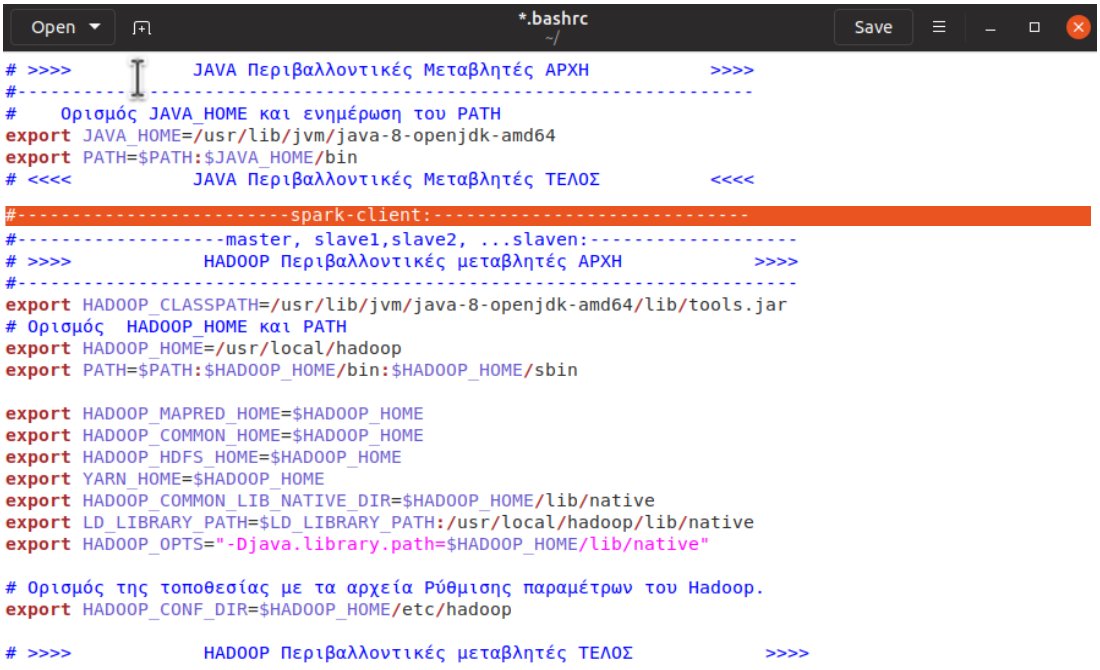
```
gedit ~/.bashrc
```

- Θα πρέπει να το ενημερώσει ως ακολούθως:

```
# >>>>                    HADOOP Περιβαλλοντικές μεταβλητές ΑΡΧΗ                 >>>>
#-------------------------------------------------------------------------------
export HADOOP_CLASSPATH=/usr/lib/jvm/java-8-openjdk-amd64/lib/tools.jar
# Ορισμός  HADOOP_HOME και PATH
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin

export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/hadoop/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"

# Ορισμός της τοποθεσίας με τα αρχεία Ρύθμισης παραμέτρων του Hadoop.
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
# >>>>                   HADOOP Περιβαλλοντικές μεταβλητές ΤΕΛΟΣ                 >>>>
```

Η ενημέρωση αυτή αφορά όλους τους κόμβους (master, slave1,slave2, ...slaven) [30] και τον spark-client (βλέπε Σχήμα 6-109):

[30] *hostnames*



Σχήμα 6-109: Το αρχείο ~/.bashrc στον κόμβο master μετά την εγκατάσταση του Apache Hadoop

Σύμφωνα με τον Πίνακα 6-1, τα αρχεία διαμόρφωσης (configuration files) που θα πρέπει να ενημερώσει ο χρήστης μετά την εγκατάσταση του Apache Hadoop είναι τα παρακάτω:

- /etc/profile.d/hadoop.sh

Και στο μονοπάτι */usr/local/hadoop/etc/hadoop/* :

- hadoop-env.sh
- core-site.xml
- hdfs-site.xml
- mapred-site.xml
- yarn-site.xml
- workers

hadoop.sh:

Είθισται να διαμορφώνεται το HADOOP_HOME ως μεταβλητή περιβάλλοντος συστήματος για το λόγο αυτό δημιουργεί αρχικά ο χρήστης, με το gedit, το αρχείο *hadoop.sh* μέσα στον κατάλογο */etc/profile.d*:

```
sudo gedit /etc/profile.d/hadoop.sh
```

και το ενημερώνει όπως παρακάτω:

```
export HADOOP_HOME=/usr/local/hadoop
```

Στον κατάλογο */etc/profile.d* αποθηκεύει κανείς script εκκίνησης όπου καθένα από αυτά αφορά συγκεκριμένη εφαρμογή και τα οποία εκτελούνται από το κέλυφος κατά την εκκίνηση του υπολογιστή του. Στη συνέχεια αφού το σώσει το μετατρέπει σε εκτελέσιμο αρχείο (βλέπε Σχήμα 6-110):

```
ls -al /etc/profile.d/hadoop.sh
```

```
sudo chmod +x /etc/profile.d/hadoop.sh
ls -al /etc/profile.d/hadoop.sh
```

```
(base) hadoopuser@master:~$ ls -al /etc/profile.d/hadoop.sh
-rw-r--r-- 1 root root 37 Δεκ   4 21:03 /etc/profile.d/hadoop.sh
(base) hadoopuser@master:~$ sudo chmod +x /etc/profile.d/hadoop.sh
(base) hadoopuser@master:~$ ls -al /etc/profile.d/hadoop.sh
-rwxr-xr-x 1 root root 37 Δεκ   4 21:03 /etc/profile.d/hadoop.sh
(base) hadoopuser@master:~$
```

Σχήμα 6-110: Μετατροπή του αρχείου hadoop.sh σε εκτελέσιμο

Ακολούθως ενημερώνει τα αρχεία που βρίσκονται στο μονοπάτι */usr/local/hadoop/etc/hadoop/*:

```
cd /usr/local/hadoop/etc/hadoop/
```

hadoop-env.sh:

```
gedit hadoop-env.sh
```

```
# Ορισμός του JAVA_HOME
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_OS_TYPE=${HADOOP_OS_TYPE:-$(uname -s)}
```

hdfs-site.xml:

```
gedit hdfs-site.xml
```

```
<configuration>

<!--καθορισμός του συντελεστή αναπαραγωγής (replication factor) -->
  <property>
    <name>dfs.replication</name>
    <value>2</value>
  </property>

<!-- Διαμόρφωση για τον NameNode -->
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
  </property>

<!--Διαμόρφωση για τον DataNode -->
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
  </property>

  <property>
    <name>dfs.permissions.superusergroup</name>
    <value>hadoopgroup</value>
    <description>Προσδιορίζει την ομάδα (group) UNIX της οποίας οι χρήστες θα
αντιμετωπίζονται ως υπερχρήστες από το HDFS.
    </description>
```

```
  </property>

</configuration>
```

Ο συντελεστής αναπαραγωγής (replication factor) υπαγορεύει τον αριθμό των αντιγράφων ενός μπλοκ που θα πρέπει να διατηρούνται στη συστάδα του χρήστη.

Ο συντελεστής αναπαραγωγής είναι εξορισμού 3 (το αρχικό μπλοκ και δύο αντίγραφά του) εδώ τροποποιήθηκε σε 2 γιατί η συστάδα θα έχει μόνο δύο slave κόμβους. Επομένως κάθε αρχείο που θα δημιουργεί ο χρήστης στο HDFS θα έχει συντελεστή αναπαραγωγής 2 και κάθε μπλοκ από το αρχείο θα αντιγράφεται στους 2 slave κόμβους της συστάδας του. Μπορεί ο χρήστης κατά τη δημιουργία ενός αρχείου να ορίσει και τον συντελεστή αναπαραγωγής για το συγκεκριμένο αρχείο σε περίπτωση που θέλει να είναι διαφορετικός από αυτόν που είχε ορίσει στο αρχείο *hdfs-site.xml*.

core-site.xml:

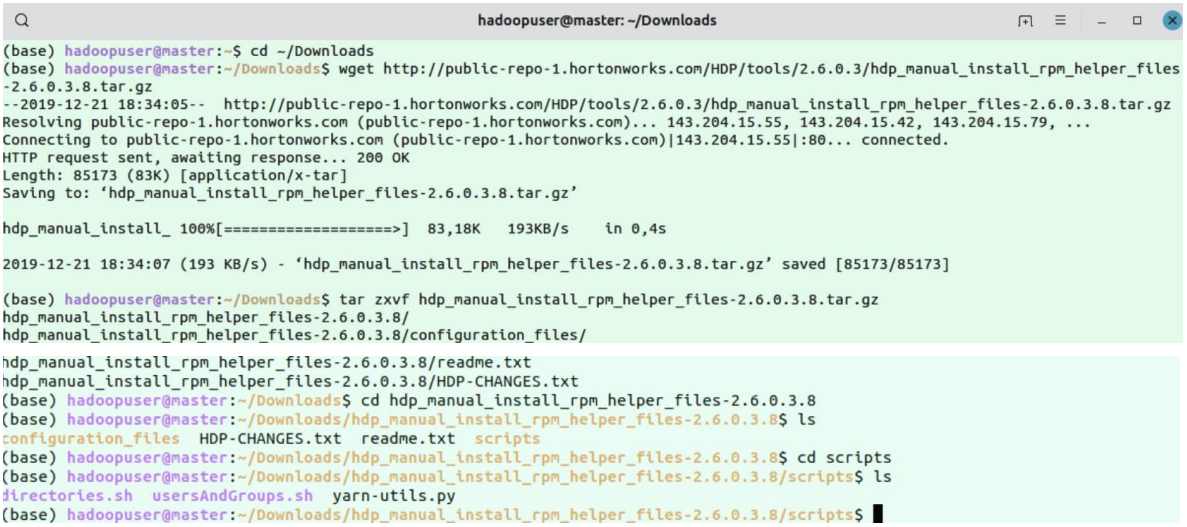
```
gedit core-site.xml
```

```
<configuration>

  <property>
    <name>hadoop.tmp.dir</name>
    <value>/usr/local/hadoop_store/tmp</value>
    <description>Ο γονικός κατάλογος για τους προσωρινούς  καταλόγους </description>
  </property>

  <property>
    <name>fs.defaultFS </name>
    <!-- NameNode URI hdfs://host:port/ -->
    <value>hdfs://master:9000</value>
    <description>Το fs.defaultFS – σε παλαιότερες εκδόσεις του Hadoop (σε εκδόσεις
προγενέστερες του YARN) είχε την ονομασία  fs.default.name. Είναι το εξορισμού
μονοπάτι που χρησιμοποιείται ως πρόθεμα (απόλυτο μονοπάτι βάσης – absolute base path)
από τον πελάτη Hadoop FS όταν δεν προσδιορίζεται το απόλυτο (πλήρες) μονοπάτι.
</description>
  </property>

</configuration>
```

Το YARN λαμβάνει υπόψη όλους τους διαθέσιμους υπολογιστικούς πόρους σε κάθε μηχάνημα της συστάδας. Οι διαμορφώσεις μνήμης στο YARN έγιναν σύμφωνα με το βοηθητικό script yarn-utils.py που παραπέμπει ο οδηγός εγκατάστασης “Hortonworks Data Platform (HDP): Command line Installation” [38] και αφού τροποποιήθηκε για να είναι συμβατό με την Python3. Στον οδηγό αυτό παρέχονται επίσης αναλυτικές πληροφορίες σχετικά με τον υπολογισμό με μη αυτόματο τρόπο των ρυθμίσεων μνήμης στα αρχεία YARN και MapReduce. Στο Σχήμα 6-113 δίδονται σχηματικά οι περιοχές μνήμης που αναφέρονται σε αυτό το script.

```
cd ~/Downloads
```

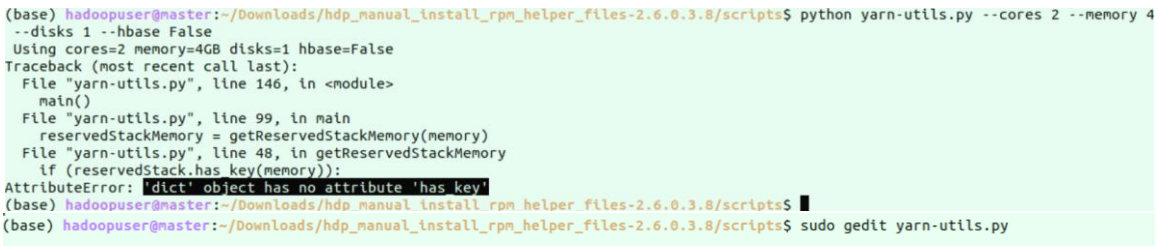
```
wget http://public-repo-1.hortonworks.com/HDP/tools/2.6.0.3/hdp_manual_install_rpm_helper_files-2.6.0.3.8.tar.gz
tar zxvf hdp_manual_install_rpm_helper_files-2.6.0.3.8.tar.gz
cd hdp_manual_install_rpm_helper_files-2.6.0.3.8
ls
cd scripts
ls
```



Σχήμα 6-111: Μεταφόρτωση του αρχείου hdp_manual_install_rpm_helper_files-2.6.0.3.8.tar.gz που περιέχει το αρχείο yarn-utils.py

Οι επιλογές (command line options) του script για τη συστάδα του Πίνακα 6-2 είναι (Cores=2, Memory=4GB, Disks=1, HBASE=False)

```
python yarn-utils.py --cores 2 --memory 4 --disks 1 --hbase False
```



(α)

(β)

Σχήμα 6-112: (α) Τροποποίηση του yarn-utils.py για να είναι συμβατό με την Python3, (β) Τρέξιμο του yarn-utils.py. Κατά την διάρκεια των ρυθμίσεων βελτιστοποίησης της συστάδας (performance tuning) τροποποιήθηκε το yarn.scheduler.minimum-allocation-mb σε 512mb για να κλιμακώνονται καλύτερα (σε βήματα των 512mb) τα containers σε σχέση με τη συνολική διαθέσιμη μνήμη του κόμβου (yarn.nodemanager.resource.memory-mb=3072mb)

Παρατηρώντας τα αποτελέσματα στο Σχήμα 6-112β διαπιστώνει κανείς ότι η διαθέσιμη μνήμη είναι 3GB, δηλαδή 1GB (OS RAM + Hadoop Daemons + μνήμη για λοιπές διεργασίες) λιγότερο από τη συνολική μνήμη.

Παρατήρηση: Κατά την διάρκεια των ρυθμίσεων βελτιστοποίησης της συστάδας (performance tuning) τροποποιήθηκε το ***yarn.scheduler.minimum-allocation-mb*** *σε* ***512mb*** για να κλιμακώνονται καλύτερα (σε βήματα των 512mb) τα containers σε σχέση με τη συνολική διαθέσιμη μνήμη του κόμβου (yarn.nodemanager.resource.memory-mb=3072mb). Ακολούθως συμπληρώνει ο χρήστης τα αρχεία mapred-site.xml και yarn-site.xml:

mapred-site.xml:

```
gedit mapred-site.xml
```

```
<configuration>

  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>yarn.app.mapreduce.am.env</name>
    <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
  </property>
  <property>
    <name>mapreduce.map.env</name>
    <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
  </property>
  <property>
    <name>mapreduce.reduce.env</name>
    <value>$HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
  </property>
  <!--  Εξορισμού τιμή για mapreduce.map.memory.mb  1024MB -->
  <property>
    <name>mapreduce.map.memory.mb</name>
    <value>1024</value>
  </property>
 <!--  Εξορισμού τιμή για mapreduce.reduce.memory.mb  3072MB -->
<property>
    <name>mapreduce.reduce.memory.mb</name>
    <value>2048</value>
  </property>
<!-- Μέγεθος σωρού - μικρότερη τιμή (~80%) από αυτή για το mapreduce.map.memory.mb
και mapreduce.reduce.memory.mb  αντίστοιχα -->
  <property>
    <name>mapreduce.map.java.opts</name>
    <value>-Xmx819m</value>
  </property>
  <property>
    <name>mapreduce.reduce.java.opts</name>
    <value>-Xmx1638m</value>
  </property>
  <property>
    <name>yarn.app.mapreduce.am.resource.mb</name>
    <value>2048</value>
  </property>
  <property>
    <name> yarn.app.mapreduce.am.command-opts </name>
    <value>-Xmx1638m</value>
  </property>
  <property>
    <name> mapreduce.task.io.sort.mb</name>
    <value>409</value>
  </property>
</configuration>
```

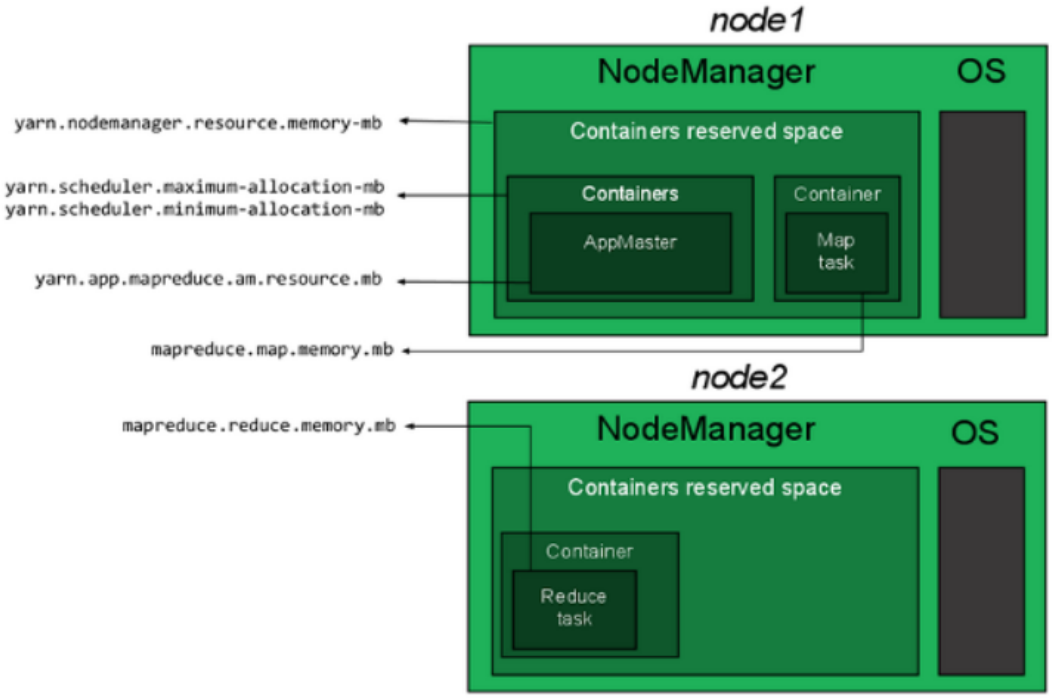
yarn-site.xml:

```
gedit  yarn-site.xml
```

```
<configuration>
<!-- Site specific YARN configuration properties -->
<!-- Διαμορφώσεις για τον ResourceManager και τον NodeManager-->
  <property>
    <name>yarn.acl.enable</name>
```

```
    <value>0</value>
  </property>
<!-- Διαμόρφωση για τον ResourceManager -->
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>master</value>
  </property>
  <property>
    <name>yarn.scheduler.maximum-allocation-mb</name>
    <value>3072</value>
    <description> Πόση μνήμη μπορεί να καταναλώσει ένα container. Η εξορισμού τιμή
της είναι 8192MB </description>
  </property>
  <property>
    <name>yarn.scheduler.minimum-allocation-mb</name>
    <value>512</value>
    <description>Η ελάχιστη κατανομή μνήμης που επιτρέπεται σε ένα container. Η
εξορισμού τιμή της είναι 1024MB </description>
  </property>
<!-- Διαμόρφωση για NodeManager -->
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>
  <property>
    <name>yarn.nodemanager.resource.memory-mb</name>
    <value>3072</value>
    <description> Πόση μνήμη μπορεί να διατεθεί για YARN containers σε έναν μόνο κόμβο
(=Node's Total RAM – (OS RAM + Hadoop Daemons + μνήμη για λοιπές διεργασίες)). Πρέπει
να είναι μικρότερη από τη συνολική μνήμη RAM του κόμβου (Node's Total RAM).  Η
εξορισμού τιμή της είναι 8192MB  </description>
  </property>
  <property>
    <name>yarn.nodemanager.vmem-check-enabled</name>
    <value>false</value>
  </property>
<!-- Αριθμός πυρήνων CPU που μπορούν να διατεθούν για containers από τον NodeManager.
-->
 <property>
    <name>yarn.nodemanager.resource.cpu-vcores</name>
    <value>2</value>
  </property>
  <property>
    <name>yarn.scheduler.minimum-allocation-vcores</name>
    <value>1</value>
  </property>
  <property>
    <name>yarn.scheduler.maximum-allocation-vcores</name>
    <value>2</value>
  </property>
</configuration>
```

Στον σύνδεσμο αυτό παρέχονται σύντομες πληροφορίες σχετικές με τις μεταβλητές στις παραπάνω διαμορφώσεις του αρχείου yarn-site.xml.



Πηγή: https://www.linode.com/docs/databases/hadoop/how-to-install-and-set-up-hadoop-cluster/

Σχήμα 6-113: Οι μεταβλητές μνήμης που απαιτούνται να καθοριστούν σε μια συστάδα Hadoop YARN

workers:

```
gedit workers
```

```
#workers
slave1
slave2
```

Ακολούθως ο χρήστης θα πρέπει να δημιουργήσει τους καταλόγους *tmp*, *datanode* και *namenode* και στη συνέχεια να αλλάξει τον ιδιοκτήτη τους καθώς και τα δικαιώματά τους (βλέπε Σχήμα 6-114).

```
sudo mkdir -p /usr/local/hadoop_store/tmp
sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode
sudo mkdir -p /usr/local/hadoop_store/hdfs/datanode

ls -al /usr/local
sudo chown -R hadoopuser:hadoopgroup /usr/local/hadoop_store/
sudo chmod 750 /usr/local/hadoop_store/
sudo chmod 777 /usr/local/hadoop_store/tmp
ls -al /usr/local
ls -al /usr/local/hadoop_store/
```

```
(base) hadoopuser@master:/usr/local$ sudo mkdir -p /usr/local/hadoop_store/tmp
(base) hadoopuser@master:/usr/local$ sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode
(base) hadoopuser@master:/usr/local$ sudo mkdir -p /usr/local/hadoop_store/hdfs/datanode
(base) hadoopuser@master:/usr/local$ ls -al /usr/local/
total 48
drwxr-xr-x 12 root       root        4096 Δεκ   4 21:34 .
drwxr-xr-x 14 root       root        4096 Απρ  16  2019 ..
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 bin
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 etc
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 games
drwxr-xr-x  9 hadoopuser hadoopgroup 4096 Σεπ  10 19:51 hadoop
drwxr-xr-x  4 root       root        4096 Δεκ   4 21:34 hadoop_store
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 include
drwxr-xr-x  3 root       root        4096 Απρ  16  2019 lib
lrwxrwxrwx  1 root       root           9 Νοε  24 20:15 man -> share/man
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 sbin
drwxr-xr-x  5 root       root        4096 Απρ  16  2019 share
(base) hadoopuser@master:/usr/local$ sudo chown -R hadoopuser:hadoopgroup /usr/local/hadoop_store/
(base) hadoopuser@master:/usr/local$ sudo chmod 750  /usr/local/hadoop_store
(base) hadoopuser@master:/usr/local$ ls -al
total 48
drwxr-xr-x 12 root       root        4096 Δεκ   4 21:34 .
drwxr-xr-x 14 root       root        4096 Απρ  16  2019 ..
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 bin
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 etc
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 games
drwxr-xr-x  9 hadoopuser hadoopgroup 4096 Σεπ  10 19:51 hadoop
drwxr-x---  4 hadoopuser hadoopgroup 4096 Δεκ   4 21:34 hadoop_store
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 include
drwxr-xr-x  3 root       root        4096 Απρ  16  2019 lib
lrwxrwxrwx  1 root       root           9 Νοε  24 20:15 man -> share/man
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 sbin
drwxr-xr-x  5 root       root        4096 Απρ  16  2019 share
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 src
(base) hadoopuser@master:/usr/local$ ls -al hadoop_store/
total 16
drwxr-x---  4 hadoopuser hadoopgroup 4096 Δεκ   4 21:34 .
drwxr-xr-x 12 root       root        4096 Δεκ   4 21:34 ..
drwxr-xr-x  4 hadoopuser hadoopgroup 4096 Δεκ   4 21:35 hdfs
drwxr-xr-x  2 hadoopuser hadoopgroup 4096 Δεκ   4 21:34 tmp
(base) hadoopuser@master:/usr/local$
(base) hadoopuser@master:/usr/local$ sudo chmod 777 /usr/local/hadoop_store/tmp
[sudo] password for hadoopuser:
(base) hadoopuser@master:/usr/local$ ls -al hadoop_store/
total 16
drwxr-x---  4 hadoopuser hadoopgroup 4096 Δεκ   4 21:34 .
drwxr-xr-x 12 root       root        4096 Δεκ   4 21:34 ..
drwxr-xr-x  4 hadoopuser hadoopgroup 4096 Δεκ   4 21:35 hdfs
drwxrwxrwx  4 hadoopuser hadoopgroup 4096 Ιαν   2 02:34 tmp
(base) hadoopuser@master:/usr/local$
```

Σχήμα 6-114: Δημιουργία των καταλόγων tmp, datanode και namenode καθώς και αλλαγή ιδιοκτήτη και τροποποίηση των δικαιωμάτων σ' αυτούς τους καταλόγους.

Στη συνέχεια θα πρέπει ο χρήστης να διαμορφώσει τον κόμβο NameNode (βλέπε Σχήμα 6-115) για το λόγο αυτό πληκτρολογεί στη γραμμή εντολών:

```
hdfs namenode -format
```

Αυτή η εντολή θα πρέπει να εκτελεστεί στον master κόμβο μόνο την πρώτη φορά που δημιουργείται η συστάδα. Η διαμόρφωση του συστήματος αρχείων σημαίνει την προετοιμασία του καταλόγου που καθορίζεται από την *dfs.namenode.name.dir* μεταβλητή.

Σχήμα 6-115: Αρχικοποίηση του συστήματος αρχείων HDFS (διαμόρφωση του NameNode).

Παρατηρήσεις:

- Ο κατάλογος *datanode* είναι αναγκαίος στον master κόμβο μόνο εφόσον ο χρήστης επιθυμεί ο συγκεκριμένος κόμβος να εκτελεί και εργασίες slave (να αποθηκεύει δεδομένα).
- Ο κατάλογος *namenode* είναι αναγκαίος μόνο στον *master* (NameNode) κόμβο για τον λόγο αυτό μετά την δημιουργία των κλώνων θα πρέπει να καταργηθεί από τους slave κόμβους.
- Στον *spark-client* ο κατάλογος *hadoop_store* μπορεί να καταργηθεί αφού δεν εξυπηρετεί κανένα σκοπό.
- Ο έλεγχος της συστάδας Hadoop θα υλοποιηθεί μετά:
    - τη λήψη και εγκατάσταση του Spark
    - τη δημιουργία των κλώνων του master (slave1, slave2, spark-client),
    - τη ρύθμιση του SSH για αυθεντικοποίηση χρήστη χωρίς κωδικό πρόσβασης
    - τις τροποποιήσεις που προαναφέρθηκαν.

Λήψη και εγκατάσταση του Spark

Αναλυτικές πληροφορίες σχετικά με την εγκατάσταση του Spark παρέχονται στην ενότητα "Εγκατάσταση του PySpark στο Ubuntu 19.04 σε τοπική λειτουργία (local mode)". Αφού επιλέξει ο χρήστης τον σύνδεσμο της έκδοσης Spark που επιθυμεί από την ιστοσελίδα των αρχειοθετημένων εκδόσεων του Spark (περιλαμβάνει όλες τις εκδόσεις) στο παράθυρο που ανοίγει αντιγράφει την έκδοση Spark που έχει κατάληξη -bin-hadoop2.7.tgz. Στη συνέχεια αφού ανοίξει μια οθόνη του τερματικού πληκτρολογεί (βλέπε Σχήμα 6-116):

```
cd ~/Downloads
wget https://archive.apache.org/dist/spark/spark-2.4.3/spark-2.4.3-bin-hadoop2.7.tgz
ls -al spark-2.4.3-bin-hadoop2.7.tgz
```

```
(base) hadoopuser@master:~/Downloads$ wget https://archive.apache.org/dist/spark/spark-2.4.3/spark-2.4.3-bin-hadoop2.7.tgz
--2019-12-05 01:35:48--  https://archive.apache.org/dist/spark/spark-2.4.3/spark-2.4.3-bin-hadoop2.7.tgz
Resolving archive.apache.org (archive.apache.org)... 163.172.17.199
Connecting to archive.apache.org (archive.apache.org)|163.172.17.199|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 229988313 (219M) [application/x-gzip]
Saving to: 'spark-2.4.3-bin-hadoop2.7.tgz'

spark-2.4.3-bin-hadoop2.7.tgz     100%[=============================================================>] 219,33M   716KB/s    in 11m 0s

2019-12-05 01:46:48 (340 KB/s) - 'spark-2.4.3-bin-hadoop2.7.tgz' saved [229988313/229988313]

(base) hadoopuser@master:~/Downloads$ ls -al spark-2.4.3-bin-hadoop2.7.tgz
-rw-r--r-- 1 hadoopuser hadoopgroup 229988313 Μαΐ   1  2019 spark-2.4.3-bin-hadoop2.7.tgz
(base) hadoopuser@master:~/Downloads$
```

Σχήμα 6-116: *Λήψη* spark-2.4.3-bin-hadoop2.7.tgz

Με τις παρακάτω εντολές μπορεί ο χρήστης να αποσυμπιέσει και εξαγάγει τον αρχείο *spark-2.4.3-bin-hadoop2.7.tgz* στον ομώνυμο φάκελο *spark-2.4.3-bin-hadoop2.7* που βρίσκεται στη διαδρομή ~/Downloads και ακολούθως να τον μεταφέρει στη θέση */opt/spark* (βλέπε Σχήμα 6-117).

```
tar xvf spark-2.4.3-bin-hadoop2.7.tgz
ls -al
sudo mv spark-2.4.3-bin-hadoop2.7/  /opt/spark
ls -al /opt/spark
```

```
(base) hadoopuser@master:~/Downloads$ tar xvf spark-2.4.3-bin-hadoop2.7.tgz
spark-2.4.3-bin-hadoop2.7/
spark-2.4.3-bin-hadoop2.7/python/
spark-2.4.3-bin-hadoop2.7/python/setup.cfg
```

(α)

```
(base) hadoopuser@master:~/Downloads$ ls -al
total 1633820
drwxr-xr-x  3 hadoopuser hadoopgroup      4096 Δεκ   5 01:53 .
drwxr-xr-x 17 hadoopuser hadoopgroup      4096 Δεκ   5 01:47 ..
-rw-r--r--  1 hadoopuser hadoopgroup 541906131 Ιουλ 25 17:59 Anaconda3-2019.07-Linux-x86_64.sh
-rw-r--r--  1 hadoopuser hadoopgroup 541906131 Ιουλ 25 17:59 Anaconda3-2019.07-Linux-x86_64.sh.1
-rw-r--r--  1 hadoopuser hadoopgroup 359196911 Σεπ  23 08:16 hadoop-3.2.1.tar.gz
drwxr-xr-x 13 hadoopuser hadoopgroup      4096 Μαϊ   1  2019 spark-2.4.3-bin-hadoop2.7
-rw-r--r--  1 hadoopuser hadoopgroup 229988313 Μαϊ   1  2019 spark-2.4.3-bin-hadoop2.7.tgz
(base) hadoopuser@master:~/Downloads$ sudo mv spark-2.4.3-bin-hadoop2.7/  /opt/spark
[sudo] password for hadoopuser:
(base) hadoopuser@master:~/Downloads$ ls -al /opt
total 20
drwxr-xr-x  5 root       root        4096 Δεκ   5 01:55 .
drwxr-xr-x 20 root       root        4096 Δεκ   2 19:56 ..
drwxr-xr-x 26 hadoopuser hadoopgroup 4096 Δεκ   3 20:14 anaconda
drwxr-xr-x 13 hadoopuser hadoopgroup 4096 Μαϊ   1  2019 spark
drwxr-xr-x  9 root       root        4096 Νοε  24 21:04 VBoxGuestAdditions-6.0.14
(base) hadoopuser@master:~/Downloads$ ls -al /opt/spark
total 136
drwxr-xr-x 13 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 .
drwxr-xr-x  5 root       root         4096 Δεκ   5 01:55 ..
drwxr-xr-x  2 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 bin
drwxr-xr-x  2 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 conf
drwxr-xr-x  5 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 data
drwxr-xr-x  4 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 examples
drwxr-xr-x  2 hadoopuser hadoopgroup 12288 Μαϊ   1  2019 jars
drwxr-xr-x  4 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 kubernetes
-rw-r--r--  1 hadoopuser hadoopgroup 21316 Μαϊ   1  2019 LICENSE
drwxr-xr-x  2 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 licenses
-rw-r--r--  1 hadoopuser hadoopgroup 42919 Μαϊ   1  2019 NOTICE
drwxr-xr-x  9 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 python
drwxr-xr-x  3 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 R
-rw-r--r--  1 hadoopuser hadoopgroup  3952 Μαϊ   1  2019 README.md
-rw-r--r--  1 hadoopuser hadoopgroup   164 Μαϊ   1  2019 RELEASE
drwxr-xr-x  2 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 sbin
drwxr-xr-x  2 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 yarn
(base) hadoopuser@master:~/Downloads$
```

(β)

Σχήμα 6-117: (α) Αποσυμπίεση και εξαγωγή του αρχείου spark-2.4.3-bin-hadoop2.7.tgz στον ομώνυμο φάκελο spark-2.4.3-bin-hadoop2.7, (β) μεταφορά του φακέλου spark-2.4.3-bin-hadoop2 στη θέση /opt/spark.

Αφού ανοίξει ο χρήστης το αρχείο *.bashrc* με το *gedit*:

```
gedit ~/.bashrc
```

θα πρέπει να καθορίσει τη μεταβλητή περιβάλλοντος SPARK_HOME και να προσθέσει το μονοπάτι για τα SPARK binaries αρχεία στη μεταβλητή PATH, εισάγοντας στο αρχείο τις ακόλουθες γραμμές (βλέπε Σχήμα 6.118):

```
export SPARK_HOME='/opt/spark'
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin:$SPARK_HOME/jars
```

```
#-----------------------spark-client:----------------------------
#------------------- master, slave1,slave2, ..., slaven ------------------
# >>>>             SPARK Περιβαλλοντικές Μεταβλητές ΑΡΧΗ            >>>>
#-------------------------------------------------------------------------

export SPARK_HOME='/opt/spark'
# Μπορείτε να προσθέσετε πολλαπλές καταχωρήσεις ταυτόχρονα
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin:$SPARK_HOME/jars
#export PATH=$PATH:$SPARK_HOME/sbin

# <<<<SPARK             Περιβαλλοντικές Μεταβλητές ΤΕΛΟΣ             <<<<
```

Σχήμα 6-118: Ενημέρωση του αρχείου .bashrc με τη μεταβλητή περιβάλλοντος SPARK_HOME και προσθήκη στη μεταβλητή PATH του μονοπατιού για τα SPARK binaries αρχεία.

Αφού κλείσει και στη συνέχεια ανοίξει ένα νέο τερματικό θα πρέπει ο χρήστης να ελέγξει αν εγκαταστάθηκε το Spark σωστά πληκτρολογώντας στη γραμμή εντολών (βλέπε Σχήμα 6-119):

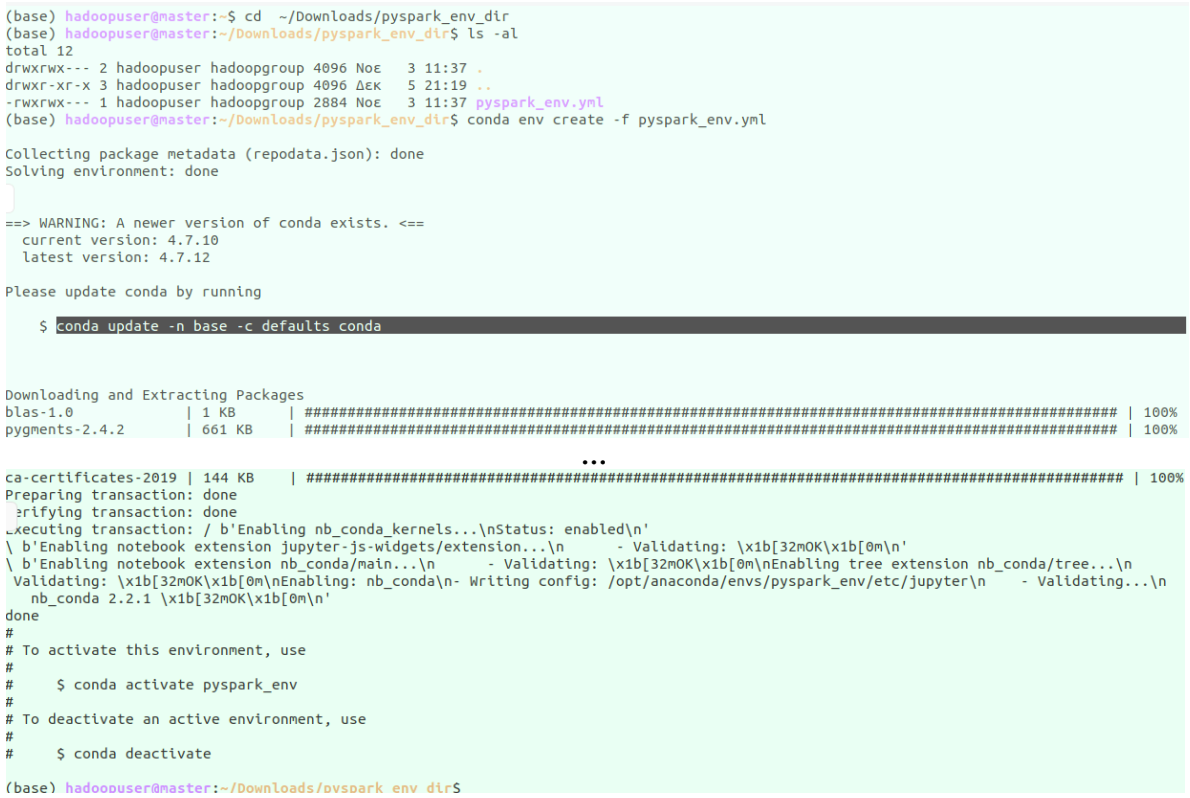
```
pyspark --version
```

```
(base) hadoopuser@master:~$ pyspark  --version
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.3
      /_/

Using Scala version 2.11.12, OpenJDK 64-Bit Server VM, 1.8.0_222
Branch
Compiled by user  on 2019-05-01T05:08:38Z
Revision
Url
Type --help for more information.
(base) hadoopuser@master:~$
```

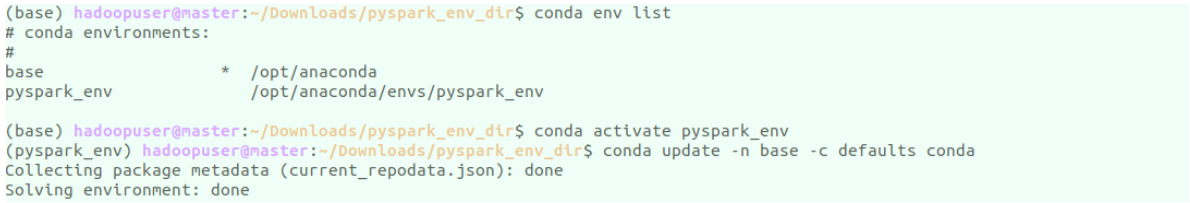
Σχήμα 6-119: Έλεγχος ορθής εγκατάστασης του pyspark

Για να μπορεί ο χρήστης να τρέχει το pyspark μέσα από το Jupyter Notebook απαιτείται η εγκατάσταση των πακέτων findspark, και jupyter. Για λόγους συντομίας θα αναδημιουργηθεί το περιβάλλον conda με ονομασία *pyspark_env* που δημιουργήθηκε στο 4ο βήμα της ενότητας "Εγκατάσταση του PySpark στο Ubuntu 19.04 σε τοπική λειτουργία (local mode)") με βάσει το αρχείο *pyspark_env.yml* που δημιουργήθηκε στο 6ο βήμα της προαναφερθείσας ενότητας (για περισσότερες πληροφορίες σχετικά με τη δημιουργία ενός περιβάλλοντος conda, τον εμπλουτισμό του με πακέτα και την αποθήκευση του σε ένα αρχείο yaml βλέπε στην ενότητα "Διαχείριση Έργων"). Στο περιβάλλον conda *pyspark_env* είχαν προστεθεί τα πακέτα *findspark*, *jupyter* (βλέπε 5ο βήμα της προαναφερθείσας ενότητας), *pandas* καθώς και το *nb_conda* (6ο βήμα της προαναφερθείσας ενότητας) που επιτρέπει στους χρήστες να έχουν πρόσβαση σε πυρήνες από άλλα περιβάλλοντα conda μέσα από το Jupyter. Πρέπει ο χρήστης αφού μεταφέρει τον κατάλογο *pyspark_env_dir* μέσα στο κατάλογο *Downloads* του master κόμβου να πληκτρολογήσει στη γραμμή εντολών:

```
cd  ~/Downloads/pyspark_env_dir
ls -al
# Αναδημιουργία του περιβάλλοντος conda pyspark_env
conda env create -f pyspark_env.yml
# Εμφάνιση των περιβαλλόντων conda του master
conda env list
# Ενεργοποίηση περιβάλλοντος pyspark_env
conda activate pyspark_env
# Ενημέρωση του πακέτου conda στο περιβάλλον base
conda update -n base -c defaults conda
```



Σχήμα 6-120: Αναδημιουργία του περιβάλλοντος conda pyspark_env



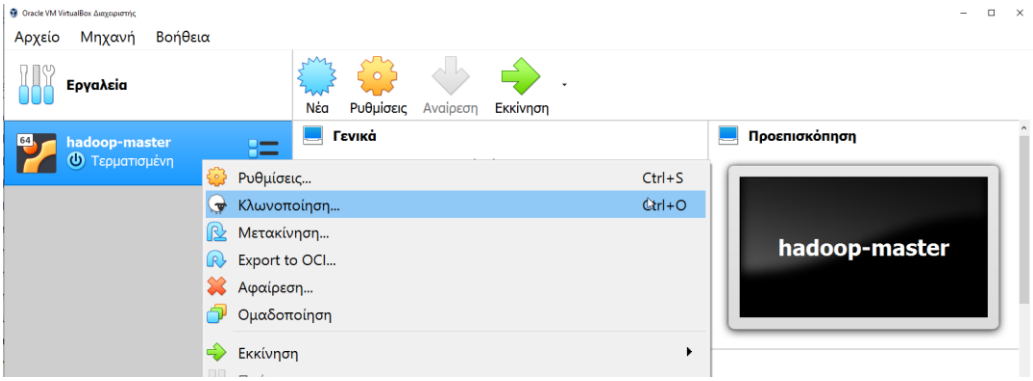
Σχήμα 6-121: Παρουσίαση των conda περιβαλλόντων - Ενεργοποίηση περιβάλλοντος pyspark_env – Ενημέρωση του πακέτου conda στο περιβάλλον base.

Παρατήρηση: Η διαμόρφωση των αρχείων *spark-env.sh* και *spark-defaults.conf* και ο έλεγχος της συστάδας Spark σε *client mode* και *cluster mode* θα υλοποιηθεί παρακάτω, αμέσως μετά τον έλεγχο της συστάδας Hadoop YARN.

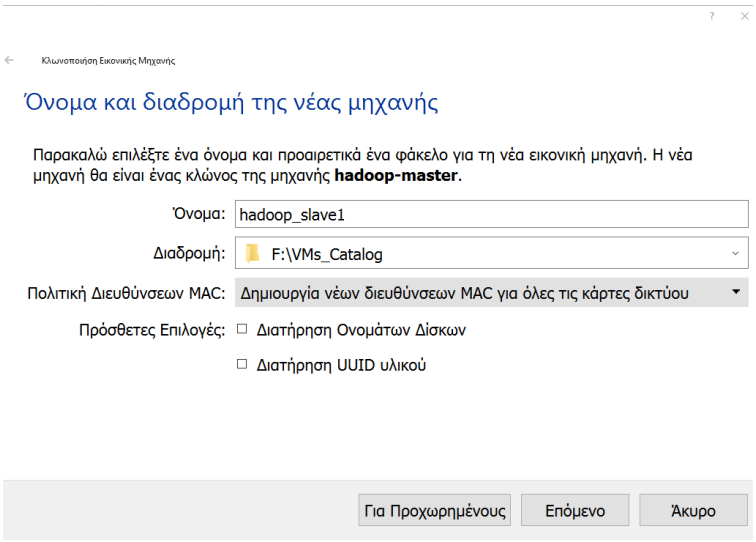
Δημιουργία κλώνων

Στη συνέχεια θα πρέπει ο χρήστης να δημιουργήσει τρεις κλώνους του *hadoop-master*, τους *hadoop-slave1*, *hadoop-slave2* και *spark-client*. Θα πρέπει να διευκρινισθεί ότι αυτά είναι τα

ονόματα των ΕΜ δηλαδή αυτά που φαίνονται μέσα στο VirtualBox. Τα ονόματα υπολογιστών (hostnames) των ΕΜ είναι αντίστοιχα master, *slave1*, *slave2* και *client*. Παρακάτω φαίνεται η διαδικασία κλωνοποίησης του *hadoop-master*, αλλαγής του ονόματος σύνδεσης (login name) και του ονόματος υπολογιστή από *master* σε *slave1* αλλά και αλλαγής της στατικής διεύθυνσης *IP*.

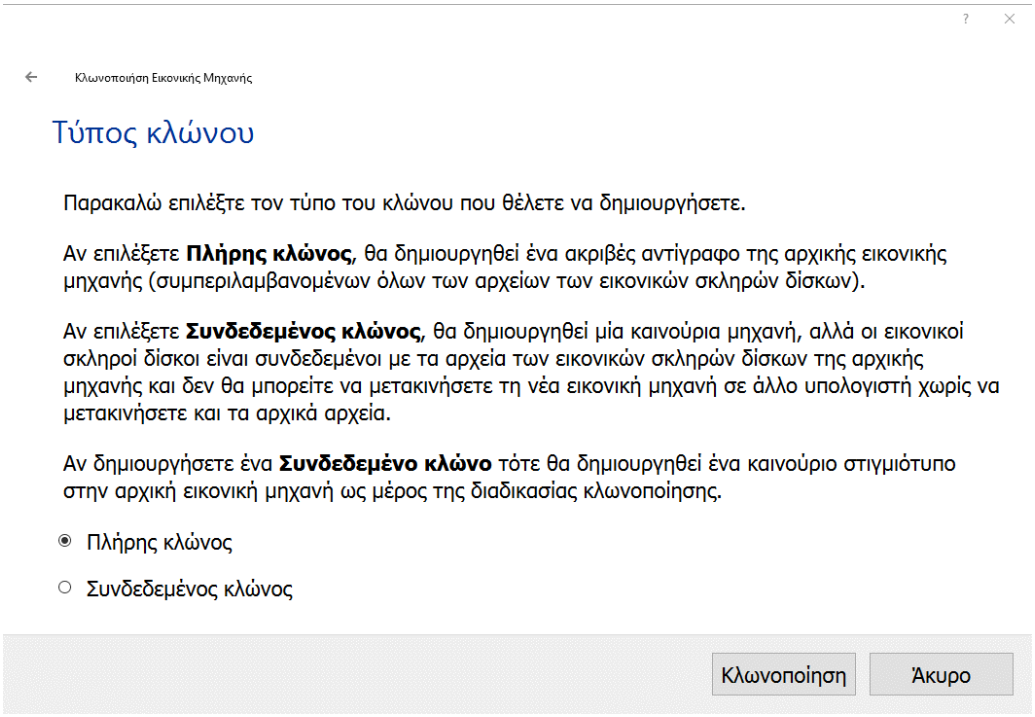


(α)

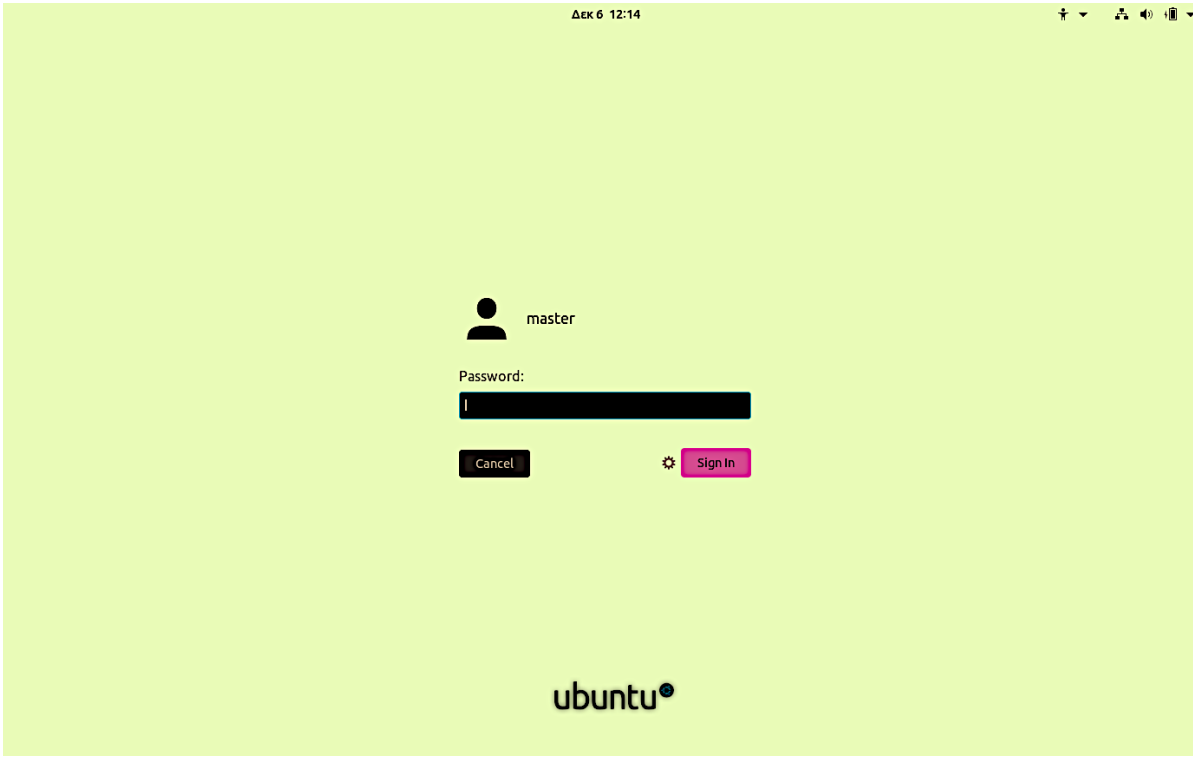


(β)

Σχήμα 6-122: (α) Έναρξη της διαδικασίας κλωνοποίησης της ΕΜ hadoop-master, (β) Ορισμός ονόματος και διαδρομής της νέας ΕΜ. Θα πρέπει ο χρήστης να προσέξει ιδιαιτέρως η "Πολιτική Διευθύνσεων MAC" να είναι αυτή που φαίνεται στην εικόνα.

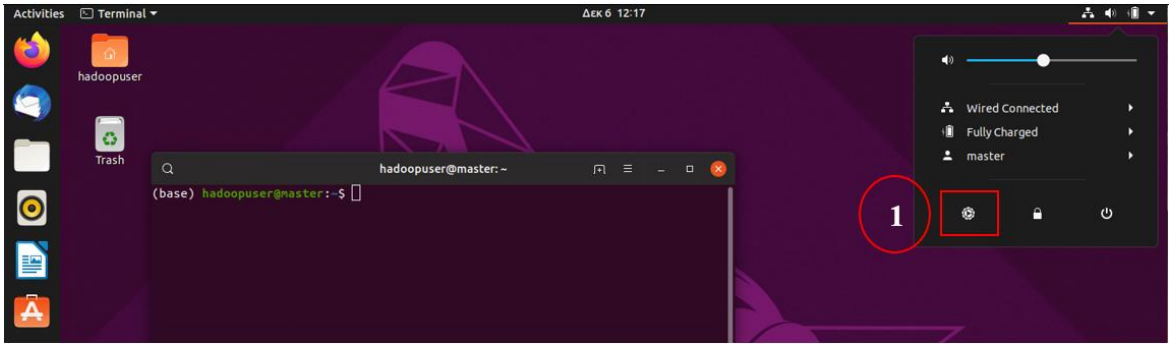


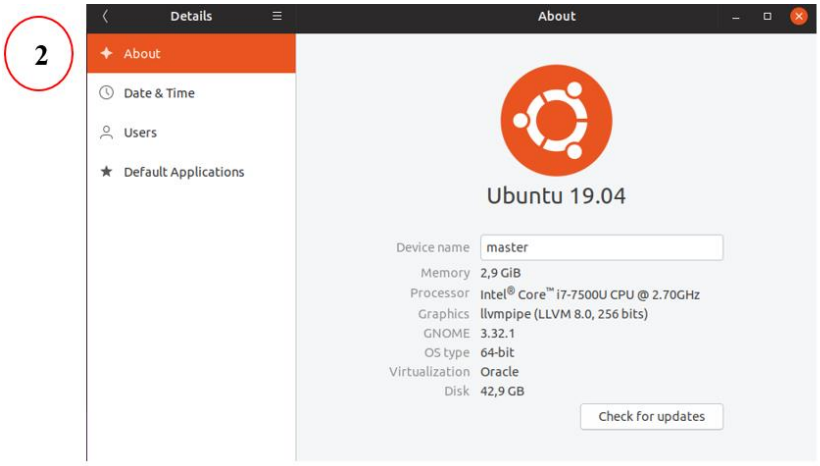
(α)

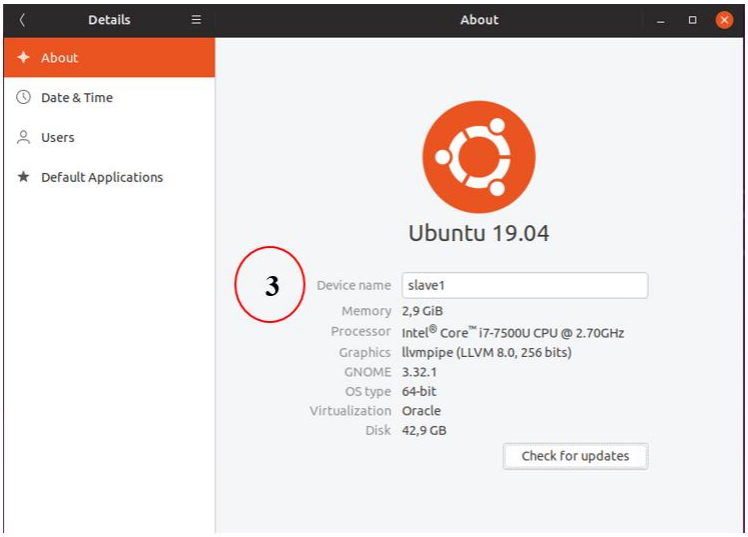


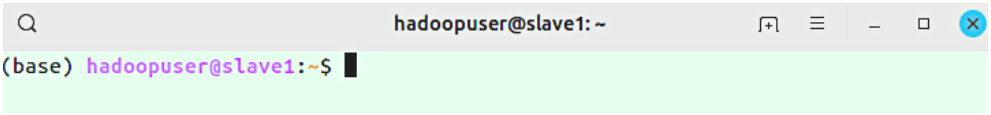
(β)

Σχήμα 6-123: (α) Επιλογή τύπου κλώνου της ΕΜ, (β) Παρατηρεί κανείς ότι το όνομα σύνδεσης (login name) του υπολογιστή της νέας ΕΜ παρέμεινε master

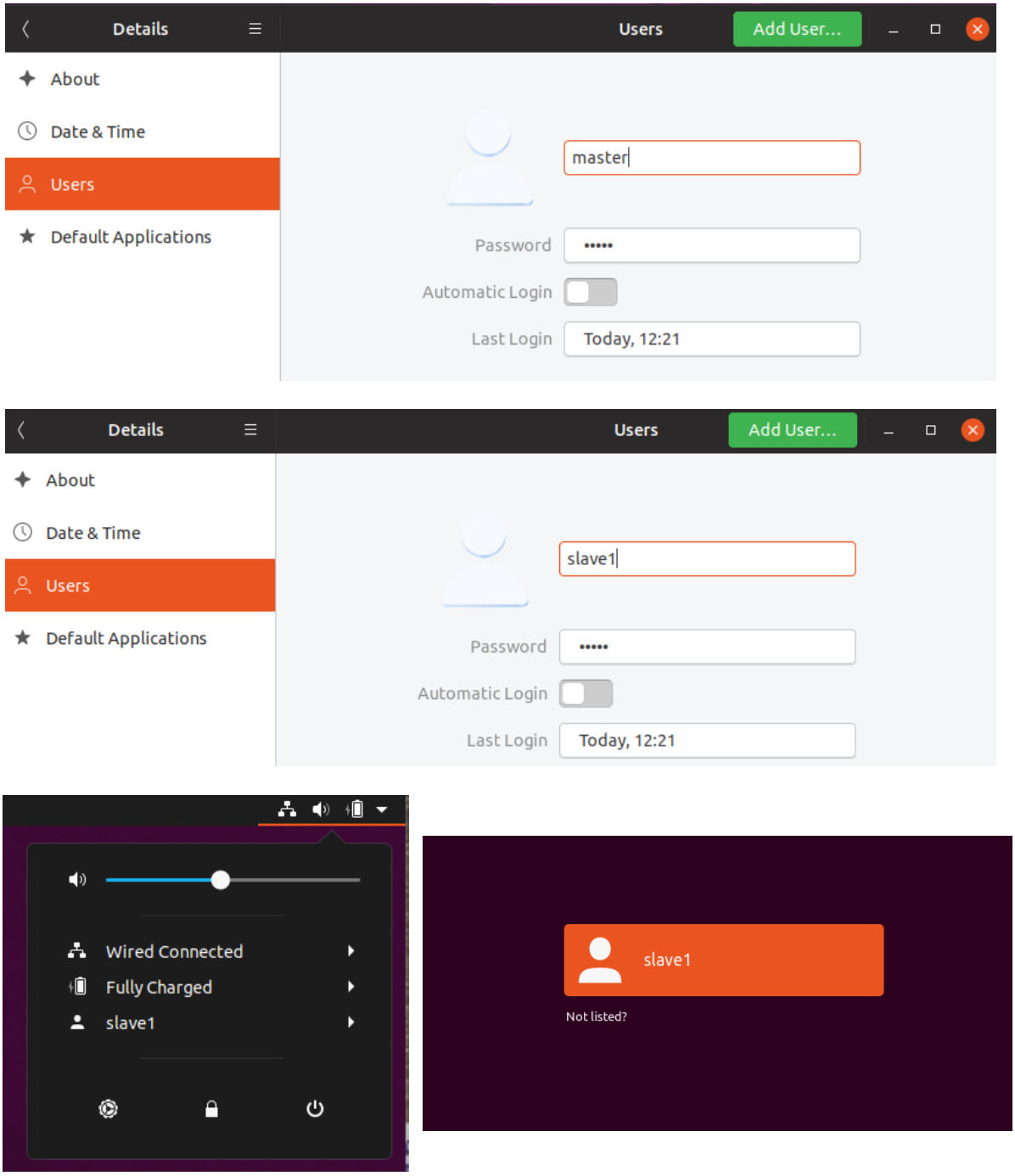






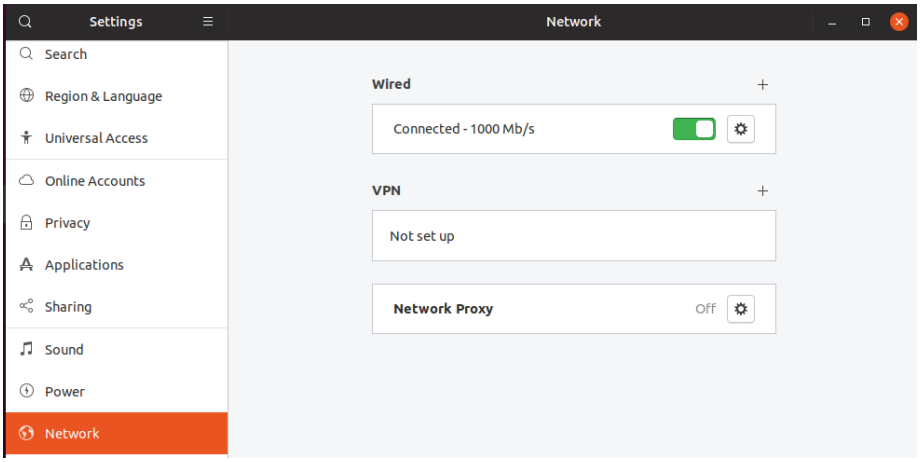


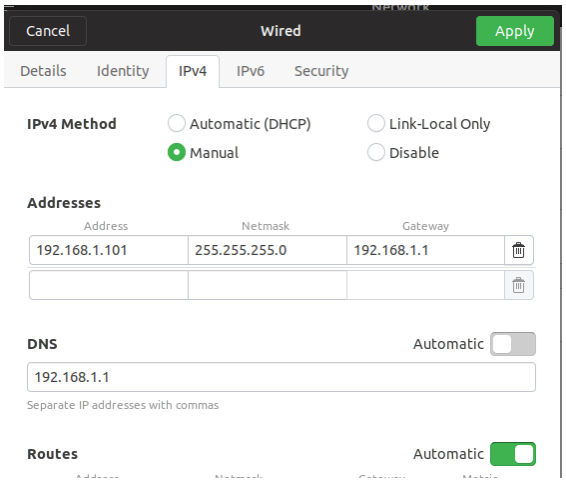
Σχήμα 6-124: Αλλαγή του ονόματος του υπολογιστή (hostname) του κλώνου hadoop-slave1 από master σε slave1



Σχήμα 6-125: Αλλαγή του ονόματος σύνδεσης (login name) του κλώνου hadoop-slave1 από master σε slave1

Ακολουθώντας ο χρήστης τις οδηγίες που περιγράφονται στην υποενότητα “ Ρύθμιση της κάρτας Δικτύου στο VirtualBox και της στατικής διεύθυνσης IP στο Ubuntu” θα πρέπει να ρυθμίσει την στατική διεύθυνση *IP* του κλώνου με βάση τον Πίνακα 6-2.



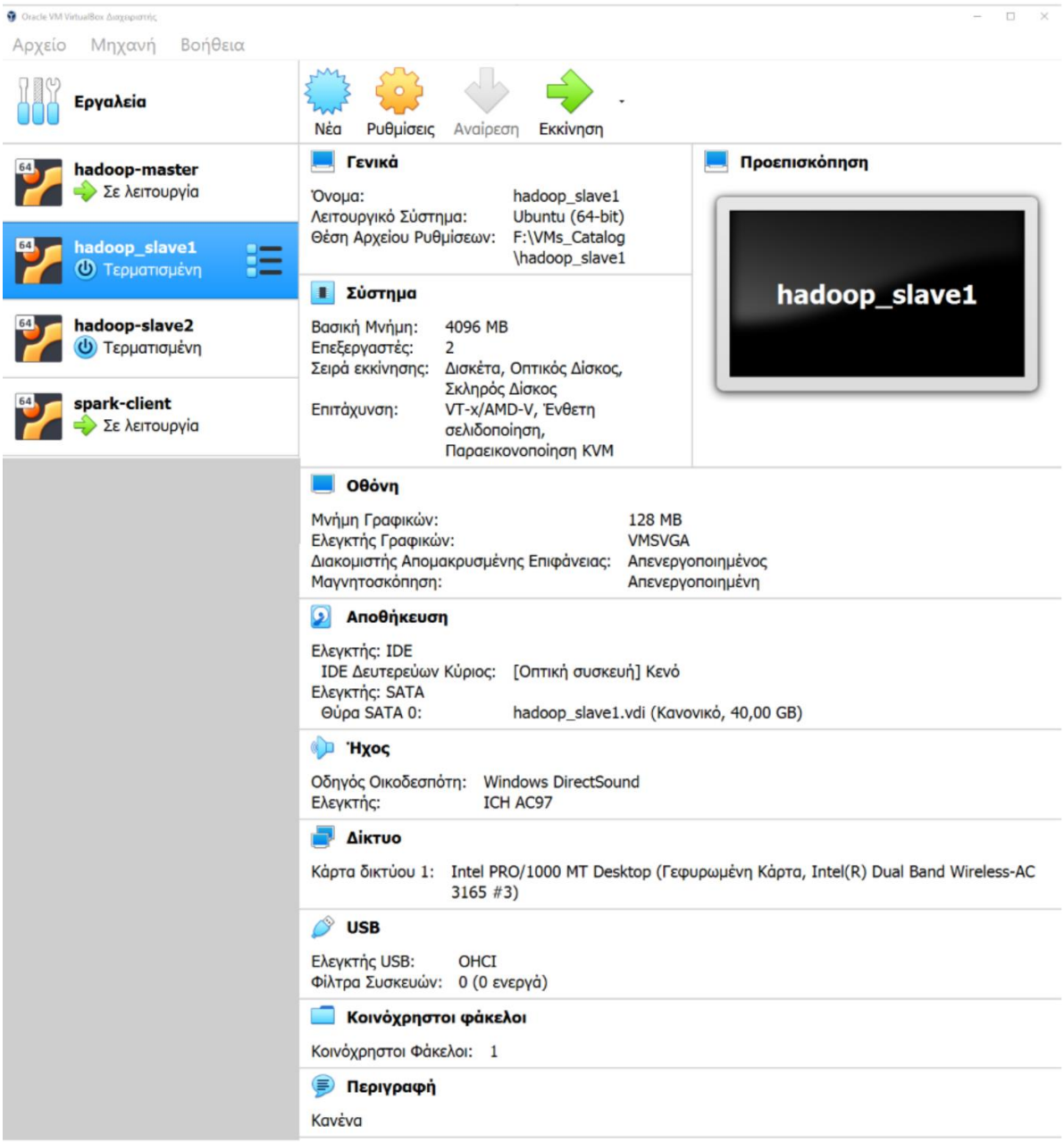



```
(base) hadoopuser@slave1:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:73:dc:3a brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.101/24 brd 192.168.1.255 scope global noprefixroute enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::aea7:6a6:ae73:7c8a/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Σχήμα 6-126: Αλλαγή στατικής διεύθυνσης στον κλώνο hadoop-slave1

Παρατήρηση: Ο χρήστης πρέπει να αλλάξει το μέγεθος της μνήμης και τον αριθμό των επεξεργαστών στους κόμβους ***slave1***, ***slave2*** και ***spark-client*** σύμφωνα με τον *Πίνακα 6-2*. Ο τρόπος αλλαγής των ανωτέρω φαίνεται στα Σχήματα 6-67 και 6-68 αντίστοιχα. Στο Σχήμα 6.127 φαίνεται ο κόμβος ***slave1*** [31]μετά τις τροποποιήσεις σύμφωνα με τον Πίνακα 6-2.

[31] *Εάν ο Η/Υ του χρήστη δεν διαθέτει ικανοποιητικό αριθμό λογικών πυρήνων. Μπορεί να χρησιμοποιήσει δυο Η/Υ, στον πρώτο να δημιουργήσει τις ΕΜ hadoop-master και hadoop-slave1 ενώ στον δεύτερο τις ΕΜ hadoop-client και hadoop-slave2.*



Σχήμα 6-127: Η κεντρική οθόνη του VirtualBox Διαχειριστής μετά την δημιουργία όλων των κλώνων σύμφωνα με τον Πίνακα 6-2. Στην οθόνη φαίνονται οι συνολικές ρυθμίσεις του κόμβου slave1

Ρύθμιση του SSH για αυθεντικοποίηση χρήστη χωρίς κωδικό πρόσβασης

Το *SSH* (*Secure Shell ή Secure Socket Shell / ασφαλές κέλυφος*) είναι ένα πρωτόκολλο δικτύου ανοικτού κώδικα που χρησιμοποιείται για ασφαλή σύνδεση μεταξύ ενός πελάτη και ενός διακομιστή για την εκτέλεση εντολών και προγραμμάτων στο διακομιστή απομακρυσμένα από τον πελάτη, την αυτοματοποίηση εργασιών όπως δημιουργία αντιγράφων ασφαλείας μέσω scripts, καθώς και για την αντιγραφή ή τον συγχρονισμό αρχείων μεταξύ πελάτη και διακομιστή μέσω του *SCP* (*Secure Copy Protocol*) που βασίζεται στο *SSH* (για

αυθεντικοποίηση και κωδικοποίηση) και στο *BSD RCP* (για τη μεταφορά των αρχείων). Βασικά το *SSH* υποστηρίζει δύο τρόπους επαλήθευσης της ταυτότητας χρήστη με το διακομιστή. Τον έλεγχο ταυτότητας με κωδικό πρόσβασης (*password-based authentication*) και τον έλεγχο ταυτότητας με δημόσιο κλειδί (*public key authentication*).

Η ρύθμιση της σύνδεσης για *"έλεγχο ταυτότητας με πιστοποίηση χωρίς κωδικό"* (*ssh passwordless login*) προϋποθέτει τη δημιουργία ενός ζεύγους κλειδιών *ssh* (*δημόσιου* και *ιδιωτικού κλειδιού*) στο μηχάνημα-πελάτη. Το *ιδιωτικό κλειδί ssh* παραμένει ασφαλές στο σταθμό εργασίας του χρήστη και το δημόσιο κλειδί ssh (το αρχείο με επέκταση *.pub*) θα πρέπει στη συνέχεια να το προσθέσει ο χρήστης στο αρχείο *authorized_keys* στον κατάλογο *~/.ssh* του λογαριασμού του που διατηρεί στους απομακρυσμένους διακομιστές που επιθυμεί να έχει πρόσβαση. Στη συνέχεια παρουσιάζεται ο τρόπος δημιουργίας του *ζεύγους κλειδιών ssh* (*δημόσιου* και *ιδιωτικού* / *public* and *private*) της *ΕΜ hadoop-master* (*με hostname: master*) με χρήση της εντολής *ssh-keygen* που περιλαμβάνεται στην τυπική σουίτα εργαλείων *OpenSSH* (βλέπε Σχήμα 6-128).

```
ssh-keygen -t rsa -b 4096 -P ""
```

όπου -t: ο αλγόριθμος δημόσιου κλειδιού (rsa, dsa, ecdsa, ed25519), -b: το μήκος κλειδιού (key size) σε bits, -P "" : no passphrase

```
(base) hadoopuser@master:~$ ssh-keygen -t rsa -b 4096 -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hadoopuser/.ssh/id_rsa):
Created directory '/home/hadoopuser/.ssh'.
Your identification has been saved in /home/hadoopuser/.ssh/id_rsa.
Your public key has been saved in /home/hadoopuser/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:ltL7wjyssgfOr4oQNDIvR980daLvr6taifQMGkMjayI hadoopuser@master
The key's randomart image is:
+---[RSA 4096]----+
|        o .      |
|       o o       |
|+o+   +          |
|oB.o o + .       |
|E.= + o S        |
|++ =.= = .       |
|. .o..=+o        |
|..  +.. *o       |
|. ..+B+oo=o      |
+----[SHA256]-----+
(base) hadoopuser@master:~$
```

Σχήμα 6-128: Δημιουργία ζεύγους κλειδιών ssh (ιδιωτικού και δημόσιου)

Παρακάτω επιδεικνύεται πως μπορεί ο χρήστης να ελέγξει την σύνδεση μέσω *ssh* χωρίς κωδικό, αντιγράφοντας το δημόσιο κλειδί της ΕΜ στο αρχείο *~/.ssh/authorized_keys* της ίδιας ΕΜ (*hadoop-master*) και εκτελώντας την εντολή *ssh master* (βλέπε Σχήμα 6-129).

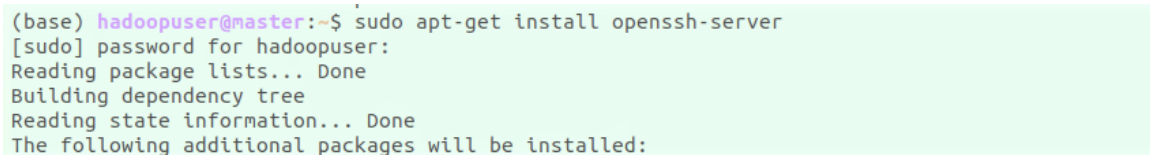
```
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
# προβολή στο τερματικό του περιεχομένου του αρχείου
cat .ssh/authorized_keys
# Δοκιμή  για το αν αν μπορεί να συνδεθεί χωρίς κωδικό στον λογαριασμό του (master)
ssh master
```

```
(base) hadoopuser@master:~$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
(base) hadoopuser@master:~$ cat .ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQCwoyQkOnP3waY+2rS35A/mNSlGu0uO/+mfoRzuco9w/rnv32F7sFn5Ln
USaXpA6ozO6RtubG6b/s8f4NMVIpoQMXRwK25c916mLPfdUWjj15x+4rfkAI4HtCxwYqLUg5jD+XBtSBVPRj8Pdq8kzHzn
fwRv3deKKw3qPWgnM25CJQ8vSvx75OZgPdn8EheBfJBviVrv9DoDkCBeJ19yuk2DQVELFmbBP5jLDU+/UIeAefZIr279fy
Lk1S1aPGet2eJoOxkPvWYaWY9KRAM5WQRHLZo9bZrtcKqdYldRuAzkcTt5uWmlnAKxLV9frkdzbTbGmOqoxedUTmLpr3Jw
afNSQZPpfDWzqb6WtXoJwDPrwr+3fGirxoTPOZ7ygBEGqFdx2sOJyhXgut0tc1YU5/Js3KbOQYIOP7DbnIYHMyKiwMS1e2
yEm0iQHkpQP2nN2O0JHQvmDCut/W7q0PFvahDHw2PuzcELxSMEyTLLk36rt5/q7WUzyRqgJ708tDtzz8eB2eqRfnYmV0eX
MhiNqasAviNEUQXaMnXtRNwRC4zifIQVqyWmg0zgjW/tkgMdu1QK5XMsnuYZhtJB4renHrS0Vrc+JqYPMxQ8frN6IdEjaV
GLGahg6crMpirdLHvkrwkUg0nQvJeemfQZ4KgnPVJz5nmM08xJNQPXyMJcaikfkw== hadoopuser@master
(base) hadoopuser@master:~$ ssh master
ssh: connect to host master port 22: Connection refused
```

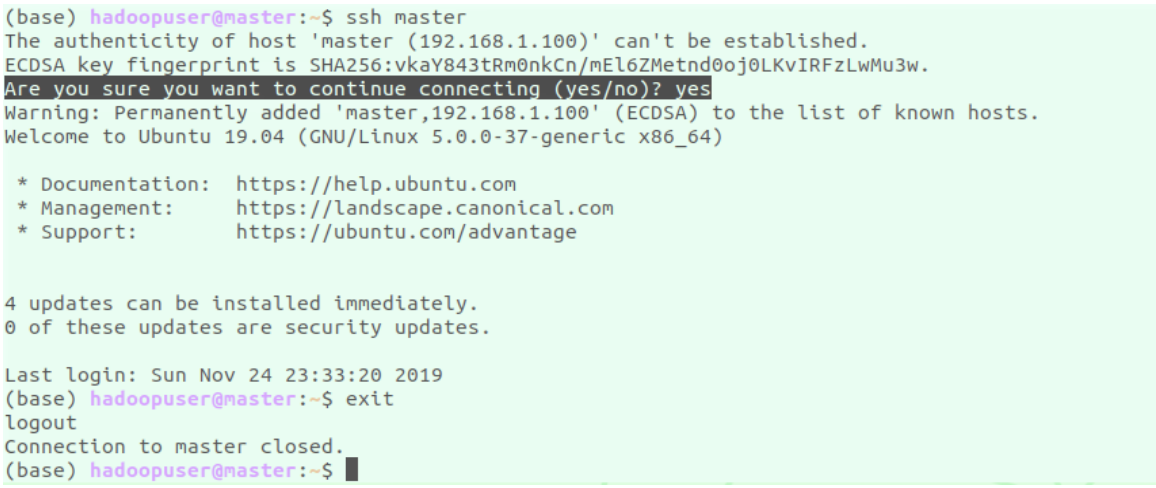
Σχήμα 6-129: Αντιγραφή του δημόσιου κλειδιού του κόμβου master στο αρχείο ~/.ssh/authorized_keys του ιδίου κόμβου και ανεπιτυχής προσπάθεια σύνδεσης στον κόμβο master μέσω ssh λόγω απουσίας της υπηρεσίας openssh-server.

Παρατηρεί κανείς ότι δεν κατέστη δυνατή η σύνδεση στον *master* μέσω *ssh* ενώ παράλληλα εκτυπώθηκε στην οθόνη το μήνυμα: *"ssh: connect to host master port 22: Connection refused"* που υποδηλώνει ότι δεν είναι εγκατεστημένη η υπηρεσία *openssh-server*. Αφού εγκαταστήσει ο χρήστης την υπηρεσία αυτή (βλέπε Σχήμα 6-130α) θα πρέπει να προσπαθήσει να συνδεθεί εκ νέου με τον *master* μέσω *ssh*. Την πρώτη φορά που συνδέεται κανείς σε έναν διακομιστή μέσω *SSH*, ο *SSH πελάτης* (*client*) εμφανίζει στον χρήστη το μήνυμα του Σχήματος 6-130β που τον πληροφορεί ότι η αυθεντικότητα του διακομιστή δεν μπορεί να διαπιστωθεί. Αποδεχόμενος τον κίνδυνο ο χρήστης, αφού βεβαιωθεί ότι προσπαθεί να συνδεθεί στον σωστό διακομιστή, πληκτρολογεί '*yes*' για να συνδεθεί στον διακομιστή ενώ ταυτόχρονα αποθηκεύεται το δημόσιο κλειδί του διακομιστή στο αρχείο *~/.ssh/known_hosts* του πελάτη έτσι ώστε να μπορεί να τον ταυτοποιήσει σε μελλοντική σύνδεση με τον συγκεκριμένο διακομιστή (*μηχανισμός γνωστών εξυπηρετητών – known_hosts mechanism*). Η ασφάλεια που παρέχει ο μηχανισμός αυτός έγκειται στο γεγονός πως εάν ο χρήστης προσπαθήσει να συνδεθεί σε μεταγενέστερη ημερομηνία και ο απομακρυσμένος διακομιστής εμφανίζει ένα άλλο δημόσιο κλειδί, τότε η σύνδεση απορρίπτεται καθώς το κλειδί δεν είναι αξιόπιστο για τον συγκεκριμένο διακομιστή:

```
sudo apt-get install openssh-server
ssh master
exit
```



(α)



(β)

Σχήμα 6-130: (α) Εγκατάσταση της υπηρεσίας openssh-server, (β) Σύνδεση στον διακομιστή master μέσω ssh και ακολούθως αποσύνδεση με την εντολή exit

Αφού εγκαταστήσει ο χρήστης την *υπηρεσία openssh-server* στους κόμβους *slave1* και *slave2* θα πρέπει να μεταφορτώσει το *Δημόσιο Κλειδί* του κόμβου *master* στους διακομιστές *slave1* και *slave2* με την εντολή *ssh-copy-id* που αντιγράφει το δημόσιο κλειδί του master στο αρχείο *~/.ssh/authorized_keys* των ανωτέρω διακομιστών (βλέπε Σχήμα 6-131).

```
# Κόμβος slave1
sudo apt-get install openssh-server

#Κόμβος slave2
sudo apt-get install openssh-server

# Κόμβος master
ssh-copy-id slave1
ssh-copy-id slave2
ssh slave1

# Κόμβος slave1
exit

# Κόμβος master
ssh slave2

# Κόμβος slave1
exit
```

```
(base) hadoopuser@slave1:~$ sudo apt-get install openssh-server
[sudo] password for hadoopuser:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
```

(α)

```
(base) hadoopuser@slave2:~$ sudo apt-get install  openssh-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
```

(β)

```
(base) hadoopuser@master:~$ ssh-copy-id slave1
The authenticity of host 'slave1 (192.168.1.101)' can't be established.
ECDSA key fingerprint is SHA256:ViOjWqjT/RqZ580s4QQedDsGw3C8zf5aKUCkOZnVhIg.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new key
s
hadoopuser@slave1's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'slave1'"
and check to make sure that only the key(s) you wanted were added.

(base) hadoopuser@master:~$ ssh slave1
Welcome to Ubuntu 19.04 (GNU/Linux 5.0.0-37-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage


4 updates can be installed immediately.
0 of these updates are security updates.

New release '19.10' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sun Dec  8 04:00:49 2019 from 192.168.1.100
(base) hadoopuser@slave1:~$ exit
logout
Connection to slave1 closed.
(base) hadoopuser@master:~$
```

(γ)

```
(base) hadoopuser@master:~$ ssh-copy-id slave2
The authenticity of host 'slave2 (192.168.1.102)' can't be established.
ECDSA key fingerprint is SHA256:ZH0rg1GZafqJqdm1m7R/ilFmqqe/eME9BD9ULNt1/F4.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new key
s
hadoopuser@slave2's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'slave2'"
and check to make sure that only the key(s) you wanted were added.

(base) hadoopuser@master:~$ ssh slave2
Last login: Sun Dec  8 03:38:31 2019 from 192.168.1.102
(base) hadoopuser@slave2:~$ exit
logout
Connection to slave2 closed.
(base) hadoopuser@master:~$
```

(δ)

Σχήμα 6-131: (α),(β) Εγκατάσταση υπηρεσίας openssh-server στους κόμβους slave1 & slave2 αντίστοιχα , (γ),(δ) Μεταφόρτωση του Δημόσιου Κλειδιού του κόμβου master στους διακομιστές slave1 και slave2 -Σύνδεση του master στους κόμβους slave1και slave2 μέσω ssh και στη συνέχεια αποσύνδεση του master.

Ακολούθως θα πρέπει ο χρήστης να δημιουργήσει ένα *ζεύγος κλειδιών ssh* (*δημόσιου* και *ιδιωτικού* / *public* and *private*) στην *EM spark-client* (βλέπε Σχήμα 6-132) και να εγκαταστήσει

την *υπηρεσία openssh-server* στην *EM spark-client* (με hostname: *spark-client*) και να μεταφορτώσει το *Δημόσιο Κλειδί* του *spark-client* στους διακομιστές *master*, *slave1* και *slave2* (βλέπε Σχήμα 6-133).

```
# Στον κόμβο spark-client
#-------------------------------------------------------------------------------
----
ssh-keygen -t rsa -b 4096 -P ""
sudo apt-get install openssh-server
ssh-copy-id master
ssh master

# Στον κόμβο master
#-------------------------------------------------------------------------------
----
exit

# Στον κόμβο spark-client
#-------------------------------------------------------------------------------
----
ssh-copy-id slave1
ssh slave1

# Στον κόμβο slave1
#-------------------------------------------------------------------------------
----
exit

# Στον κόμβο spark-client
#-------------------------------------------------------------------------------
----
ssh-copy-id slave2
ssh slave2

# Στον κόμβο slave2
#-------------------------------------------------------------------------------
----
exit
```

```
(base) hadoopuser@spark-client:~$ ssh-keygen -t rsa -b 4096 -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hadoopuser/.ssh/id_rsa):
Your identification has been saved in /home/hadoopuser/.ssh/id_rsa.
Your public key has been saved in /home/hadoopuser/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:dPJDEhQJvXFB5bZXspmIEE0RvbVPb/jByJyyTB5CpzY hadoopuser@spark-client
The key's randomart image is:
+---[RSA 4096]----+
|      .+=***.    |
|        +oo.. .  |
|        =+o oo...|
|       ..O +.o.*.|
|        S * = O+.|
|         E = *.o+|
|        . * +  o.|
|           +    .|
|                 |
+----[SHA256]-----+
```

Σχήμα 6-132: Δημιουργία ζεύγους κλειδιών ssh (ιδιωτικού και δημόσιου) στον κόμβο spark-client

```
(base) hadoopuser@spark-client:~$ sudo apt-get install openssh-server
[sudo] password for hadoopuser:
Reading package lists... Done
Building dependency tree
Reading state information... Done
(base) hadoopuser@spark-client:~$ ssh-copy-id master
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
hadoopuser@master's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'master'"
and check to make sure that only the key(s) you wanted were added.

(base) hadoopuser@spark-client:~$ ssh master
Welcome to Ubuntu 19.04 (GNU/Linux 5.0.0-37-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage


4 updates can be installed immediately.
0 of these updates are security updates.

New release '19.10' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sun Dec  8 11:55:43 2019 from 192.168.1.100
(base) hadoopuser@master:~$ exit
logout
Connection to master closed.
(base) hadoopuser@spark-client:~$
(base) hadoopuser@spark-client:~$ ssh-copy-id slave1
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
hadoopuser@slave1's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'slave1'"
and check to make sure that only the key(s) you wanted were added.

(base) hadoopuser@spark-client:~$ ssh slave1
Welcome to Ubuntu 19.04 (GNU/Linux 5.0.0-37-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage


4 updates can be installed immediately.
0 of these updates are security updates.

New release '19.10' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Mon Dec  9 12:16:28 2019 from 192.168.1.110
(base) hadoopuser@slave1:~$ exit
logout
Connection to slave1 closed.
(base) hadoopuser@spark-client:~$
(base) hadoopuser@spark-client:~$ ssh-copy-id slave2
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
hadoopuser@slave2's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'slave2'"
and check to make sure that only the key(s) you wanted were added.

(base) hadoopuser@spark-client:~$ ssh slave2
Last login: Sun Dec  8 22:47:59 2019 from 192.168.1.100
(base) hadoopuser@slave2:~$ exit
logout
Connection to slave2 closed.
(base) hadoopuser@spark-client:~$
```

Σχήμα 6-133: Εγκατάσταση υπηρεσίας openssh-server στον spark-client - Μεταφόρτωση του Δημόσιου Κλειδιού του κόμβου spark-client στους διακομιστές master, slave1 και slave2.

Τροποποιήσεις και τελικές ρυθμίσεις στους κόμβους

Στη συνέχεια θα πρέπει ο χρήστης να ολοκληρώσει τις τροποποιήσεις που αναφέρονται στις παρατηρήσεις της ενότητας "*Λήψη και εγκατάσταση του Hadoop YARN*" καθώς και στη παρατήρηση της ενότητας "*Λήψη και εγκατάσταση του Spark*":

- Κατάργηση των καταλόγων *namenode* και *hadoop_store* από τους *slave* κόμβους και από τον *spark-client* αντίστοιχα (βλέπε Σχήμα 6-134):

```
# slave1
rm -R /usr/local/hadoop_store/hdfs/namenode
#slave2
rm -R /usr/local/hadoop_store/hdfs/namenode
# spark-client
sudo rm -R /usr/local/hadoop_store
```

```
(base) hadoopuser@slave1:~$ rm -R /usr/local/hadoop_store/hdfs/namenode
(base) hadoopuser@slave1:~$ ls -al /usr/local/hadoop_store/hdfs
total 12
drwxr-xr-x 3 hadoopuser hadoopgroup 4096 Δεκ   9 02:46 .
drwxr-x--- 4 hadoopuser hadoopgroup 4096 Δεκ   4 21:34 ..
drwx------ 3 hadoopuser hadoopgroup 4096 Δεκ   8 23:21 datanode
(base) hadoopuser@slave1:~$
```

```
(base) hadoopuser@slave2:~$ rm -R /usr/local/hadoop_store/hdfs/namenode
(base) hadoopuser@slave2:~$ ls -al /usr/local/hadoop_store/hdfs
total 12
drwxr-xr-x 3 hadoopuser hadoopgroup 4096 Δεκ   9 02:48 .
drwxr-x--- 4 hadoopuser hadoopgroup 4096 Δεκ   4 21:34 ..
drwx------ 3 hadoopuser hadoopgroup 4096 Δεκ   8 23:20 datanode
(base) hadoopuser@slave2:~$
```

```
(base) hadoopuser@spark-client:~$ sudo rm -R /usr/local/hadoop_store
[sudo] password for hadoopuser:
(base) hadoopuser@spark-client:~$ ls -al /usr/local
total 44
drwxr-xr-x 11 root       root        4096 Δεκ   9 02:52 .
drwxr-xr-x 14 root       root        4096 Απρ  16  2019 ..
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 bin
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 etc
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 games
drwxr-xr-x 10 hadoopuser hadoopgroup 4096 Δεκ   4 22:24 hadoop
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 include
drwxr-xr-x  3 root       root        4096 Απρ  16  2019 lib
lrwxrwxrwx  1 root       root           9 Νοε  24 20:15 man -> share/man
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 sbin
drwxr-xr-x  5 root       root        4096 Απρ  16  2019 share
drwxr-xr-x  2 root       root        4096 Απρ  16  2019 src
(base) hadoopuser@spark-client:~$
```

Σχήμα 6-134: Κατάργηση του καταλόγου namenode από τους slave κόμβους και του hadoop_store από τον spark-client

Έλεγχος της συστάδας Hadoop YARN

Θα πρέπει στην αρχή να εκκινήσει ο χρήστης το *Hadoop HDFS* (βλέπε Σχήμα 6-135) καθώς και το *YARN* (βλέπε Σχήμα 6-135) και να βεβαιωθεί ότι τρέχουν όλες οι υπηρεσίες σε όλους τους κόμβους με την εντολή *jps* [32].

```
# Εκκίνηση HDFS - Στον κόμβο master θα πρέπει να τρέχουν οι java διεργασίες NameNode
και SecondaryNameNode – (Το σταμάτημα του HDFS γίνεται με την εντολή: stop-dfs.sh)
start-dfs.sh
jps
# Στον κόμβο slave1 - θα πρέπει να τρέχει η java διεργασία DataNode
jps
# Στον κόμβο slave2 - θα πρέπει να τρέχει η java διεργασία DataNode
Jps
# Εκκίνηση YARN - Στον κόμβο master θα πρέπει να τρέχουν οι java διεργασίες NameNode,
SecondaryNameNode και ResourceManager.
# (Το σταμάτημα του YARN γίνεται αντίστοιχα με την εντολή: stop-yarn.sh)
start-yarn.sh
jps
# Στον κόμβο slave1- θα πρέπει να τρέχουν οι java διεργασίες DataNode και NodeManager
jps
# Στον κόμβο slave2- θα πρέπει να τρέχουν οι java διεργασίες DataNode και NodeManager
jps
```

Σχήμα 6-135: Εκκίνηση Hadoop HDFS

[32] *Χρησιμοποιείται για τον έλεγχο όλων των δαιμόνων (daemons) Hadoop όπως οι NameNode, DataNode, ResourceManager, NodeManager κ.λπ. που εκτελούνται σε έναν κόμβο.*

Σχήμα 6-136: Εκκίνηση Hadoop YARN

Επίσης μετά την εκκίνηση του *Hadoop HDFS* μπορεί να πληκτρολογήσει ο χρήστης την ακόλουθη εντολή (βλέπε Σχήμα 6-137) που του παρέχει μια σύντομη αναφορά για το συνολικό σύστημα αρχείων *HDFS*:

```
hdfs dfsadmin -report
```

Σχήμα 6-137: Η εντολή hdfs dfsadmin -report παρέχει στον χρήστη μια σύντομη αναφορά για το συνολικό σύστημα αρχείων HDFS

Στη συνέχεια θα πρέπει ο χρήστης να βεβαιωθεί για την σωστή λειτουργία του *Hadoop HDFS*. Επ' ευκαιρία παρουσιάζονται μερικές βασικές εντολές για το κατανεμημένο σύστημα *HDFS* [34] (βλέπε Σχήμα 6-138):

```
# Βασικές εντολές για το κατανεμημένο σύστημα αρχείων (distributed file system) HDFS
# Όλες οι εντολές για το κατανεμημένο σύστημα αρχείων ξεκινούν με: hdfs dfs
# Εντολή ls για την απαρίθμηση των αρχείων και καταλόγων σε έναν κατάλογο του HDFS
hdfs dfs -ls /
hdfs dfs -ls /user
# Εντολή mkdir για τη δημιουργία καταλόγου στο HDFS
hdfs dfs -mkdir -p /user/hadoopuser/testHDFS
hdfs dfs -ls /user/hadoopuser/
# Εντολή για το τοπικό σύστημα αρχείων (local filesystem) του linux
cd /usr/local/hadoop

# Εντολή put για μεταφορά αρχείων/καταλόγων από το τοπικό σύστημα αρχείων στο κατανεμημένο σύστημα HDFS
hdfs dfs -put README.txt /user/hadoopuser/testHDFS/
hdfs dfs -ls /user/hadoopuser/testHDFS/
# Εντολή cat για την εκτύπωση των περιεχομένων ενός αρχείου αποθηκευμένου στο HDFS
hdfs dfs -cat /user/hadoopuser/testHDFS/README.txt | head
# Εντολή get για μεταφορά αρχείων/καταλόγων από το κατανεμημένο σύστημα HDFS στο τοπικό σύστημα αρχείων
hdfs dfs -get /user/hadoopuser/testHDFS/README.txt /usr/local/hadoop/README2.txt
# Εντολή για το τοπικό σύστημα αρχείων
cat /user/hadoopuser/testHDFS/README2.txt | head

# Εντολή mv για τη μεταφορά αρχείων εντός του HDFS
hdfs dfs -mkdir -p /user/hadoopuser/testHDFS/test/
hdfs dfs -mv  /user/hadoopuser/testHDFS/README.txt /user/hadoopuser/testHDFS/test/
hdfs dfs -ls /user/hadoopuser/testHDFS/test/
# Εντολή  cp για την αντιγραφή αρχείων εντός του HDFS
hdfs dfs -cp /user/hadoopuser/testHDFS/test/README.txt /user/hadoopuser/testHDFS/
# Εντολή rm -R για την διαγραφή αναδρομικά (recursively) αρχείων  και καταλόγων εντός του HDFS
hdfs dfs -rm -R /user/hadoopuser/testHDFS/
hdfs dfs -ls /user/hadoopuser/
```

Διαδικτυακή διεπαφή χρήστη (Web UI) για το HDFS (NameNode UI)

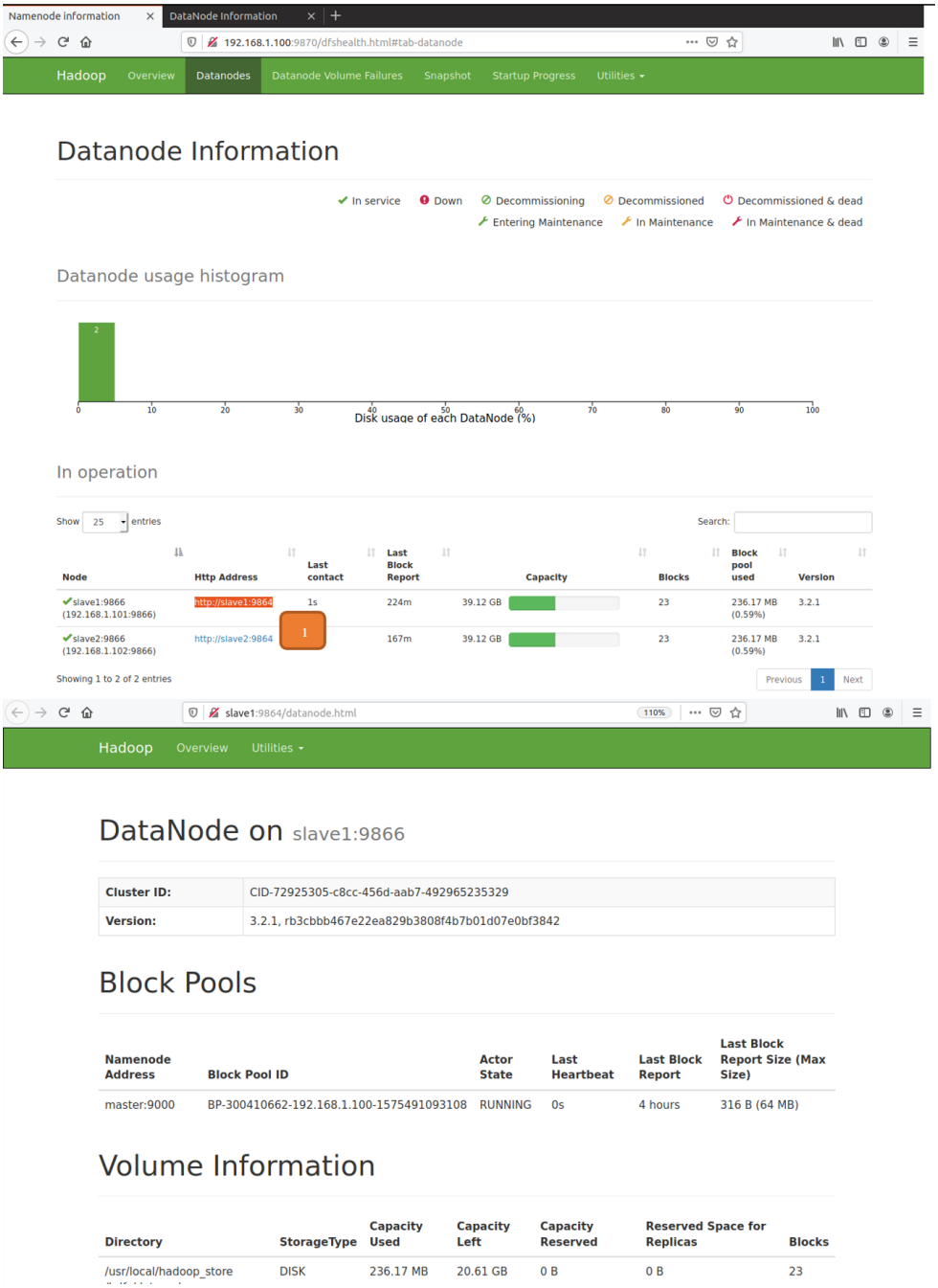
Εκτός από τη διεπαφή γραμμής εντολών, το Hadoop παρέχει επίσης μια διαδικτυακή διεπαφή χρήστη (Web UI) τόσο για το HDFS (NameNode UI) όσο και για τον Διαχειριστή πόρων YARN (YARN Resource Manager UI). Τόσο ο NameNode όσο και οι DataNodes τρέχουν έναν εσωτερικό διακομιστή ιστού που εμφανίζει χρήσιμες πληροφορίες για την τρέχουσα κατάσταση της συστάδας. Με την προεπιλεγμένη διαμόρφωση, η αρχική σελίδα (σελίδα επισκόπησης) του NameNode βρίσκεται στην διεύθυνση http://<namenode-hostname>: 9870/. Η σελίδα αυτή εμφανίζει τα δεδομένα των DataNodes της συστάδας καθώς και τα βασικά στατιστικά στοιχεία της (βλέπε Σχήμα 6-139) [35].

```
(base) hadoopuser@master:~$ hdfs dfs -ls /
Found 2 items
drwx------   - hadoopuser hadoopgroup          0 2019-12-09 13:10 /tmp
drwxr-xr-x   - hadoopuser hadoopgroup          0 2019-12-10 20:16 /user
(base) hadoopuser@master:~$ hdfs dfs -ls /user
(base) hadoopuser@master:~$ hdfs dfs -mkdir -p /user/hadoopuser/test
(base) hadoopuser@master:~$ hdfs dfs -ls /user/hadoopuser
Found 1 items
drwxr-xr-x   - hadoopuser hadoopgroup          0 2019-12-10 20:18 /user/hadoopuser/test
(base) hadoopuser@master:~$ hdfs dfs -rm -R /user/hadoopuser/test
Deleted /user/hadoopuser/test
(base) hadoopuser@master:~$ hdfs dfs -ls /user
Found 1 items
drwxr-xr-x   - hadoopuser hadoopgroup          0 2019-12-10 20:18 /user/hadoopuser
(base) hadoopuser@master:~$ hdfs dfs -ls /user/hadoopuser
(base) hadoopuser@master:~$ hdfs dfs -rm -R /user/hadoopuser/
Deleted /user/hadoopuser
(base) hadoopuser@master:~$ hdfs dfs -ls /
Found 2 items
drwx------   - hadoopuser hadoopgroup          0 2019-12-09 13:10 /tmp
drwxr-xr-x   - hadoopuser hadoopgroup          0 2019-12-10 20:19 /user
(base) hadoopuser@master:~$ hdfs dfs -ls /user
(base) hadoopuser@master:~$ hdfs dfs -mkdir -p /user/hadoopuser/testHDFS
(base) hadoopuser@master:~$ hdfs dfs -ls /user/hadoopuser/
Found 1 items
drwxr-xr-x   - hadoopuser hadoopgroup          0 2019-12-10 20:20 /user/hadoopuser/testHDFS
(base) hadoopuser@master:~$ cd /usr/local/hadoop
(base) hadoopuser@master:/usr/local/hadoop$ hdfs dfs -put README.txt /user/hadoopuser/testHDFS/
2019-12-10 20:24:24,182 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted =
 false
(base) hadoopuser@master:/usr/local/hadoop$ hdfs dfs -ls /user/hadoopuser/testHDFS/
Found 1 items
-rw-r--r--   2 hadoopuser hadoopgroup       1361 2019-12-10 20:24 /user/hadoopuser/testHDFS/README.txt
(base) hadoopuser@master:/usr/local/hadoop$ hdfs dfs -cat /user/hadoopuser/testHDFS/README.txt | head
2019-12-10 20:25:02,024 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted =
 false
For the latest information about Hadoop, please visit our website at:

   http://hadoop.apache.org/

and our wiki, at:

   http://wiki.apache.org/hadoop/

This distribution includes cryptographic software.  The country in
(base) hadoopuser@master:/usr/local/hadoop$ hdfs dfs -get /user/hadoopuser/testHDFS/README.txt /usr/local/hadoop/README2.txt
2019-12-10 20:29:46,280 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted =
 false
(base) hadoopuser@master:/usr/local/hadoop$ ls -al R*
-rw-r--r-- 1 hadoopuser hadoopgroup 1361 Δεκ  10 20:29 README2.txt
-rw-r--r-- 1 hadoopuser hadoopgroup 1361 Σεπ  10 17:35 README.txt
(base) hadoopuser@master:/usr/local/hadoop$ cat README2.txt | head
For the latest information about Hadoop, please visit our website at:

   http://hadoop.apache.org/

and our wiki, at:

   http://wiki.apache.org/hadoop/

This distribution includes cryptographic software.  The country in
which you currently reside may have restrictions on the import,
(base) hadoopuser@master:/usr/local/hadoop$ hdfs dfs -mkdir -p /user/hadoopuser/testHDFS/test/
(base) hadoopuser@master:/usr/local/hadoop$ hdfs dfs -mv  /user/hadoopuser/testHDFS/README.txt /user/hadoopuser/testHDFS/test/
(base) hadoopuser@master:/usr/local/hadoop$ hdfs dfs -ls /user/hadoopuser/testHDFS/
Found 1 items
drwxr-xr-x   - hadoopuser hadoopgroup          0 2019-12-10 20:31 /user/hadoopuser/testHDFS/test
(base) hadoopuser@master:/usr/local/hadoop$ hdfs dfs -ls /user/hadoopuser/testHDFS/test/
Found 1 items
-rw-r--r--   2 hadoopuser hadoopgroup       1361 2019-12-10 20:24 /user/hadoopuser/testHDFS/test/README.txt
(base) hadoopuser@master:/usr/local/hadoop$ hdfs dfs -cp /user/hadoopuser/testHDFS/test/README.txt /user/hadoopuser/testHDFS/
2019-12-10 20:36:33,813 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted =
 false
2019-12-10 20:36:33,990 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted =
 false
(base) hadoopuser@master:/usr/local/hadoop$ hdfs dfs -ls /user/hadoopuser/testHDFS/
Found 2 items
-rw-r--r--   2 hadoopuser hadoopgroup       1361 2019-12-10 20:36 /user/hadoopuser/testHDFS/README.txt
drwxr-xr-x   - hadoopuser hadoopgroup          0 2019-12-10 20:31 /user/hadoopuser/testHDFS/test
(base) hadoopuser@master:/usr/local/hadoop$ hdfs dfs -rm -R /user/hadoopuser/testHDFS/
Deleted /user/hadoopuser/testHDFS
(base) hadoopuser@master:/usr/local/hadoop$ hdfs dfs -ls /user/
Found 1 items
drwxr-xr-x   - hadoopuser hadoopgroup          0 2019-12-10 20:49 /user/hadoopuser
(base) hadoopuser@master:/usr/local/hadoop$
```

Σχήμα 6-138: Βασικές εντολές για το κατανεμημένο σύστημα αρχείων (distributed file system) HDFS – ls/mkdir/put/cat/get/mv/cp/rm -R

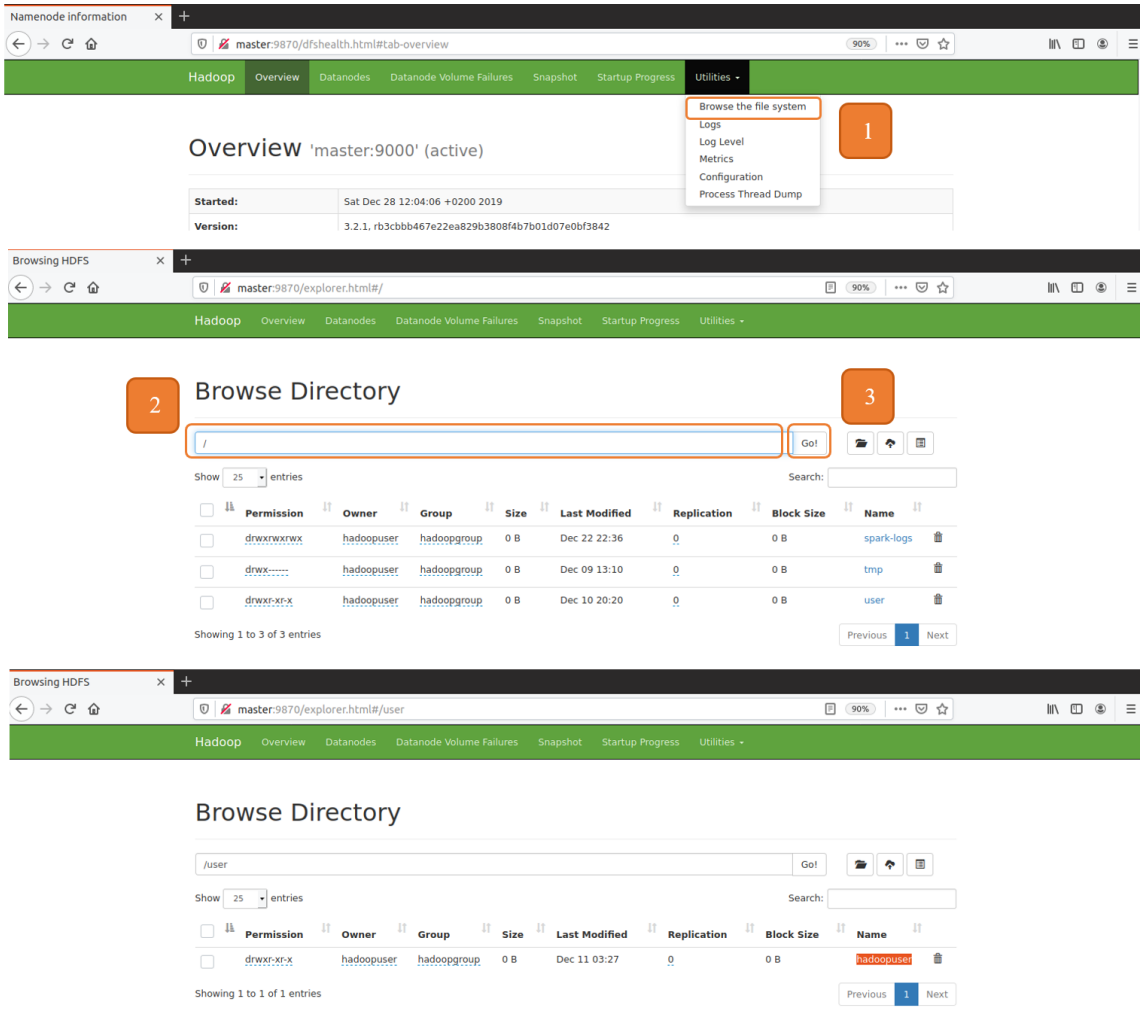
Σχήμα 6-139: Η αρχική σελίδα (σελίδα επισκόπησης) του NameNode Web UI

Η πρόσβαση σε κάποιο DataNode κόμβο γίνεται μέσω της καρτέλας Datanodes. Στο Σχήμα 6-140 φαίνεται ο διακομιστής του datanode κόμβου slave1.



Σχήμα 6-140: Στην καρτέλα datanodes πατώντας ο χρήστης πάνω στον σύνδεσμο του κόμβου slave1 (1), ανοίγει ο εσωτερικός διακομιστής ιστού του κόμβου slave1

Μέσω της διαδικτυακής διεπαφής χρήστη (Web UI) μπορεί ο χρήστης να περιηγηθεί στο σύστημα αρχείων χρησιμοποιώντας τον σύνδεσμο "*Browse the File System (Περιήγηση στο Σύστημα Αρχείων)*" που βρίσκεται στο πτυσσόμενο μενού "*Utilities (Βοηθητικά προγράμματα)*" της αρχικής σελίδας του NameNode (master, βλέπε Σχήμα 6-141).



Σχήμα 6-141: Περιήγηση στο κατανεμημένο σύστημα αρχείων HDFS μέσω του NameNode UI

Ο έλεγχος της σωστής λειτουργίας του *Hadoop YARN* θα γίνει παρακάτω μαζί με τον έλεγχο του *Spark*.

Διαμόρφωση των αρχείων spark-env.sh και spark-defaults.conf

Ακολουθεί η διαμόρφωση των αρχείων *spark-env.sh* και *spark-defaults.conf* και ο έλεγχος της συστάδας Spark με τρόπο ανάπτυξης συστάδας (cluster deploy mode) και τρόπο ανάπτυξης πελάτη (client deploy mode) (ο τρόπος ανάπτυξης του Spark καθορίζει τη θέση όπου εκτελείται ο οδηγός (driver) στο περιβάλλον ανάπτυξης). Υπάρχουν δύο κύριες κατηγορίες επιλογών διαμόρφωσης του Apache Spark: Οι ιδιότητες Spark (Spark properties) και οι μεταβλητές περιβάλλοντος (environment variables). Οι ιδιότητες Spark χρησιμοποιούνται για την ρύθμιση των διαμορφώσεων της εφαρμογής Spark και μπορούν να διαμορφωθούν ξεχωριστά για κάθε εφαρμογή [33] [40]. Οι μεταβλητές περιβάλλοντος μπορούν να

[33] *Μπορεί να ορίσει ο χρήστης τις ιδιότητες αυτές με τους παρακάτω τρόπους, με σειρά προτεραιότητας από την υψηλότερη στη χαμηλότερη: - Άμεσα σε ένα SparkConf περνώντας το στο SparkContext. - Στο χρόνο εκτέλεσης (run time), ως επιλογές γραμμής εντολών στο κέλυφος(shell) pyspark και στην spark-submit. -Στο αρχείο ιδιοτήτων [85].*

χρησιμοποιηθούν για τη ρύθμιση των διαμορφώσεων ανά μηχανή, μέσω του script *spark-env.sh* σε κάθε κόμβο. Αναλυτικές πληροφορίες για τους τρόπους ανάπτυξης του Spark παρέχονται στην ομώνυμη ενότητα "Τρόποι ανάπτυξης του Spark".

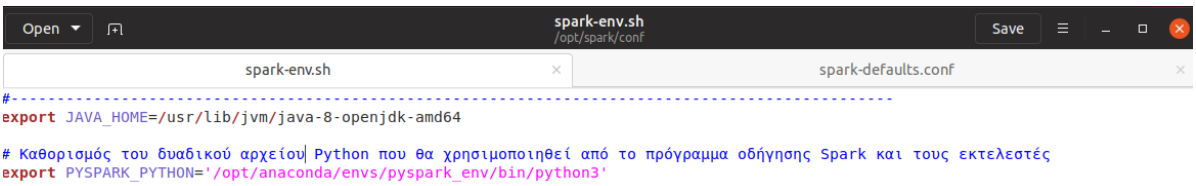
```
#Κόμβος master
cd /opt/spark/conf
# Αντιγραφή του προτύπου spark-env.sh.template στο αρχείο spark-env.sh
cp spark-env.sh.template spark-env.sh
# Άνοιγμα του αρχείου με το gedit
sudo gedit spark-env.sh
```

Κατόπιν πληκτρολογεί στο αρχείο spark-env.sh (βλέπε Σχήμα 6-143):

```
# Μέσα στο αρχείο spark-env.sh
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
# Ορίζουμε την python του περιβάλλοντος pyspark_env ως την python που θα χρησιμοποιεί το pyspark
export PYSPARK_PYTHON='/opt/anaconda/envs/pyspark_env/bin/python3'
```

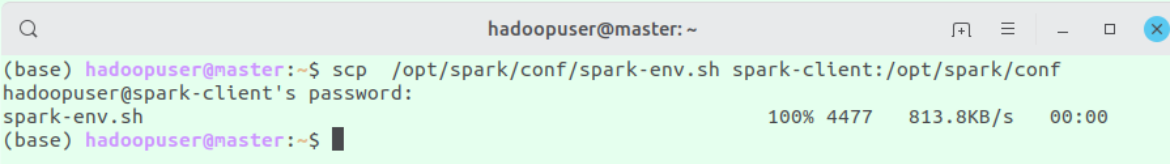
```
(base) hadoopuser@master:~$ cd /opt/spark/conf
(base) hadoopuser@master:/opt/spark/conf$ ls -al
total 44
drwxr-xr-x  2 hadoopuser hadoopgroup 4096 Δεκ   6 11:22 .
drwxr-xr-x 13 hadoopuser hadoopgroup 4096 Μαϊ   1  2019 ..
-rw-r--r--  1 hadoopuser hadoopgroup  996 Μαϊ   1  2019 docker.properties.template
-rw-r--r--  1 hadoopuser hadoopgroup 1105 Μαϊ   1  2019 fairscheduler.xml.template
-rw-r--r--  1 hadoopuser hadoopgroup 2025 Μαϊ   1  2019 log4j.properties.template
-rw-r--r--  1 hadoopuser hadoopgroup 7801 Μαϊ   1  2019 metrics.properties.template
-rw-r--r--  1 hadoopuser hadoopgroup  865 Μαϊ   1  2019 slaves.template
-rw-r--r--  1 hadoopuser hadoopgroup 1292 Μαϊ   1  2019 spark-defaults.conf.template
-rwxr-xr-x  1 hadoopuser hadoopgroup 4221 Μαϊ   1  2019 spark-env.sh.template
(base) hadoopuser@master:/opt/spark/conf$ cp spark-env.sh.template spark-env.sh
(base) hadoopuser@master:/opt/spark/conf$ ls -al
total 52
drwxr-xr-x  2 hadoopuser hadoopgroup 4096 Δεκ   6 11:23 .
drwxr-xr-x 13 hadoopuser hadoopgroup 4096 Μαϊ   1  2019 ..
-rw-r--r--  1 hadoopuser hadoopgroup  996 Μαϊ   1  2019 docker.properties.template
-rw-r--r--  1 hadoopuser hadoopgroup 1105 Μαϊ   1  2019 fairscheduler.xml.template
-rw-r--r--  1 hadoopuser hadoopgroup 2025 Μαϊ   1  2019 log4j.properties.template
-rw-r--r--  1 hadoopuser hadoopgroup 7801 Μαϊ   1  2019 metrics.properties.template
-rw-r--r--  1 hadoopuser hadoopgroup  865 Μαϊ   1  2019 slaves.template
-rw-r--r--  1 hadoopuser hadoopgroup 1292 Μαϊ   1  2019 spark-defaults.conf.template
-rwxr-xr-x  1 hadoopuser hadoopgroup 4221 Δεκ   6 11:23 spark-env.sh
-rwxr-xr-x  1 hadoopuser hadoopgroup 4221 Μαϊ   1  2019 spark-env.sh.template
(base) hadoopuser@master:/opt/spark/conf$ sudo gedit spark-env.sh
[sudo] password for hadoopuser:
```

Σχήμα 6-142: Αντιγραφή του spark-env.sh από το πρότυπο spark-env.sh.template και άνοιγμα του με το gedit



Σχήμα 6-143: Ενημέρωση αρχείου spark-env.sh

```
# Μεταφόρτωση αρχείου spark-env.sh από τον master στον spark-client
scp /opt/spark/conf/spark-env.sh spark-client:/opt/spark/conf
```



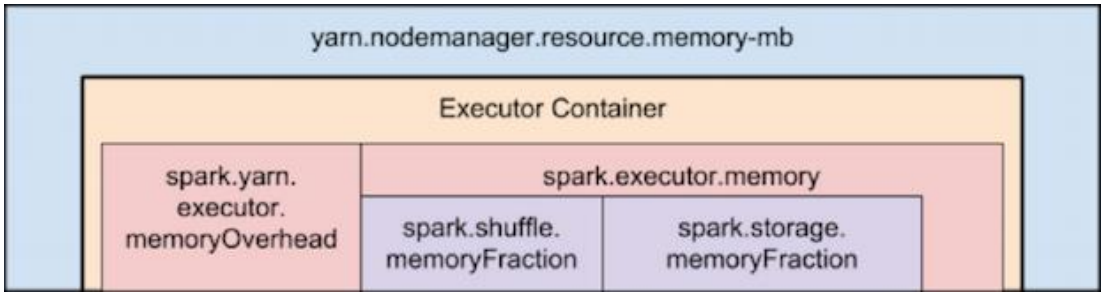
Σχήμα 6-144: Μεταφόρτωση αρχείου spark-env.sh από τον master στον spark-client

```
#Κόμβος master
# Αντιγραφή του προτύπου spark-defaults.conf.template στο αρχείο spark-defaults.conf
cp spark-defaults.conf.template spark-defaults.conf
ls -al
# Άνοιγμα του αρχείου με το gedit
sudo gedit spark-defaults.conf
```

Κατόπιν πληκτρολογεί στο αρχείο spark-defaults.conf (βλέπε Σχήμα 6-147):

```
#Κόμβος master
spark.master            yarn

# Ποσότητα μνήμης που θα χρησιμοποιηθεί για το YARN Application Master σε cluster
mode – Προκαθορισμένη τιμή 1g.
# # Μορφοποιούνται ως συμβολοσειρές μνήμης JVM με κατάληξη μονάδας μεγέθους
# ("k" -> kibibyte, "m" -> mebibyte, "g" -> gibibyte, "t" -> tebibyte)
# https://en.wikipedia.org/wiki/Gibibyte
spark.driver.memory[34]    768m
# Αριθμός πυρήνων που θα χρησιμοποιηθούν για το κύριο πρόγραμμα εφαρμογής YARN σε
# client mode - Προκαθορισμένη τιμή: 1
#spark.driver.cores     1

# Ποσότητα μνήμης που θα χρησιμοποιηθεί για το YARN Application Master σε client
mode –
# Προκαθορισμένη τιμή 512m
spark.yarn.am.memory     512m
# Αριθμός πυρήνων που πρέπει να χρησιμοποιηθούν για το κύριο πρόγραμμα εφαρμογής
YARN
# σε client mode.
# spark.yarn.am.cores   1

# Η πλήρης μνήμη που ζητείται από το YARN ανά εκτελεστή είναι =
# spark.executor.memory + spark.yarn.executor.memoryOverhead [39]
# όπου spark.yarn.executor.memoryOverhead = max(384 MiB, 0.10 *
spark.executor.memory)
# εδώ (3072m/2 - 384m), όπου 2: ο αριθμός εκτελεστών ανά εργάτη (worker)
spark.executor.memory   1152m
# Αριθμός εκτελεστών(executors) ανά εργάτη (worker) - Προκαθορισμένη τιμή: 2.
spark.executor.instances 2
# Ο αριθμός των πυρήνων ανά εκτελεστή - Προκαθορισμένη τιμή σε YARN mode: 1
spark.executor.cores     1
```

[34] *Ποσότητα μνήμης που θα χρησιμοποιηθεί για τη διεργασία του προγράμματος οδήγησης (driver process). Στη περίπτωση ανάπτυξης μιας εφαρμογής Spark σε client mode δε θα πρέπει να καθοριστεί μέσω της sparkConf μέσα στην εφαρμογή αλλά είτε δυναμικά μέσω της επιλογής --driver-memory στις εντολές spark-submit και pyspark (π.χ. ./bin/spark-submit --driver-memory 5g) είτε όπως εδώ με το εξορισμού αρχείο ιδιοτήτων spark-defaults.conf [80]. Οι ρυθμίσεις (configurations) μέσω της spark-submit έχουν μεγαλύτερη προτεραιότητα έναντι των ρυθμίσεων εντός της εφαρμογής μέσω της sparkConf, οι οποίες έχουν μεγαλύτερη προτεραιότητα έναντι των ρυθμίσεων σε αρχεία διαμόρφωσης (config files, π.χ. spark-defaults.conf).*



Πηγή:[39]

Σχήμα 6-145: Κατανομή μνήμης σε έναν κόμβο μιας συστάδας Spark με ένα executor container και διαχειριστή πόρων (RM) τον Hadoop YARN.

```
(base) hadoopuser@master:/opt/spark/conf$ cp spark-defaults.conf.template spark-defaults.conf
(base) hadoopuser@master:/opt/spark/conf$ ls -al
total 60
drwxr-xr-x  2 hadoopuser hadoopgroup 4096 Δεκ   8 23:54 .
drwxr-xr-x 13 hadoopuser hadoopgroup 4096 Μαϊ   1  2019 ..
-rw-r--r--  1 hadoopuser hadoopgroup  996 Μαϊ   1  2019 docker.properties.template
-rw-r--r--  1 hadoopuser hadoopgroup 1105 Μαϊ   1  2019 fairscheduler.xml.template
-rw-r--r--  1 hadoopuser hadoopgroup 2025 Μαϊ   1  2019 log4j.properties.template
-rw-r--r--  1 hadoopuser hadoopgroup 7801 Μαϊ   1  2019 metrics.properties.template
-rw-r--r--  1 hadoopuser hadoopgroup  883 Δεκ   8 22:57 slaves
-rw-r--r--  1 hadoopuser hadoopgroup  865 Μαϊ   1  2019 slaves.template
-rw-r--r--  1 hadoopuser hadoopgroup 1292 Δεκ   8 23:54 spark-defaults.conf
-rw-r--r--  1 hadoopuser hadoopgroup 1292 Μαϊ   1  2019 spark-defaults.conf.template
-rwxr-xr-x  1 hadoopuser hadoopgroup 4477 Δεκ   6 11:38 spark-env.sh
-rwxr-xr-x  1 hadoopuser hadoopgroup 4221 Μαϊ   1  2019 spark-env.sh.template
(base) hadoopuser@master:/opt/spark/conf$ sudo gedit spark-defaults.conf
[sudo] password for hadoopuser:
(base) hadoopuser@master:/opt/spark/conf$
```

Σχήμα 6-146: Αντιγραφή του spark-defaults.conf από το πρότυπο spark-defaults.conf.template και άνοιγμα του με το gedit

Σχήμα 6-147: Ενημέρωση του αρχείου spark-defaults.conf[35]

```
# Μεταφόρτωση αρχείου spark-defaults.conf από τον master στον spark-client
scp /opt/spark/conf/spark-defaults.conf spark-client:/opt/spark/conf/
```

[35] *Οι τιμές τροποποιήθηκαν όπως στην προηγούμενη σελίδα για να εκμεταλλεύεται ο χρήστης καλύτερα την διαθέσιμη μνήμη των ΕΜ. Εδώ οι συγκεκριμένες τιμές χρησιμοποιήθηκαν μόνο για δοκιμή του παρακάτω κώδικα.*

```
(base) hadoopuser@master:/opt/spark/conf$ scp /opt/spark/conf/spark-defaults.conf spark-client
:/opt/spark/conf/
The authenticity of host 'spark-client (192.168.1.110)' can't be established.
ECDSA key fingerprint is SHA256:cMbbxHuZV4BO6Qa7rZASeOW/F8dGS5C34CvYxPP26hM.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'spark-client,192.168.1.110' (ECDSA) to the list of known hosts.
hadoopuser@spark-client's password:
spark-defaults.conf                                   100% 1413   113.1KB/s   00:00
(base) hadoopuser@master:/opt/spark/conf$
```

Σχήμα 6-148: Μεταφόρτωση του αρχείου spark-defaults.conf από τον master στον spark-client

Έλεγχος της εκκίνησης εφαρμογών του Spark σε cluster mode σε συστάδα Hadoop YARN

Όπως ήδη έχει αναφερθεί σε ***cluster mode*** ο οδηγός Spark τρέχει μέσα σε μια διεργασία *application master*, την οποία διαχειρίζεται το YARN στη συστάδα ενώ ο πελάτης (client) μπορεί να απομακρυνθεί μετά την εκκίνηση της εφαρμογής. Ακολουθεί έλεγχος της εκκίνησης εφαρμογών του *Spark* σε *cluster mode*, εκτελώντας με το script *spark-submit* το παράδειγμα *pi.py* (βλέπε Σχήμα 6-149), για τον υπολογισμό μιας προσέγγισης του *π* μέσω της μεθόδου *Monte Carlo*, το οποίο παρέχεται με το *Spark* στον κατάλογο */opt/spark/examples/src/main/python/* (βλέπε Σχήμα 6-150):

```
# spark-client
# Μετά την εκτέλεση των scripts: start-dfs.sh και start-yarn.sh για την εκκίνηση του
# κατανεμημένου συστήματος αρχείων hadoop HDFS και του διαχειριστή πόρων YARN αντίστοιχα.
spark-submit --master yarn --deploy-mode cluster /opt/spark/examples/src/main/python/pi.py
100
```

Διαδικτυακή διεπαφή χρήστη του Διαχειριστή Πόρων YARN (YARN ResourceManager Web UI)

Με την προεπιλεγμένη διαμόρφωση, η αρχική σελίδα της διαδικτυακής διεπαφής χρήστη YARN ResourceManager (Web UI) βρίσκεται στην διεύθυνση *http://<ResourceManager-hostname>:*[36]*8088*. Μέσω αυτής ο χρήστης μπορεί να ενημερωθεί για την κατάσταση μιας εργασίας (job) προβάλλοντας πληροφορίες του διακομιστή ιστορικού εργασιών (βλέπε Σχήμα 6-151).

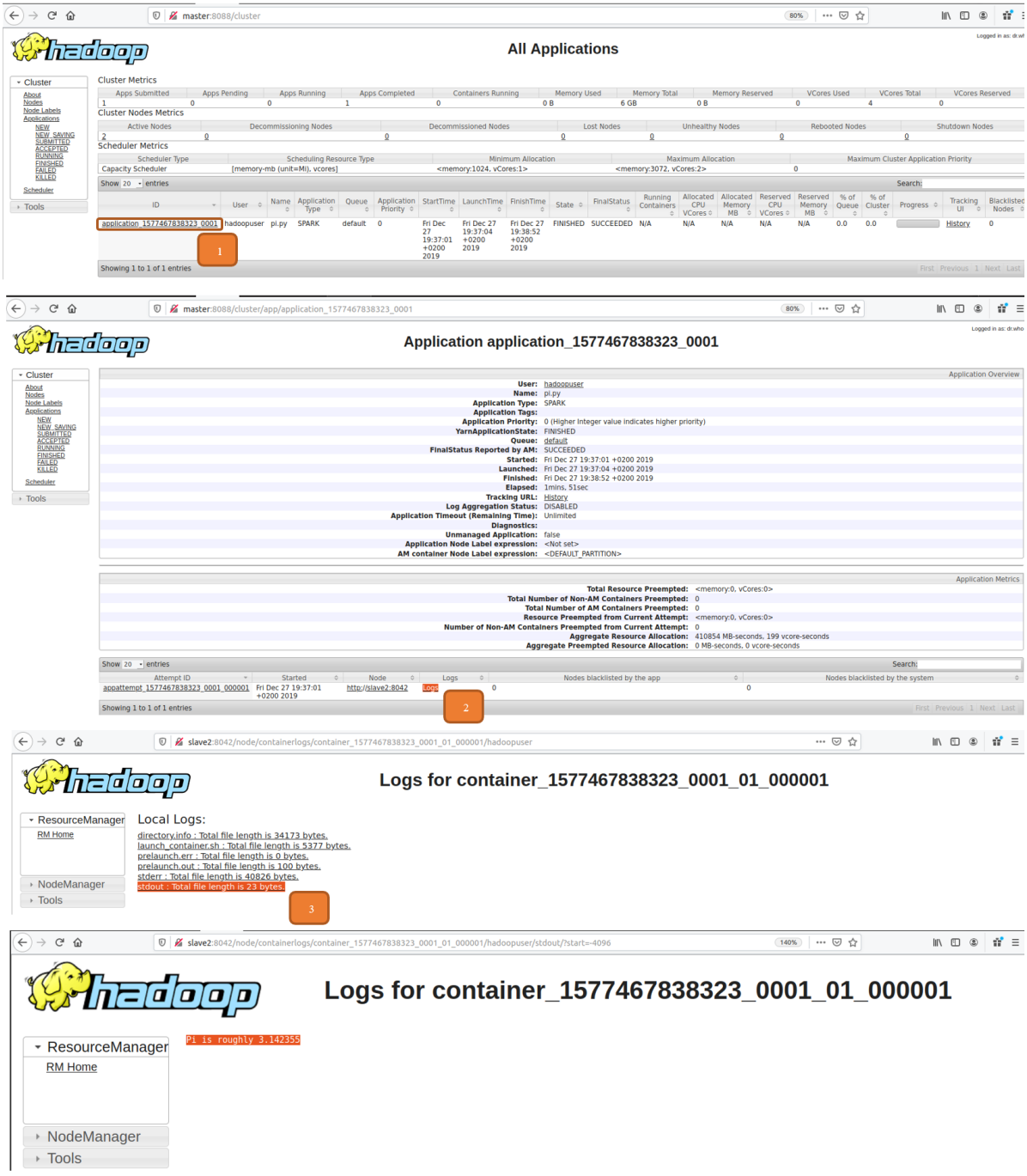
[36] master

```python
from __future__ import print_function

import sys
from random import random
from operator import add

from pyspark.sql import SparkSession


if __name__ == "__main__":
    """
        Usage: pi [partitions]
    """
    spark = SparkSession\
        .builder\
        .appName("PythonPi")\
        .getOrCreate()

    partitions = int(sys.argv[1]) if len(sys.argv) > 1 else 2
    n = 100000 * partitions

    def f(_):
        x = random() * 2 - 1
        y = random() * 2 - 1
        return 1 if x ** 2 + y ** 2 <= 1 else 0

    count = spark.sparkContext.parallelize(range(1, n + 1), partitions).map(f).reduce(add)
    print("Pi is roughly %f" % (4.0 * count / n))

    spark.stop()
```

Σχήμα 6-149: Το παράδειγμα pi.py, για τον υπολογισμό μιας προσέγγισης του π μέσω της μεθόδου Monte Carlo (παρέχεται με το Spark)

```
hadoopuser@spark-client: ~
(base) hadoopuser@spark-client:~$ spark-submit -v --master yarn --deploy-mode cluster /opt/spark/examples/src/main/python/pi.py 100
Using properties file: /opt/spark/conf/spark-defaults.conf
Adding default property: spark.master=yarn
Adding default property: spark.executor.memory=1g
Adding default property: spark.yarn.am.memory=512m
Adding default property: spark.driver.memory=1g
Parsed arguments:
  master                  yarn
  deployMode              cluster
  executorMemory          1g
  executorCores           null
  totalExecutorCores      null
  propertiesFile          /opt/spark/conf/spark-defaults.conf
  driverMemory            1g
  driverCores             null
  driverExtraClassPath    null
  driverExtraLibraryPath  null
  driverExtraJavaOptions  null
  supervise               false
  queue                   null
  numExecutors            null
  files                   null
  pyFiles                 null
  archives                null
  mainClass               null
  primaryResource         file:/opt/spark/examples/src/main/python/pi.py
  name                    pi.py
  childArgs               [100]
  jars                    null
  packages                null
  packagesExclusions      null
  repositories            null
  verbose                 true
19/12/27 19:30:59 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.
19/12/27 19:31:08 INFO Client: Uploading resource file:/tmp/spark-7425efc0-09ce-46c4-94e0-b7488780045d/__spark_libs__1636567447805816773.zip -> hdfs://master:90
00/user/hadoopuser/.sparkStaging/application_1577467838323_0001/__spark_libs__1636567447805816773.zip
19/12/27 19:36:53 INFO Client: Uploading resource file:/opt/spark/examples/src/main/python/pi.py -> hdfs://master:9000/user/hadoopuser/.sparkStaging/application
_1577467838323_0001/pi.py
19/12/27 19:36:53 INFO Client: Uploading resource file:/opt/spark/python/lib/pyspark.zip -> hdfs://master:9000/user/hadoopuser/.sparkStaging/application_1577467
838323_0001/pyspark.zip
19/12/27 19:36:55 INFO Client: Uploading resource file:/opt/spark/python/lib/py4j-0.10.7-src.zip -> hdfs://master:9000/user/hadoopuser/.sparkStaging/application
_1577467838323_0001/py4j-0.10.7-src.zip
19/12/27 19:36:56 INFO Client: Uploading resource file:/tmp/spark-7425efc0-09ce-46c4-94e0-b7488780045d/__spark_conf__3011903787437375684.zip -> hdfs://master:90
00/user/hadoopuser/.sparkStaging/application_1577467838323_0001/__spark_conf__.zip
19/12/27 19:36:57 INFO SecurityManager: Changing view acls to: hadoopuser
19/12/27 19:36:57 INFO SecurityManager: Changing modify acls to: hadoopuser
19/12/27 19:38:53 INFO Client:
         client token: N/A
         diagnostics: N/A
         ApplicationMaster host: slave2
         ApplicationMaster RPC port: 37315
         queue: default
         start time: 1577468221459
         final status: SUCCEEDED
         tracking URL: http://master:8088/proxy/application_1577467838323_0001/
         user: hadoopuser
19/12/27 19:38:53 INFO ShutdownHookManager: Shutdown hook called
19/12/27 19:38:53 INFO ShutdownHookManager: Deleting directory /tmp/spark-7425efc0-09ce-46c4-94e0-b7488780045d
19/12/27 19:38:53 INFO ShutdownHookManager: Deleting directory /tmp/spark-bac2cfa3-5d98-4778-aa18-303b63f4a4bc
(base) hadoopuser@spark-client:~$
```

Σχήμα 6-150: Εκτέλεση του παραδείγματος pi.py με την εντολή spark-submit σε cluster mode. Η επιλογή γραμμής εντολών (command line option) -v / --verbose της spark-submit παρέχει επιπλέον πληροφορίες σχετικά με την εκτέλεση της εφαρμογής/του προγράμματος Spark στη συστάδα. Όπως παρατηρεί κανείς το παράδειγμα εκτελέστηκε κανονικά (status: succeeded). Το αποτέλεσμα που προέκυψε μπορεί να το δει ο χρήστης μέσω του hadoop ResourceManager (RM) UI (βλέπε Σχήμα 6-151).

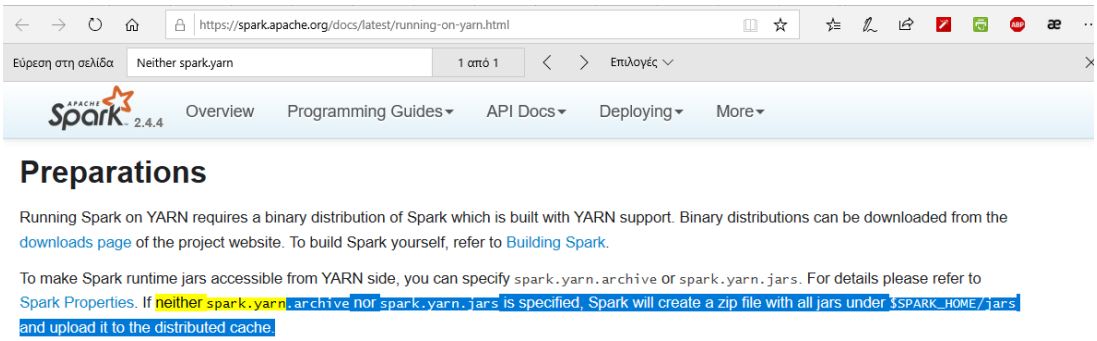


Σχήμα 6-151: Το ResourceManager (RM) Web UI του hadoop YARN– Για να δει ο χρήστης το αποτέλεσμα του παραδείγματος pi.py θα πρέπει να κάνει κλικ στο application ID (1) της εφαρμογής, στη συνέχεια στο Logs (2) και κατόπιν στο stdout:..(3)

Παρατήρηση:

- Σύμφωνα με την επίσημη ιστοσελίδα του Spark [36] (βλέπε Σχήμα 6-152) το μήνυμα "*Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME*", του Σχήματος 6-150, υποδηλώνει ότι επειδή δεν έχει καθοριστεί

η μεταβλητή *spark.yarn.jars* αλλά ούτε και η μεταβλητή *spark.yarn.archive* το Spark θα δημιουργήσει ένα αρχείο *zip* στον κατάλογο *$SPARK_HOME/jars* με όλα τα αρχεία *jars* του καταλόγου και θα τα μεταφορτώσει στην κατανεμημένη cache καθότι αυτά τα αρχεία *jars* είναι απαραίτητα για την επιτυχή λειτουργία του *Spark* στη συστάδα *YARN*.



Σχήμα 6-152: Προετοιμασίες πριν το τρέξιμο του Spark σε μια συστάδα YARN

Αυτό όπως κατανοεί κανείς καθυστερεί την εκτέλεση του *spark-submit* κάθε φορά που τρέχει η εντολή. Το ίδιο μήνυμα εμφανίζεται και όταν εκτελείται η *spark-submit* σε *client mode*. Ένας τρόπος για να αποφύγει κανείς αυτή την καθυστέρηση είναι να δημιουργήσει ένα *jar* αρχείο με όλα τα *jar* αρχεία (το αποτέλεσμα της εντολής *ls jars | wc -l* είναι 226 αρχεία, όπως φαίνεται στο Σχήμα 6-153) που υπάρχουν στον κατάλογο *$SPARK_HOME/jars* και να το μεταφορτώσει σε έναν κατάλογο στο κατανεμημένο σύστημα αρχείων *HDFS*. Θα πρέπει το αρχείο αυτό να έχει συντελεστή αναπαραγωγής ίσο με τον αριθμό των *NodeManagers* (στην συστάδα που δημιούργησε ο γράφων στη διπλωματική αυτή, ο συντελεστής αναπαραγωγής=2). Τα παραπάνω βήματα συνοψίζονται στις ακόλουθες εντολές που εκτελούνται στη γραμμή εντολών του master (βλέπε Σχήμα 6-153):

```
cd /opt/spark
# Δημιουργία ενός καταλόγου (jar-lib)  όπου θα αποθηκεύσει ο χρήστης το αρχείο
jar
mkdir -p /opt/spark/jar-lib
ls -al
# Καταμέτρηση των αρχείων μέσα στον κατάλογο jars
ls jars | wc -l
cd jar-lib
# Δημιουργία του spark-libs.jar, ενός αρχείου jar με όλα τα αρχεία που υπάρχουν
# στον κατάλογο $SPARK_HOME/jars
jar cv0f spark-libs.jar -C $SPARK_HOME/jars/  .
# Δημιουργία του καταλόγου spark-jars στο κατανεμημένο σύστημα HDFS
hdfs dfs -mkdir /user/hadoopuser/spark-jars
hdfs dfs -ls /user/hadoopuser
# Μεταφόρτωση του spark-libs.jar στον κατάλογο spark-jars
hdfs dfs -put spark-libs.jar /user/hadoopuser/spark-jars/
# Θα πρέπει το αρχείο spark-libs.jar να έχει συντελεστή αναπαραγωγής ίσο με τον
# αριθμό των NodeManagers (για τη συστάδα που έχει δημιουργηθεί, ο συντελεστής
# αναπαραγωγής=2)
```

```
hdfs dfs -setrep -w 2 hdfs:///user/hadoopuser/spark-jars/spark-libs.jar
hdfs dfs -ls /user/hadoopuser/spark-jars/spark-libs.jar
```

Ακολούθως θα πρέπει να ανοίξει ο χρήστης το αρχείο *spark-defaults.conf* στον κόμβο spark-client [37](βλέπε Σχήμα 6-154):

```
sudo gedit /opt/spark/conf/spark-defaults.conf
```

και να το ενημερώσει πληκτρολογώντας τα παρακάτω (βλέπε Σχήμα 6-154):

```
spark.yarn.archive    hdfs:///user/hadoopuser/spark-jars/spark-libs.jar
```

```
(base) hadoopuser@master:~$ cd /opt/spark
(base) hadoopuser@master:/opt/spark$ mkdir -p /opt/spark/jar-lib
(base) hadoopuser@master:/opt/spark$ ls -al
total 140
drwxr-xr-x 14 hadoopuser hadoopgroup  4096 Δεκ  11 03:15 .
drwxr-xr-x  5 root       root         4096 Δεκ   5 01:55 ..
drwxr-xr-x  2 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 bin
drwxr-xr-x  2 hadoopuser hadoopgroup  4096 Δεκ   9 20:01 conf
drwxr-xr-x  5 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 data
drwxr-xr-x  4 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 examples
drwxr-xr-x  2 hadoopuser hadoopgroup  4096 Δεκ  11 03:15 jar-lib
drwxr-xr-x  2 hadoopuser hadoopgroup 12288 Μαϊ   1  2019 jars
drwxr-xr-x  4 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 kubernetes
-rw-r--r--  1 hadoopuser hadoopgroup 21316 Μαϊ   1  2019 LICENSE
drwxr-xr-x  2 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 licenses
-rw-r--r--  1 hadoopuser hadoopgroup 42919 Μαϊ   1  2019 NOTICE
drwxr-xr-x  9 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 python
drwxr-xr-x  3 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 R
-rw-r--r--  1 hadoopuser hadoopgroup  3952 Μαϊ   1  2019 README.md
-rw-r--r--  1 hadoopuser hadoopgroup   164 Μαϊ   1  2019 RELEASE
drwxr-xr-x  2 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 sbin
drwxr-xr-x  2 hadoopuser hadoopgroup  4096 Μαϊ   1  2019 yarn
(base) hadoopuser@master:/opt/spark$ ls jars | wc -l
226
(base) hadoopuser@master:/opt/spark$ cd jar-lib
(base) hadoopuser@master:/opt/spark/jar-lib$ jar cv0f spark-libs.jar -C $SPARK_HOME/jars/  .
added manifest
adding: spark-launcher_2.11-2.4.3.jar(in = 75939) (out= 75939)(stored 0%)
adding: jersey-common-2.22.2.jar(in = 698375) (out= 698375)(stored 0%)
adding: spark-network-common_2.11-2.4.3.jar(in = 2392842) (out= 2392842)(stored 0%)
adding: xbean-asm6-shaded-4.8.jar(in = 282930) (out= 282930)(stored 0%)
                                    ...
adding: zstd-jni-1.3.2-2.jar(in = 2333186) (out= 2333186)(stored 0%)
adding: stax-api-1.0-2.jar(in = 23346) (out= 23346)(stored 0%)
adding: antlr-2.7.7.jar(in = 445288) (out= 445288)(stored 0%)
(base) hadoopuser@master:/opt/spark/jar-lib$
(base) hadoopuser@master:/opt/spark/jar-lib$ hdfs dfs -mkdir /user/hadoopuser/spark-jars
(base) hadoopuser@master:/opt/spark/jar-lib$ hdfs dfs -ls /user/hadoopuser
Found 2 items
drwxr-xr-x   - hadoopuser hadoopgroup          0 2019-12-11 01:47 /user/hadoopuser/.sparkStaging
drwxr-xr-x   - hadoopuser hadoopgroup          0 2019-12-11 03:27 /user/hadoopuser/spark-jars
(base) hadoopuser@master:/opt/spark/jar-lib$
(base) hadoopuser@master:/opt/spark/jar-lib$ hdfs dfs -put spark-libs.jar /user/hadoopuser/spark-jars
2019-12-11 03:29:27,896 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTruste
d = false, remoteHostTrusted = false
(base) hadoopuser@master:/opt/spark/jar-lib$ hdfs dfs -setrep -w 2 hdfs:///user/hadoopuser/spark-jars/
spark-libs.jar
Replication 2 set: hdfs:///user/hadoopuser/spark-jars/spark-libs.jar
Waiting for hdfs:///user/hadoopuser/spark-jars/spark-libs.jar ... done
(base) hadoopuser@master:/opt/spark/jar-lib$ hdfs dfs -ls /user/hadoopuser/spark-jars/spark-libs.jar
-rw-r--r--   2 hadoopuser hadoopgroup  238588915 2019-12-11 03:29 /user/hadoopuser/spark-jars/spark-li
bs.jar
(base) hadoopuser@master:/opt/spark/jar-lib$
```

Σχήμα 6-153: Στο σχήμα αυτό και στο επόμενο περιγράφονται τα βήματα που πρέπει να γίνουν για να εξαλειφθεί το μήνυμα "Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME" και να βελτιωθεί η ταχύτητα εκτέλεσης της εντολής spark-submit τόσο σε cluster όσο και σε client mode.

[37] *Αν θέλει ο χρήσης να τρέχει εφαρμογές spark και από τον master όπως στο παράδειγμα, τότε θα πρέπει να ενημερώσει ομοίως και το αρχείο spark-defaults.conf του master προκειμένου να αποφύγει την επιπλέον καθυστέρηση. Για να τρέξει την εφαρμογή pi.py από τον spark-client (που είναι ο συνήθως ο κανόνας) σε* ***cluster mode****: spark-submit --master yarn --deploy-mode* ***cluster*** */opt/spark/examples/src/main/python/pi.py 100*

```
(base) hadoopuser@spark-client:~$ sudo gedit /opt/spark/conf
[sudo] password for hadoopuser:
(base) hadoopuser@spark-client:~$ cd /opt/spark/conf
(base) hadoopuser@spark-client:/opt/spark/conf$ sudo gedit spark-defaults.conf
```

Open | spark-defaults.conf /opt/spark/conf | Save

```
54 # Θα πρέπει το αρχείο spark-libs.jar να έχει συντελεστή αναπαραγωγής ίσο με τον
55 # αριθμό των NodeManagers (για τη συστάδα που έχει δημιουργηθεί, ο συντελεστής
56 # αναπαραγωγής=2)
57 # # Στον spark-client και αν θέλει ο χρήσης να τρέχει εφαρμογές spark και από τον master,
58 # τότε θα πρέπει να ενημερώσει ομοίως και το αρχείο spark-defaults.conf του master
59 spark.yarn.archive hdfs:///user/hadoopuser/spark-jars/spark-libs.jar
60
61
```

Σχήμα 6-154: Στο σχήμα αυτό και στο προηγούμενο περιγράφονται τα βήματα που πρέπει να γίνουν για να εξαλειφθεί το μήνυμα "Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME" και να βελτιωθεί η ταχύτητα εκτέλεσης της εντολής spark-submit τόσο σε cluster όσο και σε client mode.

Έλεγχος της εκκίνησης εφαρμογών του Spark σε client mode σε συστάδα με διαχειριστή πόρων το Hadoop YARN

Ακολουθεί έλεγχος της εκκίνησης εφαρμογών του *Spark* σε ***client mode***, εκτελώντας με το script *spark-submit* το παράδειγμα *pi.py* (βλέπε Σχήμα 6-156) στην ΕΜ **spark-client**:

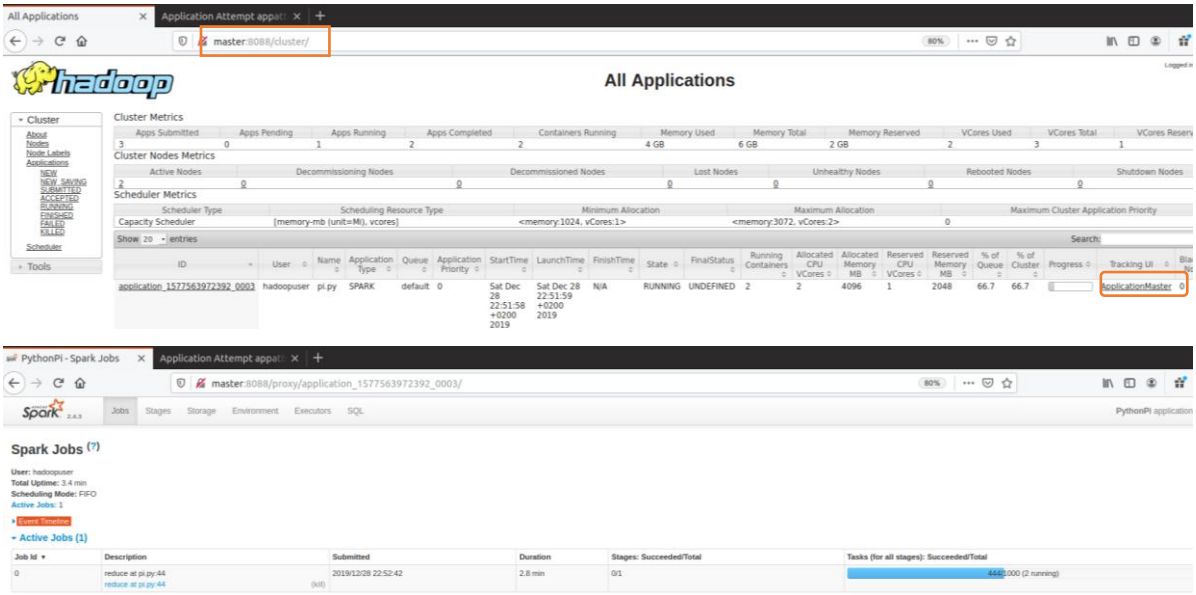
```
spark-submit --master yarn --deploy-mode client /opt/spark/examples/src/main/python/pi.py
100
```

```
(base) hadoopuser@spark-client:/opt/spark$ spark-submit --master yarn --deploy-mode client /opt/spark/examples/src/main/python/pi.py 100
2019-12-16 14:39:14,050 INFO spark.SparkContext: Running Spark version 2.4.3
2019-12-16 14:39:14,154 INFO spark.SparkContext: Submitted application: PythonPi
2019-12-16 14:39:14,362 INFO spark.SecurityManager: Changing view acls to: hadoopuser
2019-12-16 14:39:14,363 INFO spark.SecurityManager: Changing modify acls to: hadoopuser
2019-12-16 14:39:14,363 INFO spark.SecurityManager: Changing view acls groups to:
2019-12-16 14:39:14,364 INFO spark.SecurityManager: Changing modify acls groups to:
2019-12-16 14:39:14,364 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users  with view permissions
2019-12-16 14:39:22,236 INFO yarn.Client: Source and destination file systems are the same. Not copying hdfs://master:9000/user/hadoopuser/spark-jars/spark-libs.jar
2019-12-16 14:39:22,601 INFO yarn.Client: Uploading resource file:/opt/spark/python/lib/pyspark.zip -> hdfs://master:9000/user/hadoopuser/.sparkStaging/application_1576498538832_0001/pyspark.zip
2019-12-16 14:39:30,529 INFO yarn.Client: Uploading resource file:/opt/spark/python/lib/py4j-0.10.7-src.zip -> hdfs://master:9000/user/hadoopuser/.sparkStaging/application_1576498538832_0001/py4j-0.10.7-src.zip
2019-12-16 14:39:31,321 INFO yarn.Client: Uploading resource file:/tmp/spark-ac25d9d9-18a5-4d4c-ba68-b249573ec201/__spark_conf__262228484478
9161948.zip -> hdfs://master:9000/user/hadoopuser/.sparkStaging/application_1576498538832_0001/__spark_conf__.zip
2019-12-16 14:39:34,380 INFO spark.SecurityManager: Changing view acls to: hadoopuser
Pi is roughly 3.140406
2019-12-16 14:40:53,524 INFO server.AbstractConnector: Stopped Spark@70fc4e1c{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2019-12-16 14:40:53,539 INFO ui.SparkUI: Stopped Spark web UI at http://spark-client:4040
2019-12-16 14:40:53,556 INFO cluster.YarnClientSchedulerBackend: Interrupting monitor thread
2019-12-16 14:40:53,617 INFO cluster.YarnClientSchedulerBackend: Shutting down all executors
2019-12-16 14:40:53,626 INFO cluster.YarnSchedulerBackend$YarnDriverEndpoint: Asking each executor to shut down
2019-12-16 14:40:53,679 INFO cluster.SchedulerExtensionServices: Stopping SchedulerExtensionServices
(serviceOption=None,
 services=List(),
 started=false)
2019-12-16 14:40:53,692 INFO cluster.YarnClientSchedulerBackend: Stopped
2019-12-16 14:40:53,782 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
2019-12-16 14:40:53,834 INFO memory.MemoryStore: MemoryStore cleared
2019-12-16 14:40:53,851 INFO storage.BlockManager: BlockManager stopped
2019-12-16 14:40:53,854 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
2019-12-16 14:40:53,864 INFO scheduler.OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
2019-12-16 14:40:53,914 INFO spark.SparkContext: Successfully stopped SparkContext
2019-12-16 14:40:54,494 INFO util.ShutdownHookManager: Shutdown hook called
2019-12-16 14:40:54,495 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-ac25d9d9-18a5-4d4c-ba68-b249573ec201/pyspark-7d22c17d-b1bd-454f-84d4-693913984447
2019-12-16 14:40:54,498 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-252856c3-442e-4d1b-92e6-ec6d5dfcd8cc
2019-12-16 14:40:54,500 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-ac25d9d9-18a5-4d4c-ba68-b249573ec201
(base) hadoopuser@spark-client:/opt/spark$
```

Σχήμα 6-155: Ανάπτυξη του Spark προγράμματος pi.py στη συστάδα Hadoop YARN σε client mode. Όπως παρατηρεί ο χρήστης δεν εμφανίζεται πλέον το μήνυμα "Neither spark.yarn.jars nor spark.yarn.archive is set …" αλλά έχει αντικατασταθεί από το μήνυμα "Source and Destination file systems are the same. Not copying …" που υποδηλώνει ότι τα αρχεία jar διαβάζονται από το κατανεμημένο σύστημα αρχείων HDFS και συνεπώς εκτελείται πιο γρήγορα η εφαρμογή Spark. Μετά την ολοκλήρωση της εκτέλεσης του προγράμματος το αποτέλεσμα (Pi is roughly 3.140406) επιστρέφει στον spark-client

Διαδικτυακή διεπαφή χρήστη της εφαρμογής Spark (Spark Application Web UI) & ενεργοποίηση του Διακομιστή Ιστορικού Spark (Spark History Server) σε συστάδα Hadoop HDFS/YARN

Μια ρύθμιση που κατά τον γράφοντα αξίζει να υλοποιήσει ο χρήστης κατά την εγκατάσταση του Spark είναι η ενεργοποίηση του *Διακομιστή Ιστορικού Spark (Spark History Server)*. Με την προεπιλεγμένη διαμόρφωση του *Spark*, μπορεί να δει ο χρήστης αναλυτικές πληροφορίες για μια εφαρμογή όπως *χρονοδιάγραμμα συμβάντων (event timeline), οπτική αναπαράσταση του κατευθυνόμενου άκυκλου γράφου(DAG)* μιας *εργασίας (job), στάδια εργασίας (job stages), έργα (tasks), συνοπτικές μετρήσεις των έργων* (*summary tasks metrics*), *συγκεντρωτικές μετρήσεις ανά εκτελεστή (Aggregated metrics by executor), ονοματισμένοι συσσωρευτές* (*named accumulators*), *κατανάλωση πόρων της συστάδας Spark* κ.λπ. μόνο όμως για όσο χρονικό διάστημα τρέχει η εφαρμογή. Η πρόσβαση σ' αυτές τις πληροφορίες γίνεται μέσω του συνδέσμου "*Application Master*"[38] της εφαρμογής από την αρχική σελίδα της διεπαφής χρήστη *ResourceManager Web UI του Hadoop/YARN (http:/<ResourceManager IP[39]>:8088)*, ο οποίος ανακατευθύνει στη διεπαφή χρήστη *Spark Application Web UI* (βλέπε Σχήμα 6-156).



Σχήμα 6-156: Με την προεπιλεγμένη (default) διαμόρφωση του Spark η διεπαφή χρήστη Spark Web UI είναι διαθέσιμη μέσω του συνδέσμου "ApplicationMaster" της εφαρμογής για όσο χρονικό διάστημα τρέχει η εφαρμογή

[38] *Ο σύνδεσμος αυτός εμφανίζεται στα δεξιά της εφαρμογής που τρέχει και μόνο για το χρονικό διάστημα που η εφαρμογή τρέχει.*

[39] *Στην συγκεκριμένη περίπτωση ο RM είναι 192.168.1.100 ή το hostname της ΕΜ, δηλαδή master.*

Ο *Διακομιστής Ιστορικού Spark* επιτρέπει τον χρήστη να επανεξετάσει τις παραπάνω πληροφορίες και μετά το πέρας της εφαρμογής. Οι πληροφορίες αυτές είναι χρήσιμες για τον χρήστη στην προσπάθεια του να βελτιώσει την αποδοτικότητα του κώδικά του αλλά και για την ρύθμιση της συστάδας για *βέλτιστη απόδοση (performance tuning)*. Στην συνέχεια περιγράφεται ο τρόπος διαμόρφωσης του Spark έτσι ώστε να είναι ενεργός ο *Διακομιστής Ιστορικού Spark*.

Αρχικά θα πρέπει ο χρήστης να δημιουργήσει έναν κατάλογο στο κατανεμημένο σύστημα αρχείων *HDFS* όπου θα αποθηκεύονται οι πληροφορίες καταγραφής συμβάντων όταν τρέχει μια εφαρμογή *Spark*. Θα πρέπει ο χρήστης να δώσει πλήρη δικαιώματα σε αυτόν τον κατάλογο (βλέπε Σχήμα 6-157):

```
#Στην ενεργοποιημένη συστάδα Hadoop YARN (scripts start-dfs.sh, start-yarn.sh)
hdfs dfs -mkdir -p /spark-logs
hdfs dfs -chmod 777 /spark-logs
```

```
(base) hadoopuser@master:~$ hdfs dfs -mkdir -p /spark-logs
(base) hadoopuser@master:~$ hdfs dfs -ls /
Found 3 items
drwxr-xr-x   - hadoopuser hadoopgroup          0 2019-12-30 00:47 /spark-logs
drwx------   - hadoopuser hadoopgroup          0 2019-12-09 13:10 /tmp
drwxr-xr-x   - hadoopuser hadoopgroup          0 2019-12-10 20:20 /user
(base) hadoopuser@master:~$ hdfs dfs -chmod 777 /spark-logs
(base) hadoopuser@master:~$ hdfs dfs -ls /
Found 3 items
drwxrwxrwx   - hadoopuser hadoopgroup          0 2019-12-30 00:47 /spark-logs
drwx------   - hadoopuser hadoopgroup          0 2019-12-09 13:10 /tmp
drwxr-xr-x   - hadoopuser hadoopgroup          0 2019-12-10 20:20 /user
(base) hadoopuser@master:~$
```

Σχήμα 6-157: Δημιουργία ενός καταλόγου στο κατανεμημένο σύστημα αρχείων HDFS, για την αποθήκευση των πληροφοριών καταγραφής συμβάντων όταν τρέχει μια εφαρμογή Spark και ακολούθως απόδοση πλήρων δικαιωμάτων στον κατάλογο αυτό

Στη συνέχεια θα πρέπει να ενημερώσει σε κάθε κόμβο της συστάδας το αρχείο *$SPARK_HOME/conf/spark-defaults.conf* για να ενεργοποιήσει την καταγραφή συμβάντων και να καθορίσει τον κατάλογο για τα αρχεία καταγραφής συμβάντων [41] (βλέπε Σχήμα 6-158):

```
# cd /opt/spark/conf/
cd $SPARK_HOME/conf/
sudo gedit spark-defaults.conf
```

```
# Αρχείο spark-defaults.conf στον spark-client (Client Side)
#------------------------------------------------------------------------------
--
# Ρύθμιση του διακομιστή ιστορικού για συστάδα Spark με Διαχειριστή Πόρων Hadoop YARN
#------------------------------------------------------------------------------
---
# (Client Side: spark-client)
# Αυτή η ιδιότητα χρησιμοποιείται για την ανακατασκευή του web UI μετά την ολοκλήρωση
# της εφαρμογής.
spark.eventLog.enabled             true
```

```
# Ο κατάλογος όπου θα αποθηκεύονται οι πληροφορίες καταγραφής συμβάντων όταν τρέχει
# μια εφαρμογή Spark. Η πιο συνετή επιλογή είναι να παραπέμπει σε έναν κατάλογο του
# κατανεμημένου συστήματος αρχείων Hadoop HDFS. Ο κατάλογος  θα πρέπει να έχει
# δημιουργηθεί εκ των προτέρων.
spark.eventLog.dir                hdfs://master:9000/spark-logs

# Η Ιδιότητα αυτή καθορίζει αν θα συμπιέζονται τα αρχεία καταγραφής συμβάντων. Ο
προκαθορισμένος αλγόριθμος συμπίεσης είναι ο Snappy.
spark.eventLog.compress           true
```

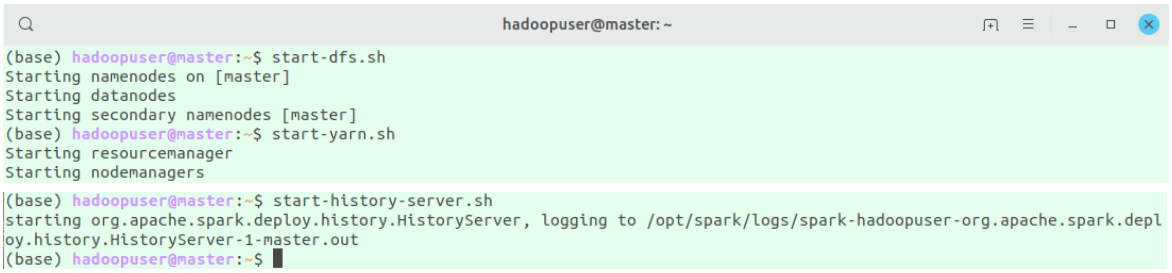
```
## Αρχείο spark-defaults.conf στον υπολογιστή που θα επιλεγεί να τρέχει ο Διακομιστής
# Ιστορικού (Server Side: π.χ master, slave1, slave2, spark-client)
#-------------------------------------------------------------------------------
# Ρύθμιση του διακομιστή ιστορικού για συστάδα Spark με Διαχειριστή Πόρων Hadoop YARN
#-------------------------------------------------------------------------------
# Ο κατάλογος αυτός θα πρέπει να είναι ο ίδιος με αυτόν της ιδιότητας
# spark.eventLog.dir στον spark-client (Client Side)
spark.history.fs.logDirectory     hdfs://master:9000/spark-logs

#spark.history.fs.update.interval   10s
# Η θύρα στην οποία συνδέεται η διεπαφή web του Διακομιστή Ιστορικού.
#spark.history.ui.port 18080

# Καθορίζει αν ο Διακομιστής Ιστορικού θα καθαρίζει τα αρχεία καταγραφής συμβάντων
# σε προκαθορισμένα χρονικά διαστήματα που καθορίζονται από την ιδιότητα
# spark.history.fs.cleaner.maxAge
spark.history.fs.cleaner.enabled   true
# Καθορίζει πόσο συχνά θα γίνεται έλεγχος από το σύστημα για την εκκαθάριση αρχείων
# καταγραφής συμβάντων. Δυνατές τιμές: 1d | 1h | 30m | 1000s
spark.history.fs.cleaner.interval  1d
# Δυνατές τιμές: 1d | 1h | 30m | 1000s
spark.history.fs.cleaner.maxAge    7d
```

Σχήμα 6-158: Ενημέρωση του αρχείο $SPARK_HOME/conf/spark-defaults.conf για να έχει πρόσβαση ο χρήστης στον Διακομιστή Ιστορικού Spark (Spark History Server)

Σημείωση: Θα πρέπει και στον *spark-client* το αρχείο *spark-defaults.conf* να έχει τις παραπάνω ρυθμίσεις εκτός από τον κόμβο στον οποίο θα τρέχει ο Διακομιστής Ιστορικού Spark[40]. Επίσης πρέπει να τονιστεί ότι για την ομαλή λειτουργία του Διακομιστή Ιστορικού Spark απαιτείται να του εκχωρηθεί μνήμη μέσω της μεταβλητής περιβάλλοντος SPARK_DAEMON_MEMORY (προεπιλογή: 1g)[41] στο αρχείο *spark-env.sh* [41]. Σε κάποιες εφαρμογές παραγωγής με μεγάλες εργασίες (jobs) μπορεί να χρειαστεί ο χρήστης να αυξήσει κατά πολύ το μέγεθός της.

Ο γράφων επέλεξε να εκκινεί τον Διακομιστή Ιστορικού στον *master*. Αφού εκκινήσει τον Διακομιστή Ιστορικού *Spark* (βλέπε Σχήμα 6-159):

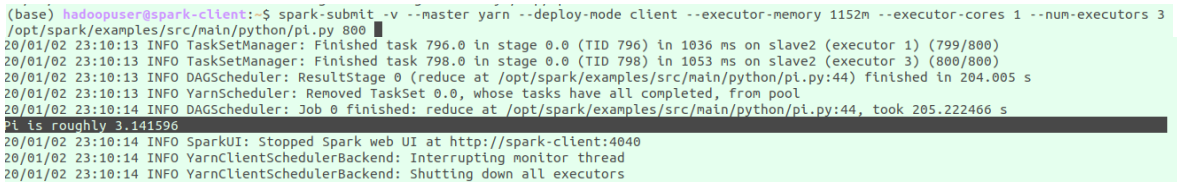
```
# master
#-------------------------------------------------------------------------------
start-dfs.sh
start-yarn.sh
# Εκκίνηση του Διακομιστή Ιστορικού Spark
start-history-server.sh
```



Σχήμα 6-159: Εκκίνηση των Hadoop HDFS / Hadoop YARN και του Spark History Server

Θα πρέπει ο χρήστης να δοκιμάσει να ξανατρέξει το παράδειγμα *pi.py* είτε σε *client mode* είτε σε *cluster mode* (βλέπε Σχήματα 6-160, 6-161, 6-162)

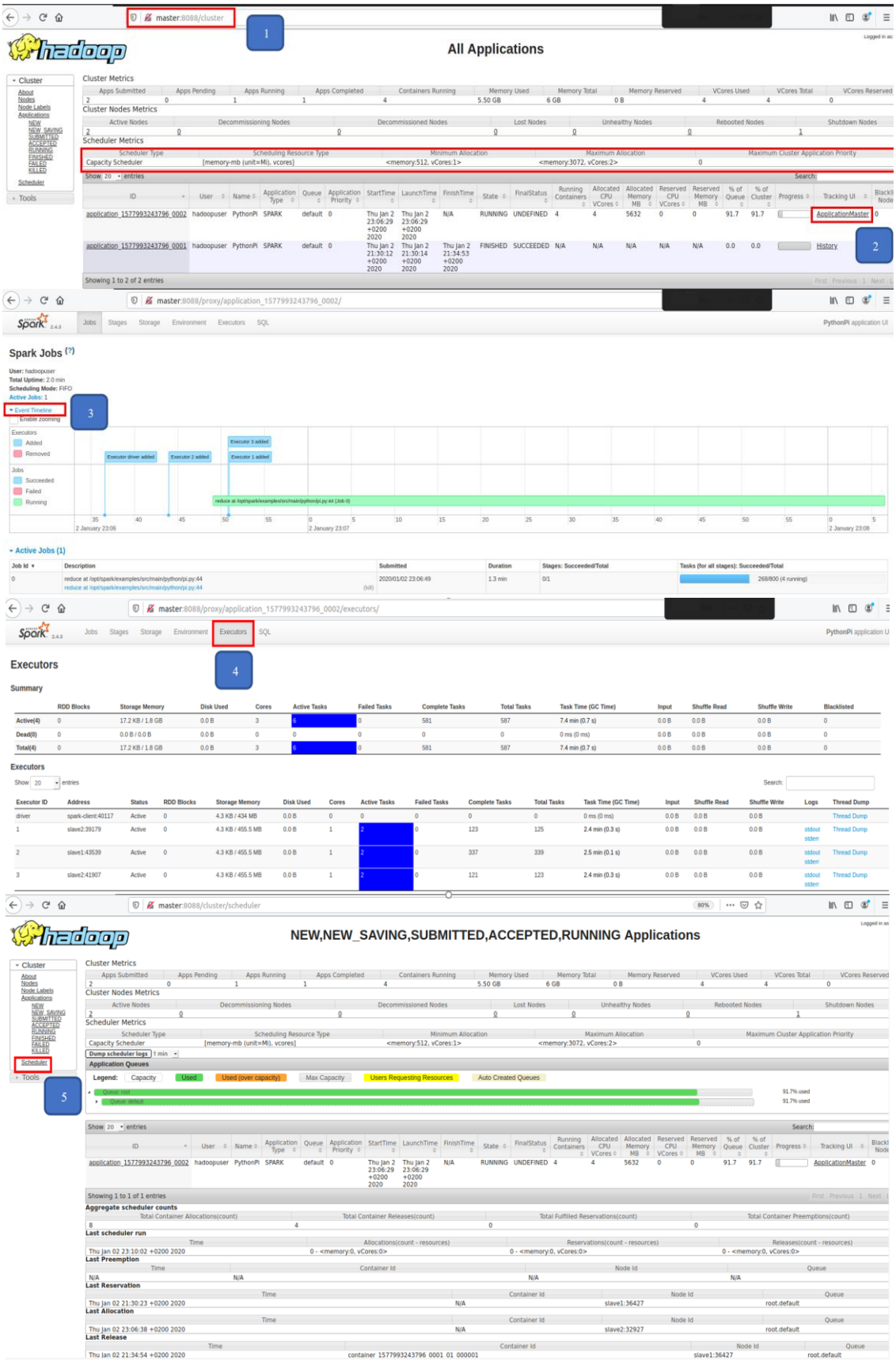
```
# spark-client
#-------------------------------------------------------------------------------
spark-submit -v --master yarn --deploy-mode client --executor-memory 1152m --executor-cores 1 --num-executors 3 /opt/spark/examples/src/main/python/pi.py 800
```



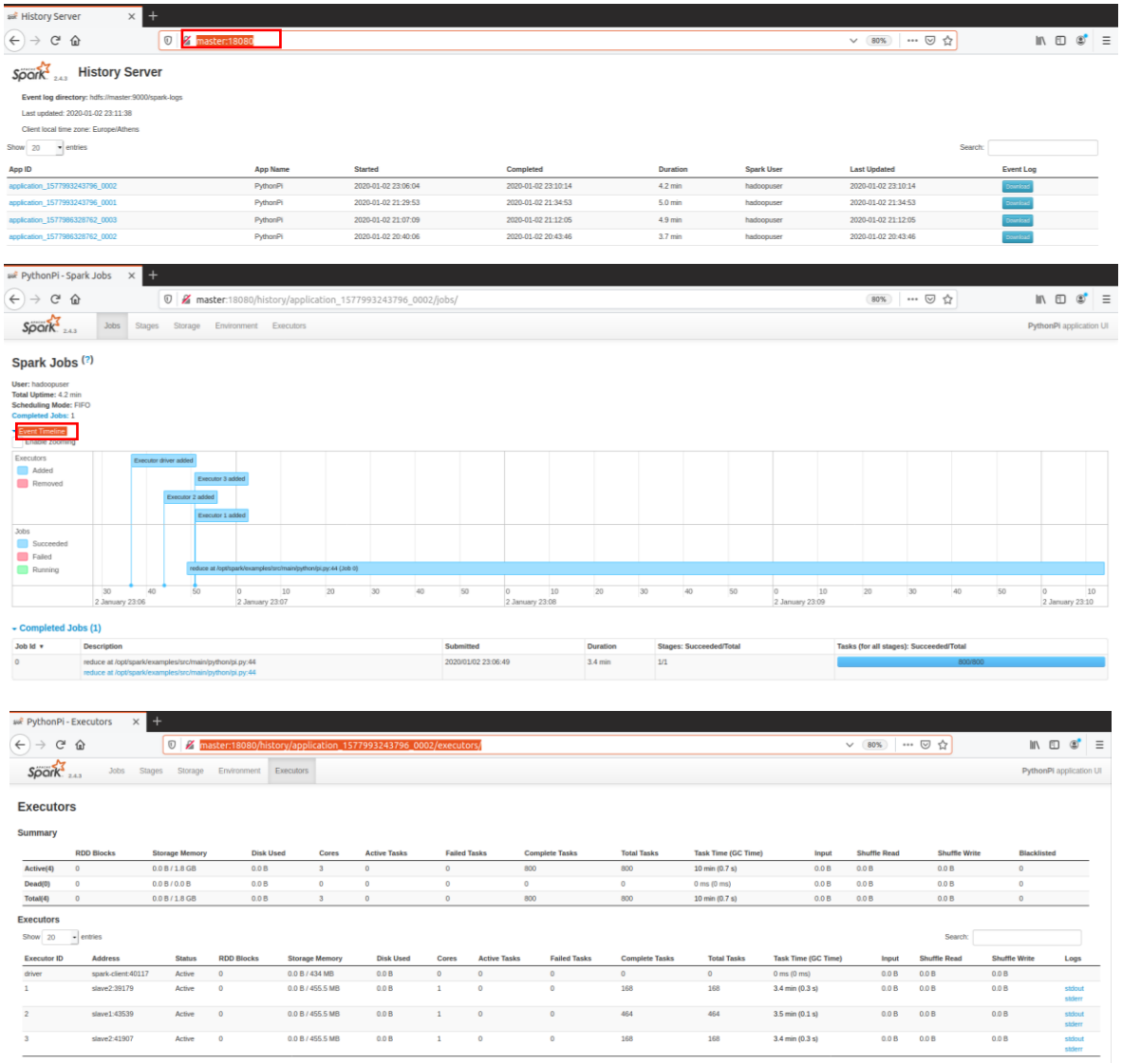
Σχήμα 6-160: Το script spark-submit με κάποια επιπλέον command line options -v /--verbose (παρέχει μακροσκελείς / αναλυτικές πληροφορίες), --executor-memory (ποσότητα μνήμης για χρήση ανά διεργασία εκτελεστή), --executor-cores (αριθμός πυρήνων ανά εκτελεστή), --num-executors (συνολικός αριθμός εκτελεστών ανά εφαρμογή Spark. Όταν είναι ενεργοποιημένη η δυναμική κατανομή πόρων, ό αρχικός αριθμός των εκτελεστών θα είναι τουλάχιστον ίσος με αυτή την επιλογή γραμμής εντολών της spark-submit)

[40] *Σε client mode μπορεί ο Διακομιστής Ιστορικού Spark να τρέχει και στον spark-client. Ειδικά για συστάδα Spark σε Standalone mode χωρίς κατανεμημένο σύστημα αρχείων HDFS είναι η μοναδική επιλογή.*

[41] *export SPARK_DAEMON_MEMORY= '2g'. Η μεταβλητή αυτή χρησιμοποιείται και για την εκχώρηση μνήμης στους δαίμονες (daemons) Spark master και worker [88].*



Σχήμα 6-161: (1) Πρόσβαση στο ResourceManager Web UI. Στο κόκκινο πλαίσιο φαίνεται ο τύπος του Χρονοδρομολογητή και το ελάχιστο και μέγιστο των πόρων που μπορεί να διαθέσει σε ένα container (2) Πρόσβαση στο Spark Web UI για όσο χρονικό διάστημα τρέχει η εφαρμογή, (3) Χρονοδιάγραμμα Γεγονότων (Event Timeline), (4) Η καρτέλα (tab) executors παρέχει πληροφορίες για τους εκτελεστές, (5) Πληροφορίες για τους πόρους που διέθεσε ο Χρονοδρομολογητής για την εκτέλεση της εφαρμογής



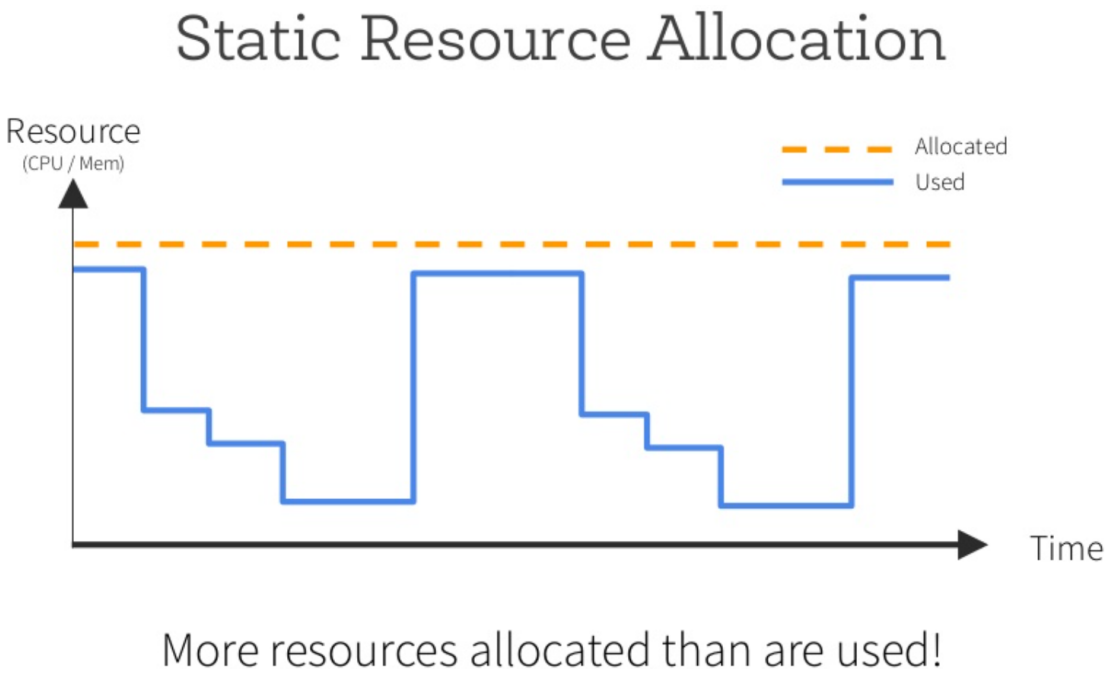
Σχήμα 6-162: Ο Διακομιστής Ιστορικού Spark

Δυναμική κατανομή πόρων (Dynamic source allocation) στο Spark σε συστάδα Hadoop YARN

Μέχρι τώρα η κατανομή των πόρων γινόταν στατικά (static resource allocation, βλέπε Σχήμα 6-163), δηλαδή πριν τρέξει ο χρήστης το παράδειγμα *py.pi* έπρεπε να έχει προκαθορίσει τους πόρους που ζητούσε να του διατεθούν από τη συστάδα. Ο προκαθορισμός αυτός των πόρων μπορούσε να υλοποιηθεί είτε στα αρχεία *spark-defaults.conf* (spark.executor.instances, spark.executor.cores, spark.executor.memory) και *spark-env.sh* (SP`ARK_EXECUTOR_CORES, SPARK_EXECUTOR_MEMORY, SPARK_EXECUTOR_INSTANCES) είτε ως επιλογές (options, --executor-cores, --executor-memory, --num-executors[42]) του script *spark-submit* είτε εντός του προγράμματος με την *SparkConf* [30]. Αυτό το προγραμματιστικό υπόδειγμα είναι κατάλληλο

[42] *Σε συστάδα Spark σε Standalone mode (βλέπε επόμενη ενότητα) χρησιμοποιείται η επιλογή γραμμής εντολών (command line options) --total-executor-cores (συνολικός αριθμός πυρήνων εκτελεστή ανά εφαρμογήtotal).*

για εφαρμογές επεξεργασίας κατά δεσμίδες (batch processing) όπου γίνεται επεξεργασία μεγάλου όγκου δεδομένων και μετά το πέρας της επεξεργασίας οι πόροι αποδεσμεύονται και τίθενται εκ νέου στην διάθεση του Χρονοδρομολογητή της συστάδας.



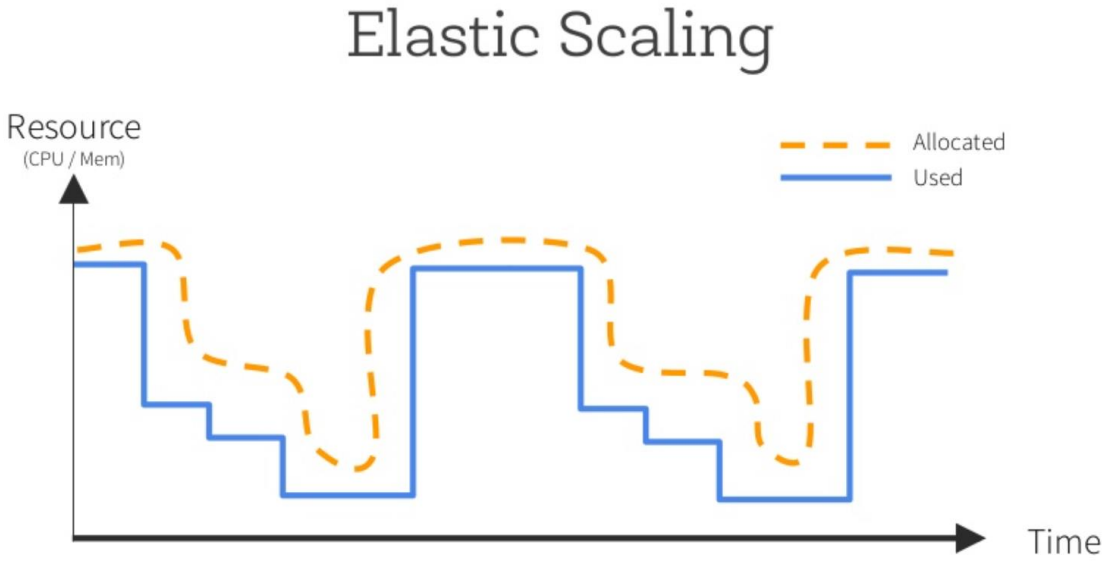
Πηγή: https://www.slideshare.net/databricks/large-scalesparktalk

Σχήμα 6-163: Στην Στατική Κατανομή Πόρων (Static Resource Allocation) οι πόροι που δεσμεύονται είναι περισσότεροι από όσους χρησιμοποιούνται

Οι πόροι όμως είναι σημαντικοί παράγοντες που επηρεάζουν την αποδοτική εκτέλεση του Spark. Είναι δύσκολο όμως να εκτιμηθεί ο ακριβής φόρτος εργασίας και έτσι να καθοριστεί εκ των προτέρων ο απαιτούμενος αριθμός εκτελεστών. Αν για παράδειγμα για μια μακροχρόνια υπηρεσία (όπως ο JDBCServer) χωρίς έργα δεσμευτούν πολλαπλοί εκτελεστές τότε οι εναπομείναντες για δυνητικές εφαρμογές πόροι θα είναι ανεπαρκείς. Αυτό το απλό παράδειγμα καταδεικνύει την σπατάλη πόρων και την ανεπαρκή χρονοδρομολόγησή τους. Ο μηχανισμός που διαθέτει το *Spark* για την αποφυγή τέτοιων καταστάσεων τόσο σε μια *multi-tenant* συστάδα όσο και σε μια *single-tenant* (όπου πολλαπλές εφαρμογές του χρήστη/μισθωτή μοιράζονται τους πόρους της) είναι η δυναμική χρονοδρομολόγηση (χρονοπρογραμματισμός) των πόρων που επιτρέπει την αύξηση ή την μείωση **του αριθμού των εκτελεστών** (--num-executors / spark.executor.instances) με βάση τις συνθήκες φόρτου εργασίας (βλέπε Σχήμα 6-164).

Ο λόγος που απαιτείται η δυναμική κατανομή των πόρων σε μια συστάδα YARN είναι γιατί τα MRv2 (MapReduce version 2) έργα εκτελούνται σε διεργασίες βραχείας διάρκειας που η

διάρκεια τους καθορίζεται από τη διάρκεια των *έργων* και τα *containers* δεσμεύονται μόνο για αυτή τη χρονική διάρκεια ενώ στο *Spark* τα *έργα* εκτελούνται ως νήματα σε διεργασίες μακράς διάρκειας, τους εκτελεστές, που η διάρκειά τους είναι ίση με τη διάρκεια του κύκλου ζωής της εφαρμογής *Spark* και τα *containers* παραμένουν δεσμευμένα για αυτό το χρονικό διάστημα.



Πηγή: https://www.slideshare.net/databricks/large-scalesparktalk

Σχήμα 6-164: Στη Δυναμική κατανομή Πόρων (Dynamic Resource Allocation / Elastic Scaling) κάθε χρονική στιγμή υπάρχει μια αντιστοιχία ανάμεσα στους πόρους που δεσμεύονται και σε αυτούς που χρησιμοποιούνται.

Η προεπιλεγμένη διαμόρφωση του *Spark* επιτρέπει τη στατική κατανομή πόρων. Αν ο χρήστης το επιθυμεί μπορεί να αλλάξει τη διαμόρφωση του *Spark* έτσι ώστε να επιτρέπει τη δυναμική κατανομή πόρων ακολουθώντας τα παρακάτω βήματα:

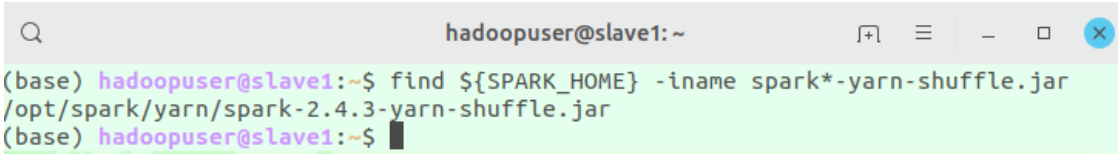
- Θα πρέπει να ενημερώσει το αρχείο *spark-defaults.conf* σε κάθε *worker/slave (NodeManager)* κόμβο με τις ακόλουθες *Spark* ιδιότητες [30][42]:

```
# Καθορίζει εάν πρόκειται να χρησιμοποιηθεί η δυναμική κατανομή πόρων
# (προκαθορισμένη τιμή: false)
spark.dynamicAllocation.enabled                              true

# Καθορίζει εάν θα ρυθμιστεί μια εξωτερική υπηρεσία shuffle, η οποία θα διατηρεί
# τα γραμμένα από τους εκτελεστές αρχεία shuffle ώστε οι εκτελεστές να μπορούν να
# αφαιρεθούν με ασφάλεια:
spark.shuffle.service.enabled                                true

# Μερικές προαιρετικές ιδιότητες:
# Λαμβάνονται υπόψη μόνο όταν είναι ενεργοποιημένη η Δυναμική Κατανομή
#------------------------------------------------------------------------------
# Ελάχιστος αριθμός εκτελεστών (προκαθορισμένη τιμή: 0)
spark.dynamicAllocation.minExecutors                         0
# Μέγιστος αριθμός εκτελεστών (προκαθορισμένη τιμή: infinity)
spark.dynamicAllocation.maxExecutors                         3
# Αρχικός αριθμός εκτελεστών
# (προκαθορισμένη τιμή: spark.dynamicAllocation.minExecutors)
```

```
spark.dynamicAllocation.initialExecutors                    0
# Εάν ένας εκτελεστής έχει παραμείνει αδρανής για περισσότερο από αυτή τη διάρκεια,
# ο εκτελεστής θα καταργηθεί. (προκαθορισμένη τιμή:60s)
spark.dynamicAllocation.executorIdleTimeout                 90s
# Εάν υπάρχουν καθυστερημένες εργασίες που εκκρεμούν για περισσότερο από αυτή τη
# διάρκεια, θα ζητηθούν νέοι εκτελεστές (προκαθορισμένη τιμή:1s)
spark.dynamicAllocation.schedulerBacklogTimeout             5s
# Όπως η παραπάνω ιδιότητα αλλά χρησιμοποιείται μόνο για μεταγενέστερες αιτήσεις
# εκτελεστή(προκαθορισμένη τιμή:spark.dynamicAllocation.schedulerBacklogTimeout)
spark.dynamicAllocation.sustainedSchedulerBacklogTimeout    5s
```

- Θα πρέπει να εντοπίσει, σε κάθε *worker/slave (NodeManager)* κόμβο το *αρχείο* `spark-<version>-yarn-shuffle.jar` [36].

```
find ${SPARK_HOME} -iname spark*-yarn-shuffle.jar
```

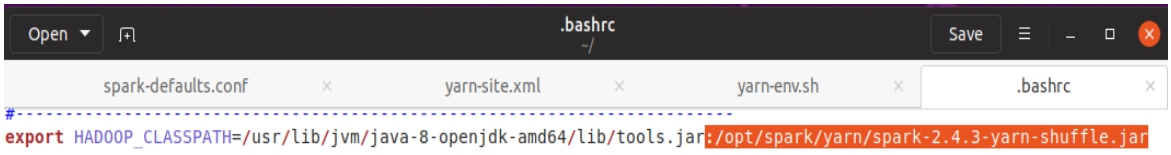


Σχήμα 6-165: Εντοπισμός του αρχείου spark-<version>-yarn-shuffle.jar με την εντολή find

- Και να το προσθέσει στο classpath κάθε *worker/slave (NodeManager)* κόμβου της συστάδας του. Ο γράφων ενημέρωσε το classpath στο αρχείο *.bashrc* ως ακολούθως:

```
sudo gedit ~/.bashrc
```

```
# Αρχείο.bashrc
export HADOOP_CLASSPATH=/usr/lib/jvm/java-8-openjdk-
amd64/lib/tools.jar:/opt/spark/yarn/spark-2.4.3-yarn-shuffle.jar
```



Σχήμα 6-166: Ενημέρωση του HADOOP_CLASSPATH σε κάθε worker/slave (NodeManager) κόμβο

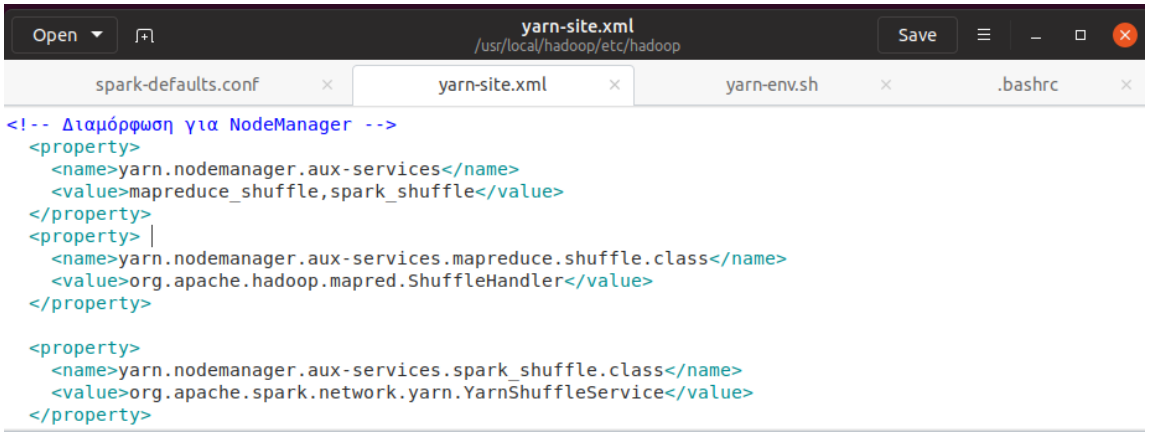
Για να εφαρμοστούν οι αλλαγές, θα πρέπει να πληκτρολογήσει:

```
source ~/.bashrc
```

- Να τροποποιήσει σε κάθε *worker/slave (NodeManager)* κόμβο το αρχείο *yarn-site.xml:*

```
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle,spark_shuffle</value>
  </property>

  <property>
    <name>yarn.nodemanager.aux-services.spark_shuffle.class</name>
    <value>org.apache.spark.network.yarn.YarnShuffleService</value>
  </property>
```

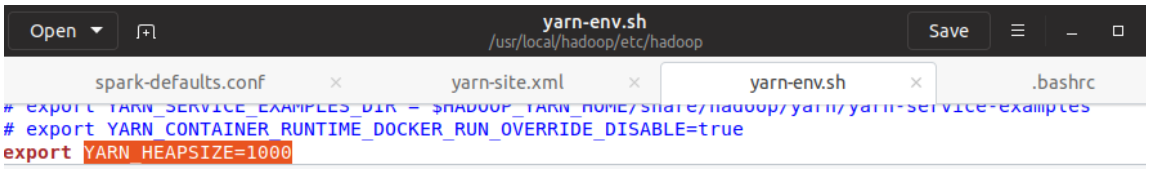
Με έντονα μπλε γράμματα είναι οι προσθήκες στο αρχείο



Σχήμα 6-167: Τροποποίηση του αρχείου yarn-site.xml σε κάθε worker/slave (NodeManager) κόμβο

- Θα πρέπει να ενημερώσει το αρχείο *yarn-env.sh* σε όλους τους workers/slaves κόμβους με την ακόλουθη μεταβλητή περιβάλλοντος [36]:

```
# Η μεταβλητή περιβάλλοντος αυτή θα πρέπει να ρυθμιστεί ανάλογα με τις εκάστοτε
# συνθήκες
YARN_HEAP_SIZE=1000
```



Σχήμα 6-168: Ενημέρωση της μεταβλητής περιβάλλοντος YARN_HEAPSIZE στο αρχείο yarn-env.sh σε κάθε worker/slave (NodeManager) κόμβο

- Θα πρέπει στη συνέχεια να επανεκκινήσει όλους τους NodeManagers της συστάδας:

```
# master - σταμάτημα του YARN
stop-yarn.sh
```

Παρατήρηση: Όταν είναι ενεργοποιημένη η *Δυναμική Κατανομή Πόρων* δε θα πρέπει η εκκίνηση του YARN να γίνεται με το script *start-yarn.sh* από τον master κόμβο αλλά:

```
# master - εκκίνηση ResourceManager
yarn --daemon start resourcemanager

#-------------------------------------------------------------------------------

# slave1, slave2 - εκκίνηση NodeManager
yarn --daemon start nodemanager
```

Το σταμάτημα του YARN μπορεί να γίνει με το script *stop-yarn.sh*.

Ο έλεγχος ορθής λειτουργίας της Δυναμικής Κατανομής Πόρων θα γίνει στην επόμενη ενότητα μέσω του Jupyter Notebook.

Το Pyspark μέσω του Jupyter Notebook σε συστάδα με διαχειριστή πόρων το YARN

Αναλυτικές πληροφορίες για το Jupyter Notebook και τον τρόπο εγκατάστασής του σε εικονικό περιβάλλον conda (virtual environment) παρέχονται στην υποενότητα "Βήμα 5ο: Το PySpark μέσω του Jupyter Notebook". Στην υποενότητα αυτή εγκαταστάθηκε το Jupyter Notebook στο περιβάλλον conda *pyspark_env*. Το περιβάλλον conda *pyspark_env* αναδημιουργήθηκε στην υποενότητα "Λήψη και εγκατάσταση του Spark (σε μια συστάδα Hadoop Yarn)". Συνεπώς υπάρχουν όλα τα προαπαιτούμενα για να μπορεί ο χρήστης να τρέξει το Jupyter. Το εικονικό περιβάλλον conda *pyspark_env* επειδή έχει τα ίδια εγκατεστημένα πακέτα σε όλους τους κόμβους της συστάδας επιτρέπει στον χρήστη να χρησιμοποιήσει στο πρόγραμμα του όλα αυτά τα πακέτα χωρίς να απαιτείται η ενσωμάτωση τους μέσω zip ή egg[43] αρχείων με το script *spark_submit*. Με ένα απλό script μπορεί ο διαχειριστής της συστάδας να ενημερώνει το περιβάλλον pyspark_env με νέα πακέτα σε όλους τους κόμβους της συστάδας από τον κόμβο master.

Στη συνέχεια επιδεικνύεται η ενεργοποίηση του περιβάλλοντος conda *pyspark_env*[44] και η εκκίνηση του *Jupyter notebook (βλέπε Σχήμα 6-170, εικόνα 1)*. Μετά τη δημιουργία ενός στιγμιότυπου του *SparkSession* (*SparkSession Instance*) μέσω του *builder design pattern* (βλέπε Σχήμα 6-169, κελί In[5] του Jupyter Notebook) ακολουθεί η εκτέλεση του προγράμματος *υπολογισμού του* $\boldsymbol{\pi}$ *για δυο διαφορετικές τιμές* (*Job ID:0, 300* και Job ID:1, *2500*) της μεταβλητής *partitions* και η παρακολούθηση μέσω του Spark UI της προόδου εκτέλεσης των εργασιών Job ID 0 & 1 (βλέπε Σχήμα 6-170). Όπως φαίνεται στο χρονοδιάγραμμα γεγονότων (event timeline) με την εκκίνηση των εργασιών προσαρτώνται όλοι οι εκτελεστές και στη συνέχεια με το πέρας των εργασιών αυτών αποδίδονται πίσω στη συστάδα. Διαπιστώνεται λοιπόν ότι η Δυναμική Κατανομή Πόρων[45] λειτούργησε με τον τρόπο που αναμενόταν.

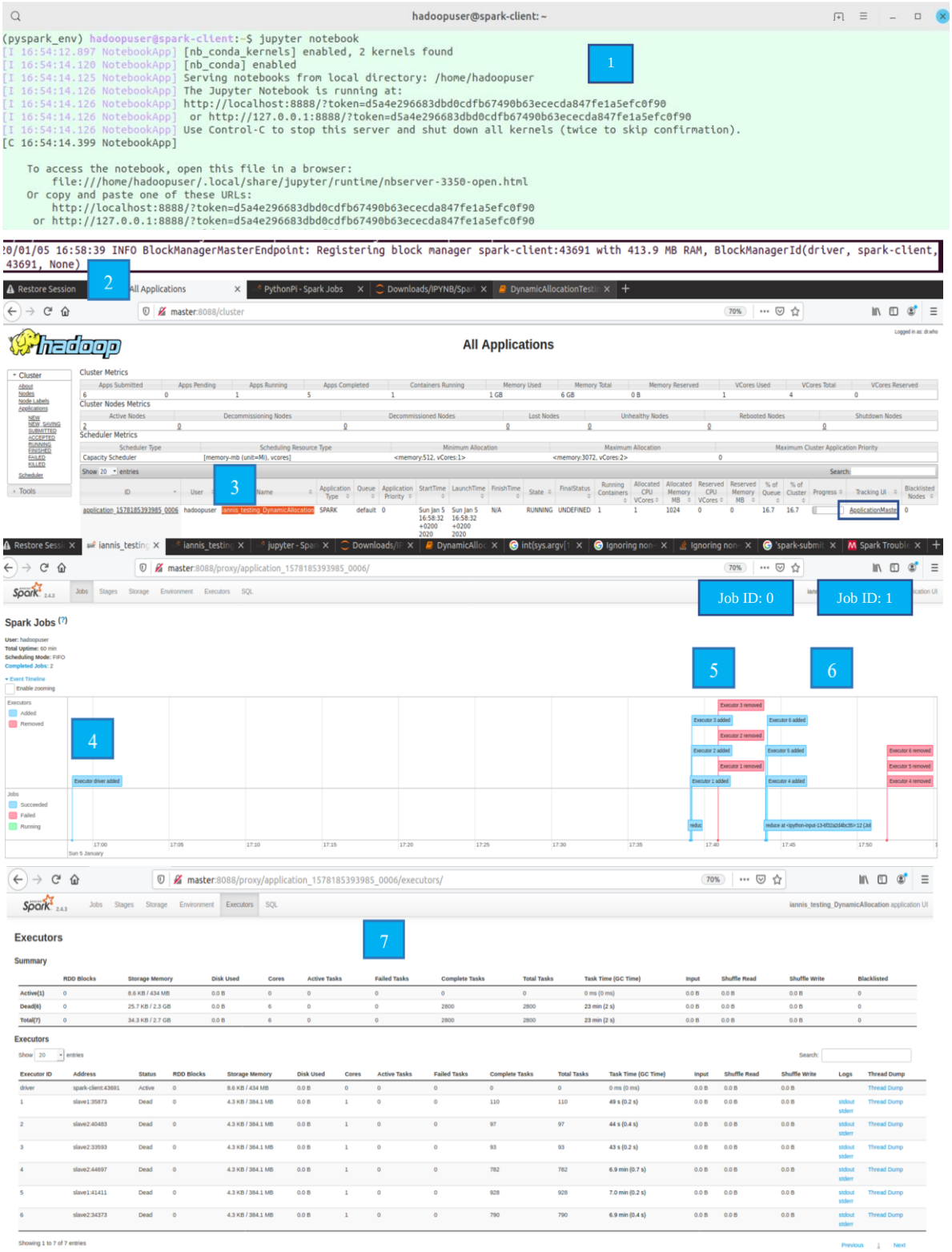
```
# spark-client
conda activate pyspark_env
jupyter notebook
```

[43] *Τα αρχεία egg είναι μια παλαιότερη μορφή διανομής Python πακέτων (η νέα μορφή διανομής πακέτων Python ονομάζεται wheel).*

[44] *Για να μην απαιτείται η ενεργοποίηση του περιβάλλοντος conda pyspark_env μπορεί ο χρήστης να εισάγει το script* ***conda activate pyspark_env*** *μέσα στο ~/.bashrc αρχείο του spark-client. Ο γράφων το εισήγαγε σε όλους τους κόμβους της συστάδας και στον spark-client.*

[45] *Στην ενότητα "Το Pyspark μέσω του Jupyter Notebook σε συστάδα Spark με εγγενή διαχειριστή πόρων (Spark Standalone mode)" δίδεται ένα πιο ολοκληρωμένο παράδειγμα όπου το πρόγραμμα pi.py τρέχει ταυτόχρονα μέσω του script spark-submit και μέσω του Jupyter notebook.*

Σχήμα 6-169: Στο κελί In[5] δημιουργείται ένα στιγμιότυπο του SparkSession με ενεργοποιημένη την Δυναμική Κατανομή Πόρων. Στο κελί In[11] τρέχει το πρόγραμμα υπολογισμού του π, με partitions=300 και στο κελί In[13] με partitions=2500.



Σχήμα 6-170: (1) Εκκίνηση του Jupyter notebook, (2),(3), (4) Η εφαρμογή Spark (iannis_testing_DynamicAllocation) εκκινεί όταν ο χρήστης τρέχει το κελί In[5] στο Jupyter notebook (βλέπε Σχήμα 6-169) και αμέσως εκκινεί και ο spark driver (στον spark-client, γιατί όταν δεν ορίζεται η επιλογή --deploy-mode ξεκινά σε client-mode[47]), (5) Job ID: 0 (executors with ID: 1,2,3). Όταν τρέχει το πρόγραμμα υπολογισμού του π, με partitions=300 (βλέπε κελί In[11], Σχήμα 6-169) (6) Job ID: 1 (executors with ID: 4,5,6). Όταν τρέχει το πρόγραμμα υπολογισμού του π, με partitions=2500 (βλέπε κελί In[13], Σχήμα 6-169) , (7) Βλέπουμε ότι στο τέλος έχουμε Active(1) τον driver (spark-client) με Executor ID:driver και Dead (6) executors από σύνολο - Total (7) executors.

[47] *client-mode: .config("spark.submit.deployMode", "client"), cluster-mode: .config("spark.submit.deployMode", "cluster")*

Εγκατάσταση του Spark με τον εγγενή διαχειριστή συστάδας Spark (Spark Standalone Cluster Manager)

Ο αυτόνομος (εγγενής) διαχειριστής συστάδας Spark (Spark Standalone cluster manager) είναι ένας απλός διαχειριστής συστάδας που είναι διαθέσιμος ως μέρος της διανομής Spark. Χρησιμοποιεί έναν απλό Χρονοδρομολογητή FIFO για εφαρμογές. Από προεπιλογή, κάθε εφαρμογή χρησιμοποιεί όλους τους διαθέσιμους κόμβους της συστάδας. Ο αριθμός των κόμβων μπορεί να περιοριστεί *ανά εφαρμογή*, *ανά χρήστη* ή *σε συνολικό επίπεδο*. Εξ ορισμού, η αυτόνομη συστάδα Spark είναι ανθεκτική στις αποτυχίες εργάτη (worker). Μπορεί επίσης να εξασφαλίσει και Υψηλή Διαθεσιμότητα (High Availability, HA) μέσω ενός ξεχωριστού ελεγκτή αποτυχίας ZooKeeper[48] (failover controller). Υπάρχουν δυο τρόποι εγκατάστασης μιας συστάδας Spark:

- Εγκατάσταση Αυτόνομης Συστάδας Spark (Spark Standalone)
- Εγκατάσταση του Spark παράλληλα με το κατανεμημένο σύστημα αρχείων Hadoop HDFS. Απαιτεί απλά την εκκίνησή του ως ξεχωριστή υπηρεσία στους ίδιους κόμβους [43], εξασφαλίζοντας με αυτόν τον τρόπο πρόσβαση στο κατανεμημένο σύστημα αρχείων HDFS (Hadoop Distributed File System).

Για να εγκαταστήσει κανείς το *Spark* σε *Standalone mode*, θα πρέπει απλά να τοποθετήσει μια μεταγλωττισμένη έκδοση του *Spark* σε κάθε κόμβο της συστάδας [43]. Το προαπαιτούμενο λογισμικό φαίνεται στον Πίνακα 6-1 ενώ τα βήματα εγκατάστασης περιγράφονται αναλυτικά στην προηγούμενη ενότητα "Εγκατάσταση του Spark σε μια συστάδα Hadoop YARN". Για λόγους συνοχής τα βήματα αυτά αναφέρονται παρακάτω:

- Πλάνο εγκατάστασης του Spark σε συστάδα Hadoop YARN και σε Standalone συστάδα
- Προκαταρκτικές Εργασίες πριν την δημιουργία των Εικονικών Μηχανών
- Δημιουργία Εικονικής Μηχανής
- Εγκατάσταση του Ubuntu 19.04
- Αλλαγή του ονόματος της κύριας ομάδας (primary groupname) του χρήστη με όνομα hadoopuser
- Εγκατάσταση προσθηκών επισκέπτη Virtualbox
- Διαμόρφωση αρχείου hosts

[48] *Μπορεί κανείς να διαμορφώσει πολλούς master κόμβους στη συστάδα του, για σκοπούς HA, που θα συνδέονται με το ίδιο στιγμιότυπο του Zookeeper που παρέχει μηχανισμό εκλογής αρχηγού (leader election). Έτσι εξασφαλίζεται ότι κάθε χρονική στιγμή θα υπάρχει ένας εκλεγμένος αρχηγός ενώ οι υπόλοιποι θα είναι σε standby mode.*

- Ρύθμιση της κάρτας Δικτύου στο VirtualBox και της στατικής διεύθυνσης IP στο Ubuntu
- Εγκατάσταση της Java
- Λήψη και Εγκατάσταση του Anaconda (για python)
- Λήψη και εγκατάσταση του Hadoop / YARN[49] (στην περίπτωση που θέλει ο χρήστης να εκμεταλλευτεί το κατανεμημένο σύστημα αρχείων HDFS)
- Λήψη και εγκατάσταση του Spark
- Δημιουργία κλώνων (hadoop-slave1, hadoop-slave2 και spark-client)
- Ρύθμιση του SSH για αυθεντικοποίηση χρήστη χωρίς κωδικό πρόσβασης
- Τροποποιήσεις και τελικές ρυθμίσεις στους κόμβους
    - Έλεγχος της συστάδας Hadoop YARN (μόνο έλεγχο του HDFS)
    - Διαδικτυακή διεπαφή χρήστη (Web UI) για το HDFS (NameNode UI)
    - Το Pyspark μέσω του Jupyter Notebook σε συστάδα με διαχειριστή πόρων το YARN. Κατά τη δημιουργία του στιγμιότυπου του SparkSession (SparkSession Instance) μέσω του builder design pattern (βλέπε Σχήμα 6-169, κελί In[5]) αντί για *.master yarn* θα πρέπει να εισάγει ο χρήστης *.master spark://master:7077*.

Διαμόρφωση των αρχείων spark-env.sh, spark-defaults.conf και slaves για συστάδα Spark σε Standalone mode

Παρακάτω στον Πίνακα 6-5 δίνονται βασικές ιδιότητες Spark (Spark properties) και μεταβλητές περιβάλλοντος για τη διαμόρφωση της συστάδας Spark σε Standalone mode.

Πίνακας 6-5: Ιδιότητες Spark και Μεταβλητές Περιβάλλοντος για τη διαμόρφωση μιας συστάδας Spark σε Standalone mode

	Ανά εργάτη (worker)	Ανά εφαρμογή (Application)	Ανά εκτελεστή (executor)
CPU	SPARK_WORKER_CORES	spark.deploy.defaultCores [50] spark.cores.max[51]	spark.executor.cores[52]
Memory	SPARK_WORKER_MEMORY		spark.executor.memory

Από τον παραπάνω πίνακα για client mode [53] και υποθέτοντας ότι υπάρχει αρκετή μνήμη προκύπτει ότι :

[49] *Όχι το κομμάτι που αφορά το YARN*
[50] *Μέσω της SPARK_MASTER_OPTS στο αρχείο spark-env.sh στον master.*
[51] *Η αντίστοιχη επιλογή γραμμής εντολών (command line option) της spark-submit είναι η: --total-executor-cores*
[52] *spark.executor.cores ≤ spark.cores.max*
[53] *Επί του παρόντος, το **Spark Standalone mode** δεν υποστηρίζει **cluster mode** για εφαρμογές **Python** [31]*

- Ο αριθμός των ταυτόχρονων εφαρμογών που μπορεί να υποστηρίξει μια συστάδα Spark εκφράζεται από την ακόλουθη εξίσωση:

$$\#Applications/cluster = floor((SPARK\_WORKER\_CORES \times \#workers/cluster) \div spark.cores.max^{54})$$
$$= floor((\#Cores/cluster) \div spark.cores.max)$$

- Ο αριθμός των εκτελεστών που θα αναπτύξει το Spark για κάθε εφαρμογή εκφράζεται από την ακόλουθη εξίσωση:

#executors/application = floor(spark.cores.max (ή spark.deploy.defaultCores) ÷ spark.executor.cores)

- *Το μέγεθος της μνήμη ανά εργάτη εκφράζεται από την ακόλουθη εξίσωση:*

$$SPARK\_WORKER\_MEMORY \geq (spark.executor.memory \times \#executors/application \times \#Applications/cluster) \div (\#workers/cluster)$$

Ακολουθεί η ενημέρωση των αρχείων *spark-env.sh* και *spark-defaults.sh*. Θα πρέπει να τονιστεί ότι δεν είναι απαραίτητο να ορισθούν όλες οι παρακάτω ιδιότητες Spark και οι μεταβλητές περιβάλλοντος καθώς αποδίδονται σε αυτές προεπιλεγμένες από το Spark τιμές.

```
# slave1, slave2
gedit /opt/spark/conf/spark-env.sh
```

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
# Καθορισμός του δυαδικού αρχείου Python που θα χρησιμοποιηθεί από το πρόγραμμα οδήγησης # Spark και τους εκτελεστές
export PYSPARK_PYTHON='/opt/anaconda/envs/pyspark_env/bin/python3'
# Το συνολικό ποσό μνήμης που επιτρέπεται σε εφαρμογές Spark να χρησιμοποιήσουν στο μηχάνημα - Προκαθορισμένη τιμή: Το σύνολο της μνήμης μείον 1GB
export SPARK_WORKER_MEMORY='2688m'[55]
# Ο συνολικός αριθμός πυρήνων που επιτρέπεται σε εφαρμογές Spark να χρησιμοποιήσουν στο μηχάνημα - Προκαθορισμένη τιμή: όλοι οι διαθέσιμοι πυρήνες
export SPARK_WORKER_CORES=2

#Επιτρέπει την περιοδική εκκαθάριση των καταλόγων εφαρμογών στους εργάτες
# spark.worker.cleanup.enabled
# Καθορίζει το χρονικό διάστημα, σε δευτερόλεπτα, κατά το οποίο ο εργάτης καθαρίζει
# καταλόγους εργασιών παλαιότερων εφαρμογών στο τοπικό κόμβο
# spark.worker.cleanup.interval
# Καθορίζει το χρονικό διάστημα σε δευτερόλεπτα που θα διατηρούνται οι κατάλογοι εργασιών
# εφαρμογής σε κάθε εργάτη (π.χ. 7dx(24h/d)x(60m/h)x(60s/m)=604800s)
# Dspark.worker.cleanup.appDataTtl
export SPARK_WORKER_OPTS="-Dspark.worker.cleanup.enabled=true -Dspark.worker.cleanup.interval=2100 -Dspark.worker.cleanup.appDataTtl=604800"
```

```
# master
```

[54] *ή το spark.deploy.defaultCores αν δεν έχει ορισθεί τιμή για το spark.cores.max*

[55] *3072m – 384m =2668m . Το 384m αφορά το spark.executor.memoryOverhead και προκύπτει από την πράξη max(3072m*10%, 384m) [40]*

```
gedit /opt/spark/conf/spark-env.sh
```

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64

# Καθορισμός του δυαδικού αρχείου Python που θα χρησιμοποιηθεί από το πρόγραμμα
οδήγησης # Spark και τους εκτελεστές
export PYSPARK_PYTHON='/opt/anaconda/envs/pyspark_env/bin/python3'

# Καθορίζει τον προεπιλεγμένο αριθμό πυρήνων που δίνει στις εφαρμογές σε συστάδα Spark
# σε Standalone mode εάν δεν έχει οριστεί το spark.cores.max – Προεπιλεγμένη τιμή:
# (infinite)
export SPARK_MASTER_OPTS="-Dspark.deploy.defaultCores=4"
#------------------------------------------------------------------------------
#export SPARK_MASTER_HOST='192.168.1.100'
#export SPARK_MASTER_PORT='7077'
```

```
# spark-client
gedit /opt/spark/conf/spark-env.sh
```

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64

# Καθορισμός του δυαδικού αρχείου Python που θα χρησιμοποιηθεί από το πρόγραμμα
# οδήγησης Spark και τους εκτελεστές
export PYSPARK_PYTHON='/opt/anaconda/envs/pyspark_env/bin/python3'
```

```
# spark-client
gedit /opt/spark/conf/spark-defaults.conf
```

```
# Η URL του διαχειριστή συστάδας
spark.master                     spark://master:7077
# Ποσότητα μνήμης που διατίθεται driver - Προκαθορισμένη τιμή: 1g
spark.driver.memory              768m
# Πυρήνες που διατίθενται ανά εκτελεστή - Προκαθορισμένη τιμή: όλοι οι πυρήνες του
# εργάτη
spark.executor.cores             2
# Ποσότητα μνήμης που διατίθεται ανά διεργασία εκτελεστή - Προκαθορισμένη τιμή: 1g
spark.executor.memory            2688m
# Το σύνολο των πυρήνων που διατίθενται ανά εφαρμογή στη συστάδα Spark – Προκαθορισμένη
# τιμή: Δεν έχει
# Ο μέγιστος αριθμός πυρήνων CPU που μπορούν να διατεθούν ανά εφαρμογή στη συστάδα
# Προκαθορισμένη τιμή: αν δεν τεθεί τιμή θα πάρει την τιμή του
# spark.deploy.defaultCores
# spark.cores.max                 4
```

Για να είναι δυνατή η εκκίνηση των *slaves* κόμβων από τον *master* πρέπει ο χρήστης να δημιουργήσει ένα αντίγραφο, με όνομα *slaves*, του αρχείου *slaves.template* που βρίσκεται στον κατάλογο */opt/spark/conf* και κατόπιν να το ενημερώσει με τους κόμβους της συστάδας. Αφού ανοίξει το αρχείο slaves στον κόμβο master με το gedit:

```
#master
cd /opt/spark/conf/
ls -al
cp slaves.template slaves
gedit /opt/spark/conf/slaves
```

Θα πρέπει να πληκτρολογήσει:

```
#αρχείο slaves
slave1
slave2
```

Έλεγχος της εκκίνησης εφαρμογών του Spark σε client mode σε συστάδα Spark σε Standalone mode[56]

Θα πρέπει στην αρχή ο χρήστης να εκκινήσει τη συστάδα Spark σε Standalone mode εκτελώντας στον master κόμβο τα παρακάτω scripts:

```
# master
# Εκκίνηση του διακομιστή master
start-master.sh[57] --host master --port 7077
# Εκκίνηση των slaves κόμβων που περιλαμβάνονται στο αρχείο slaves
start-slaves.sh[58]
```

```
(pyspark_env) hadoopuser@master:~$ start-master.sh --host master --port 7077
starting org.apache.spark.deploy.master.Master, logging to /opt/spark/logs/spark-hadoopuser-org.apache.spark.deploy
.master.Master-1-master.out
(pyspark_env) hadoopuser@master:~$ start-slaves.sh
slave2: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-hadoopuser-org.apache.spar
k.deploy.worker.Worker-1-slave2.out
slave1: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-hadoopuser-org.apache.spar
k.deploy.worker.Worker-1-slave1.out
```

Σχήμα 6-171: Εκκίνηση της συστάδας Spark σε Standalone mode (Εκκίνηση του διακομιστή master και των slaves κόμβων)

Προαιρετικά μπορεί ο χρήστης να εκκινήσει τους slaves μέσω ssh:

```
# master
ssh slave1 ". ~/.bashrc;/opt/spark/sbin/start-slave.sh[59] spark://master:7077"
ssh slave2 ". ~/.bashrc;/opt/spark/sbin/start-slave.sh spark://master:7077"

#ή

#master
ssh slave1
# slave1
start-slave.sh spark://master:7077
exit

#master
ssh slave2
#slave2
start-slave.sh spark://master:7077
exit
```

Ακολουθεί έλεγχος της εκκίνησης εφαρμογών του *Spark* σε *client mode*, εκτελώντας με το script *spark-submit* το παράδειγμα *pi.py* (βλέπε Σχήμα 6-172) στην ΕΜ *spark-client*:

```
# spark-client
spark-submit –verbose --master spark://master:7077 --deploy-mode client
/opt/spark/examples/src/main/python/pi.py 300
```

[56] *Με ή χωρίς υποστήριξη από το κατανεμημένο σύστημα αρχείων HDFS. Στην περίπτωση συστάδας Spark που υποστηρίζεται από το HDFS θα πρέπει να γίνει πρώτα η εκκίνηση του κατανεμημένου συστήματος αρχείων Hadoop HDFS με το script: start-dfs.sh*

[57] *Το σταμάτημα του master γίνεται με το script stop-master.sh*

[58] *Το σταμάτημα των slaves γίνεται με το script /opt/spark/sbin/stop-slaves.sh*

[59] *Το σταμάτημα του slave τοπικά ή μέσω ssh γίνεται με το script stop-slave.sh*

```
(pyspark_env) hadoopuser@spark-client:~$ spark-submit --verbose --master spark://master:7077 --deploy-mode client  /opt/spark/examples
/src/main/python/pi.py 300
Using properties file: /opt/spark/conf/spark-defaults.conf
Adding default property: spark.driver.memory=768m
Adding default property: spark.executor.memory=2688m
Adding default property: spark.master=spark://master:7077
Adding default property: spark.executor.cores=2
20/01/08 16:47:42 INFO StandaloneAppClient$ClientEndpoint: Executor added: app-20200108164741-0000/0 on worker-20200108121940-192.168.
1.102-33809 (192.168.1.102:33809) with 2 core(s)
20/01/08 16:47:42 INFO StandaloneSchedulerBackend: Granted executor ID app-20200108164741-0000/0 on hostPort 192.168.1.102:33809 with
2 core(s), 2.6 GB RAM
20/01/08 16:47:42 INFO StandaloneAppClient$ClientEndpoint: Executor added: app-20200108164741-0000/1 on worker-20200108121941-192.168.
1.101-44433 (192.168.1.101:44433) with 2 core(s)
20/01/08 16:47:42 INFO StandaloneSchedulerBackend: Granted executor ID app-20200108164741-0000/1 on hostPort 192.168.1.101:44433 with
2 core(s), 2.6 GB RAM
20/01/08 16:47:42 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, spark-client, 33935, None)
20/01/08 16:47:42 INFO BlockManagerMasterEndpoint: Registering block manager spark-client:33935 with 265.5 MB RAM, BlockManagerId(driv
er, spark-client, 33935, None)
20/01/08 16:47:42 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, spark-client, 33935, None)
20/01/08 16:47:42 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, spark-client, 33935, None)
20/01/08 16:47:42 INFO StandaloneAppClient$ClientEndpoint: Executor updated: app-20200108164741-0000/1 is now RUNNING
20/01/08 16:47:42 INFO StandaloneAppClient$ClientEndpoint: Executor updated: app-20200108164741-0000/0 is now RUNNING
20/01/08 16:48:34 INFO DAGScheduler: Job 0 finished: reduce at /opt/spark/examples/src/main/python/pi.py:44, took 44.332327 s
Pi is roughly 3.141425
20/01/08 16:48:34 INFO SparkUI: Stopped Spark web UI at http://spark-client:4040
20/01/08 16:48:34 INFO StandaloneSchedulerBackend: Shutting down all executors
20/01/08 16:48:34 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Asking each executor to shut down
20/01/08 16:48:34 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
20/01/08 16:48:34 INFO MemoryStore: MemoryStore cleared
20/01/08 16:48:34 INFO BlockManager: BlockManager stopped
20/01/08 16:48:34 INFO BlockManagerMaster: BlockManagerMaster stopped
20/01/08 16:48:34 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
20/01/08 16:48:34 INFO SparkContext: Successfully stopped SparkContext
20/01/08 16:48:35 INFO ShutdownHookManager: Shutdown hook called
20/01/08 16:48:35 INFO ShutdownHookManager: Deleting directory /tmp/spark-1c0b3ca2-0e7c-467c-ab44-3844f496db8f/pyspark-908bc9ec-7c6a-4
497-9749-e18cdd4c8ec9
20/01/08 16:48:35 INFO ShutdownHookManager: Deleting directory /tmp/spark-508aadfc-c3dc-49aa-9bb6-943a42869588
20/01/08 16:48:35 INFO ShutdownHookManager: Deleting directory /tmp/spark-1c0b3ca2-0e7c-467c-ab44-3844f496db8f
(pyspark_env) hadoopuser@spark-client:~$
```

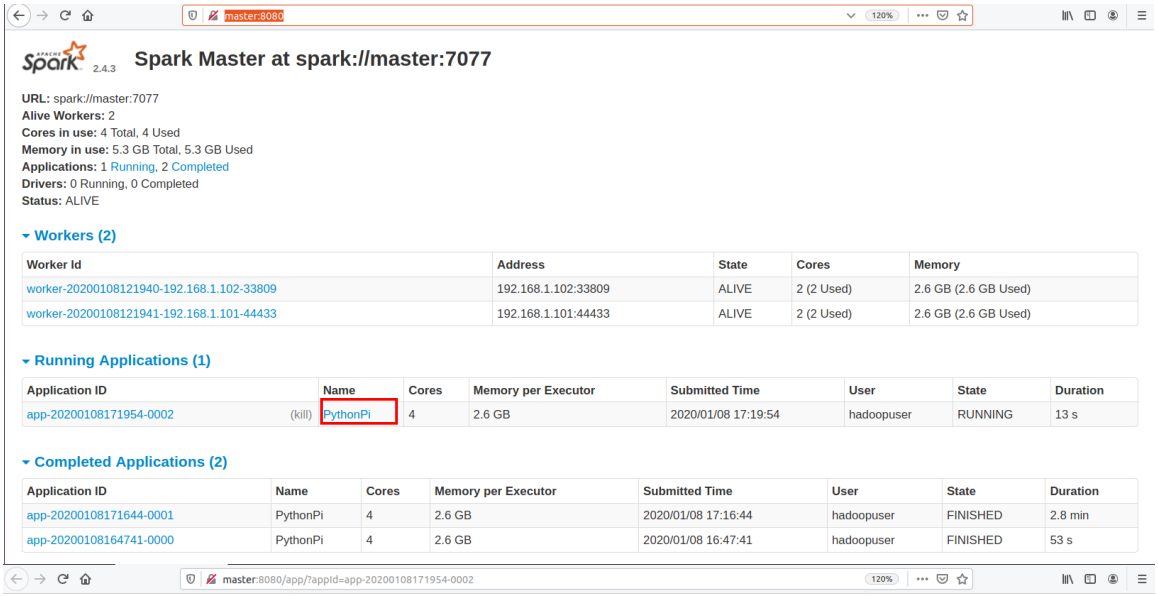
Σχήμα 6-172: Υποβολή του script pi.py μέσω του script spark-submit για επιβεβαίωση της ορθής λειτουργίας του Spark σε Standalone mode

Σουίτα Διεπαφών Χρήστη Web του Spark (Spark Web UIs)

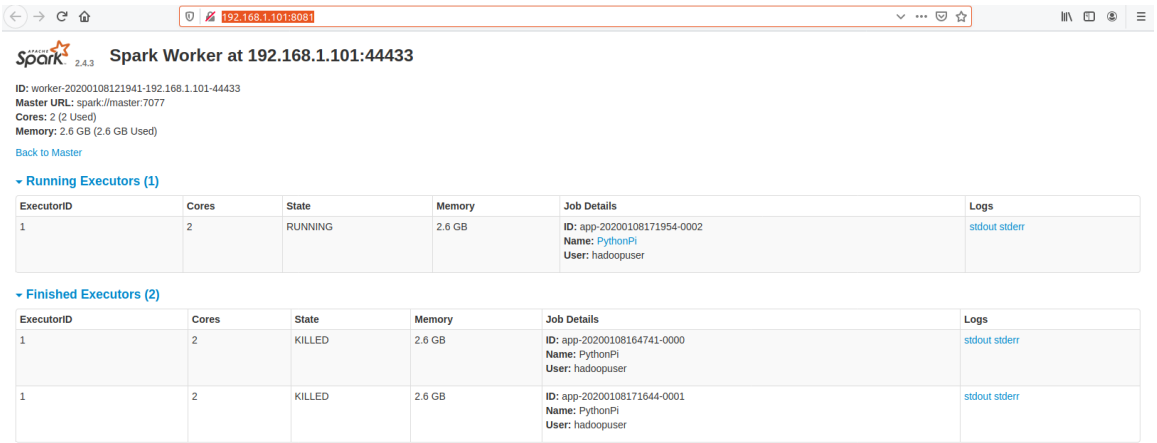
Με την προεπιλεγμένη διαμόρφωση του Spark, μπορεί να δει ο χρήστης αναλυτικές πληροφορίες για μια εφαρμογή όπως *χρονοδιάγραμμα συμβάντων (event timeline)*, *οπτική αναπαράσταση του κατευθυνόμενου άκυκλου γράφου (DAG) μιας εργασίας (job)*, *στάδια εργασίας* (job stages), έργα, *συνοπτικές μετρήσεις των έργων* (*summary tasks metrics*), συγκεντρωτικές μετρήσεις ανά εκτελεστή (Aggregated metrics by executor), *ονοματισμένοι συσσωρευτές* (*named accumulators*), κατανάλωση πόρων (μνήμη, πυρήνες) της συστάδας Spark κ.λπ. μόνο όμως για όσο χρονικό διάστημα τρέχει η εφαρμογή. Η πρόσβαση σ' αυτές τις πληροφορίες γίνεται μέσω της σουίτας διεπαφών χρήστη Spark (Web User Interfaces / Web UIs) η οποία παρέχει τις ακόλουθες διεπαφές χρήστη:

- *Master web UI*, προσβάσιμο μέσω της http://<master_ip>:8080 (βλέπε Σχήμα 6-173).
- *Worker web UI*, προσβάσιμο μέσω της http://<slave_ip>:8081 (βλέπε Σχήμα 6-174)
- *Application Detail web UI*, προσβάσιμο μέσω της http://<driver_ip>:4040 (βλέπε Σχήμα 6-175).

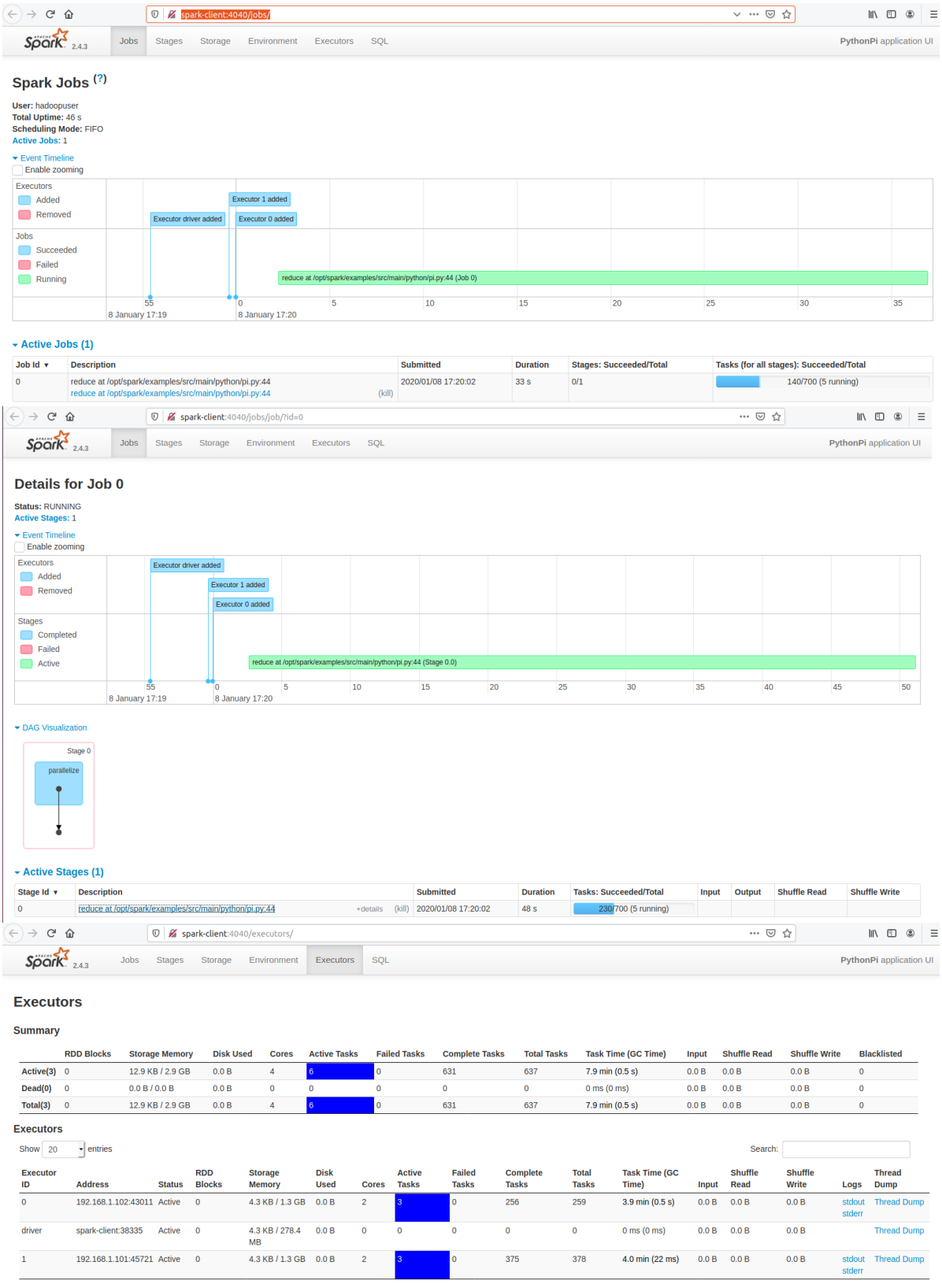
Κάθε εφαρμογή Spark μέσω του *SparkContext* εκκινεί το δικό της *web UI* στιγμιότυπο. Εάν εκτελούνται πολλαπλά *SparkContexts* στον ίδιο υπολογιστή, θα δεσμεύονται σε διαδοχικές θύρες ξεκινώντας από την 4040 (4041, 4042, κ.λπ.).



Σχήμα 6-173: Το Spark Master Web UI



Σχήμα 6-174: Το Spark Worker Web UI



Σχήμα 6-175: Μερικές από τις καρτέλες του Application Detail Web UI. Κάθε εφαρμογή Spark μέσω του SparkContext εκκινεί το δικό της web UI στιγμιότυπο.

Διακομιστής Ιστορικού Spark σε συστάδα Spark σε Standalone mode

Ο Διακομιστής Ιστορικού Spark επιτρέπει τον χρήστη να επανεξετάσει τις παραπάνω πληροφορίες και μετά το πέρας της εφαρμογής. Οι πληροφορίες αυτές είναι χρήσιμες για τον χρήστη στην προσπάθεια του να βελτιώσει την αποδοτικότητα του κώδικά του αλλά και για την ρύθμιση της συστάδας για βέλτιστη απόδοση (*performance tuning*). Υπάρχουν οι δυο παρακάτω περιπτώσεις συστάδας *Spark* σε *Standalone mode*:

A. Υποστηριζόμενη από το κατανεμημένο σύστημα αρχείων Hadoop HDFS.

Ο τρόπος διαμόρφωσης του Spark έτσι ώστε να είναι ενεργός ο Διακομιστής Ιστορικού Spark σε συστάδα που έχει εγκατεστημένο το κατανεμημένο σύστημα αρχείων Hadoop HDFS παρουσιάστηκε στην υποενότητα "*Διαδικτυακή διεπαφή χρήστη της εφαρμογής Spark (Spark Application Web UI) & ενεργοποίηση του Διακομιστή Ιστορικού Spark (Spark History Server) σε συστάδα Hadoop HDFS/YARN*". Παρακάτω παρουσιάζονται μόνο η διαμόρφωση των αρχείων spark-defaults.conf, ο τρόπος ενεργοποίησης της συστάδας *Spark* σε *Standalone mode* (βλέπε Σχήμα 6-176), του κατανεμημένου συστήματος αρχείων *Hadoop HDFS* (βλέπε Σχήμα 6-177) και του Διακομιστή Ιστορικού Spark (βλέπε Σχήμα 6-178). Για την διαμόρφωση των αρχείων του *Hadoop HDFS/YARN* θα πρέπει ο χρήστης να ανατρέξει στην προαναφερθείσα υποενότητα:

Αρχεία spark-defaults.conf

```
#------------------------------------------------------------------------------
# spark-client
# CLIENT SIDE - Spark UI PROPERTIES
#------------------------------------------------------------------------------
spark.eventLog.enabled                                  true

# Όταν η συστάδα Spark υποστηρίζεται από το κατανεμημένο σύστημα αρχείων Hadoop HDFS
spark.eventLog.dir                                      hdfs://master:9000/spark-logs

# Όταν η συστάδα Spark δεν υποστηρίζεται από το κατανεμημένο σύστημα αρχείων Hadoop
# HDFS
# η URL μπορεί να είναι μία τοπική (προκαθορισμένη τιμή - file:///tmp/spark-events
#spark.eventLog.dir                                     file:///opt/spark/spark-
events

# Η συμπίεση που θα χρησιμοποιήσει το Spark είναι αυτή της ιδιότητας Spark:
# spark.io.compression.codec - (προκαθορισμένη τιμή κωδικοποιητή: ο LZ4)
# (προκαθορισμένη τιμή: false)
# spark.eventLog.compress                               true
```

```
#------------------------------------------------------------------------------
# SERVER SIDE (εκεί όπου θα τρέχει ο Διακομιστής Ιστορικού Spark )
# spark-client, master, slave1, slave2
#------------------------------------------------------------------------------
-----
# Όταν η συστάδα Spark υποστηρίζεται από το κατανεμημένο σύστημα αρχείων Hadoop HDFS
spark.history.fs.logDirectory                       hdfs://master:9000/spark-logs

# Όταν η συστάδα Spark δεν υποστηρίζεται από το κατανεμημένο σύστημα αρχείων Hadoop
```

```
# HDFS
# η URL μπορεί να είναι μία τοπική (προκαθορισμένη τιμή - file:///tmp/spark-events)
#spark.history.fs.logDirectory                       file:///opt/spark/spark-events
#spark.history.fs.update.interval                    10s
#spark.history.ui.port                               18080
# Καθορίζει αν ο Διακομιστής Ιστορικού θα καθαρίζει τα αρχεία καταγραφής συμβάντων
# σε προκαθορισμένα χρονικά διαστήματα που καθορίζονται από την ιδιότητα
# spark.history.fs.cleaner.maxAge
spark.history.fs.cleaner.enabled   true
# Καθορίζει πόσο συχνά θα γίνεται έλεγχος από το σύστημα για την εκκαθάριση αρχείων
# καταγραφής συμβάντων. Δυνατές τιμές: 1d | 1h | 30m | 1000s
spark.history.fs.cleaner.interval  1d
# Δυνατές τιμές: 1d | 1h | 30m | 1000s
spark.history.fs.cleaner.maxAge    7d
```

Εκκίνηση στον κόμβο *master* του κατανεμημένου συστήματος αρχείων *Hadoop HDFS, του Spark σε Standalone mode και του Διακομιστή Ιστορικού Spark*[60]*:*

```
# master
#-------------------------------------------------------------------------------
# Εκκίνηση του κατανεμημένου συστήματος αρχείων Hadoop HDFS
start-dfs.sh
# Εκκίνηση της συστάδας Spark σε Standalone mode
start-master.sh --host master --port 7077
start-slaves.sh
# Εκκίνηση του Διακομιστή Ιστορικού Spark
start-history-server.sh[61]
```

```
(pyspark_env) hadoopuser@master:~$ start-master.sh --host master --port 7077
starting org.apache.spark.deploy.master.Master, logging to /opt/spark/logs/spark-hadoopuser-org.apache.spark.deploy.
master.Master-1-master.out
(pyspark_env) hadoopuser@master:~$ start-slaves.sh
slave2: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-hadoopuser-org.apache.spark
.deploy.worker.Worker-1-slave2.out
slave1: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-hadoopuser-org.apache.spark
.deploy.worker.Worker-1-slave1.out
```

Σχήμα 6-176: Εκκίνηση της συστάδας Spark σε Standalone mode

```
(pyspark_env) hadoopuser@master:~$ start-dfs.sh
Starting namenodes on [master]
Starting datanodes
Starting secondary namenodes [master]
(pyspark_env) hadoopuser@master:~$ jps
4803 NameNode
5001 SecondaryNameNode
5150 Jps
3087 Master
```

Σχήμα 6-177: Εκκίνηση του κατανεμημένου συστήματος αρχείων Hadoop HDFS

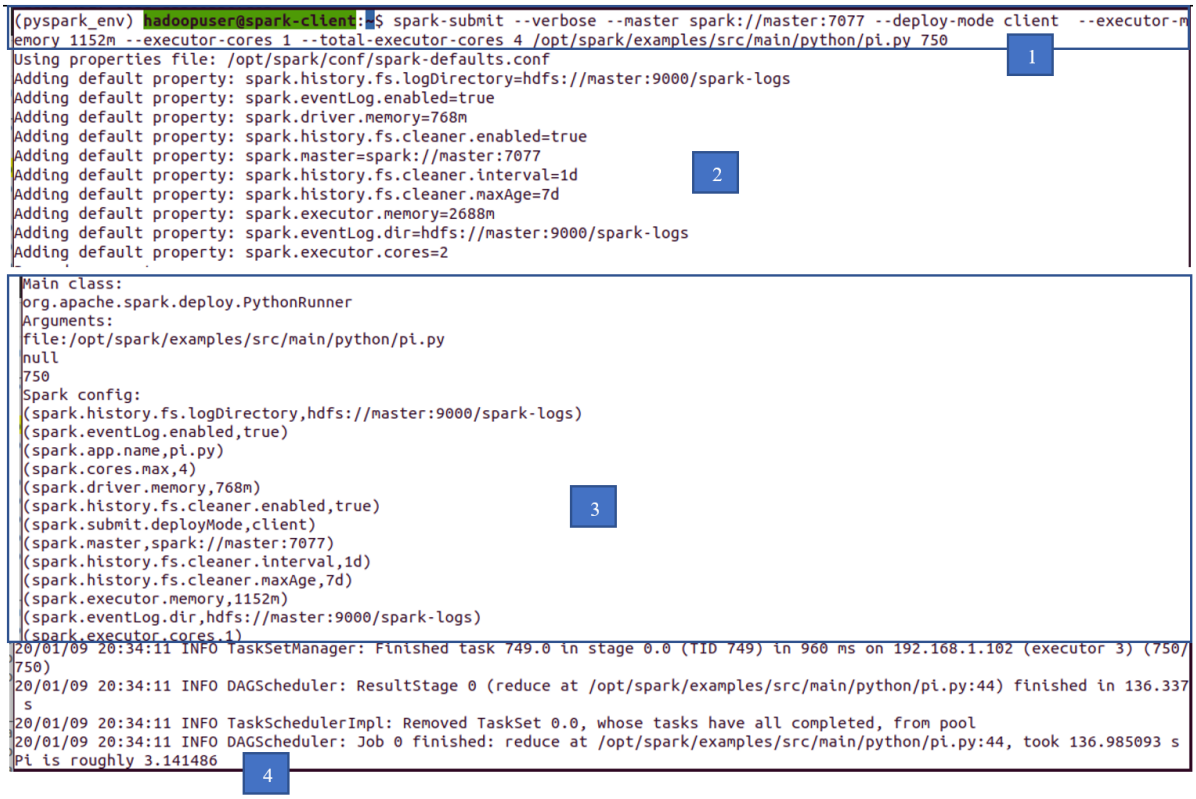
```
(pyspark_env) hadoopuser@master:~$ start-history-server.sh
starting org.apache.spark.deploy.history.HistoryServer, logging to /opt/spark/logs/spark-hadoopuser-org.apache.spark
.deploy.history.HistoryServer-1-master.out
(pyspark_env) hadoopuser@master:~$ jps
4803 NameNode
5892 HistoryServer
5269 Master
5926 Jps
5001 SecondaryNameNode
```

Σχήμα 6-178: Εκκίνηση του Διακομιστή Ιστορικού Spark στον κόμβο master

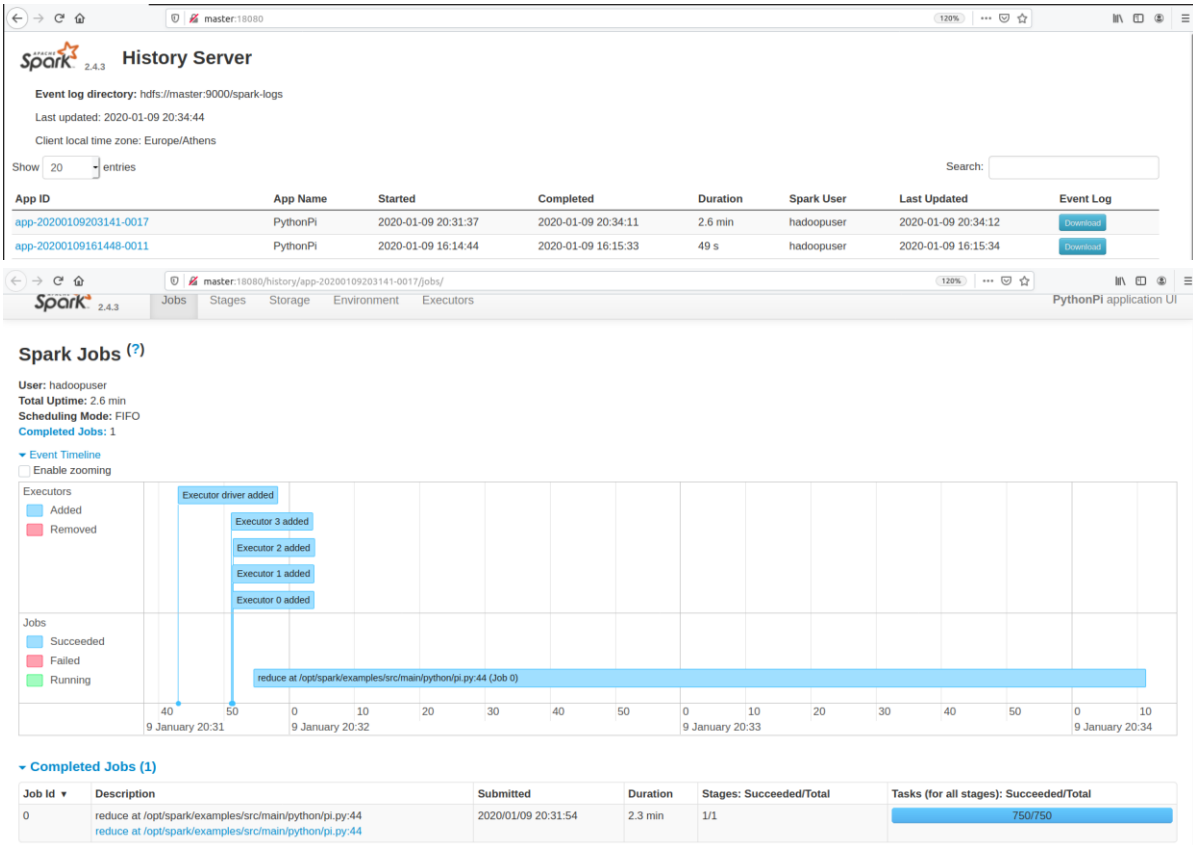
Θα πρέπει ο χρήστης στη συνέχεια να δοκιμάσει να ξανατρέξει το παράδειγμα *pi.py* σε *client mode* (βλέπε Σχήματα 6-179, 6-180).

[60] *Ο γράφων επέλεξε στην προκειμένη περίπτωση να ξεκινήσει τον Διακομιστή Ιστορικού στον κόμβο master.*
[61] *Το σταμάτημα του Διακομιστή Ιστορικού Spark γίνεται με το script stop-history-server.sh*

```
# spark-client
#-------------------------------------------------------------------------------------
spark-submit --verbose --master spark://master:7077 --deploy-mode client --executor-
memory 1152m --executor-cores 1 --total-executor-cores 4
/opt/spark/examples/src/main/python/pi.py 750
```



Σχήμα 6-179: (1) Το script spark-submit με κάποια επιπλέον command line options: --verbose (παρέχει μακροσκελείς / αναλυτικές πληροφορίες), --executor-memory (ποσότητα μνήμης για χρήση ανά διεργασία εκτελεστή), --executor-cores(αριθμός πυρήνων ανά εκτελεστή), --total-executor-cores (ο συνολικός αριθμός πυρήνων των εκτελεστών στην συστάδα ανά εφαρμογή Spark), (2) Οι ιδιότητες Spark του αρχείου διαμόρφωσης spark-defaults.conf, (3) Παρατηρούμε ότι τελικά επικράτησαν οι επιλογές της γραμμής εντολών (command line options) έναντι των ιδιοτήτων Spark, (4) Το αποτέλεσμα του script pi.py



Σχήμα 6-180: Ο Διακομιστής Ιστορικού Spark

B. Μη υποστηριζόμενη από κατανεμημένο σύστημα αρχείων.

Στην περίπτωση συστάδας Spark σε Standalone mode που δεν υποστηρίζεται από ένα κατανεμημένο σύστημα αρχείων (π.χ. HDFS) θα πρέπει ο χρήστης να δημιουργήσει έναν κατάλογο όπου θα αποθηκεύονται οι πληροφορίες καταγραφής συμβάντων όταν τρέχει μια εφαρμογή Spark. Η προεπιλογή του Spark είναι να αποθηκεύει τα αρχεία καταγραφής στη θέση */tmp/spark-events* (θα πρέπει να αναδημιουργεί ο χρήστης τον κατάλογο *spark-events* κάθε φορά που επιθυμεί την προσωρινή αποθήκευση των πληροφοριών καταγραφής συμβάντων). Ο γράφων επέλεξε να αποθηκεύονται τα αρχεία καταγραφής συμβάντων στην θέση */opt/spark/spark-events (βλέπε Σχήμα 6-181)* έτσι ώστε να μπορεί να ελέγχει το χρονικό διάστημα διατήρησης των αρχείων αυτών. *Θα πρέπει να τονιστεί ότι στην περίπτωση αυτή ο Διακομιστής Ιστορικού Spark τρέχει μόνο στον spark-client και θα πρέπει να ενημερωθεί μόνο το αρχείο διαμόρφωσης spark-defaults.conf στον spark-client.*

```
# spark-client
# Δημιουργία καταλόγου για την αποθήκευση των αρχείων καταγραφής συμβάντων των εφαρμογών από τον Διακομιστή Ιστορικού Spark
mkdir -p /opt/spark/spark-events
# Αλλαγή δικαιωμάτων στον κατάλογο spark-events
chmod 777 -R /opt/spark/spark-events
```

```
(pyspark_env) hadoopuser@spark-client:~$ mkdir -p /opt/spark/spark-events
(pyspark_env) hadoopuser@spark-client:~$ chmod 777 -R /opt/spark/spark-events
```

Σχήμα 6-181: Δημιουργία στον spark-client καταλόγου για την αποθήκευση των αρχείων καταγραφής συμβάντων των εφαρμογών από τον Διακομιστή Ιστορικού Spark και αλλαγή των δικαιωμάτων του.

```
# spark-defaults.conf
#_____________________________________________________________________________
# spark-client
# Ο Διακομιστής Ιστορικού Spark τρέχει στον client
#------------------------------------------------------------------------------
spark.eventLog.enabled                  true
spark.eventLog.dir                      file:///opt/spark/spark-events
#------------------------------------------------------------------------------
spark.history.fs.logDirectory           file:///opt/spark/spark-events
```

Στη συνέχεια θα πρέπει ο χρήστης να εκκινήσει τη συστάδα Spark σε Standalone mode (βλέπε Σχήμα 6-182):

```
# master
#------------------------------------------------------------------------------
# Εκκίνηση της συστάδας Spark σε Standalone mode
start-master.sh --host master --port 7077
start-slaves.sh
```

```
(pyspark_env) hadoopuser@master:~$ start-master.sh --host master --port 7077
starting org.apache.spark.deploy.master.Master, logging to /opt/spark/logs/spark-hadoopuser-org.apache.spark.deploy.master.Master-
1-master.out
(pyspark_env) hadoopuser@master:~$ start-slaves.sh
slave2: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-hadoopuser-org.apache.spark.deploy.worker
.Worker-1-slave2.out
slave1: /opt/spark/conf/spark-env.sh: line 1: 0#!/usr/bin/env: No such file or directory
slave1: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-hadoopuser-org.apache.spark.deploy.worker
.Worker-1-slave1.out
(pyspark_env) hadoopuser@master:~$ jps
10449 Jps
10392 Master
(pyspark_env) hadoopuser@master:~$ ssh slave1 jps
10356 Worker
10470 Jps
(pyspark_env) hadoopuser@master:~$ ssh slave2 jps
3124 Jps
3067 Worker
```

Σχήμα 6-182: Εκκίνηση της συστάδας Spark σε Standalone mode

Κατόπιν θα πρέπει να εκκινήσει τον Διακομιστή Ιστορικού Spark (βλέπε Σχήμα 6-183) και να τρέξει το πρόγραμμα pi.py (βλέπε Σχήμα 6-184):

```
#spark-client
#------------------------------------------------------------------------------
# Εκκίνηση του Διακομιστή Ιστορικού Spark
start-history-server.sh
# Υποβολή του script pi.py με το script spark-submit
spark-submit --verbose --master spark://master:7077 --deploy-mode client
/opt/spark/examples/src/main/python/pi.py 150
```

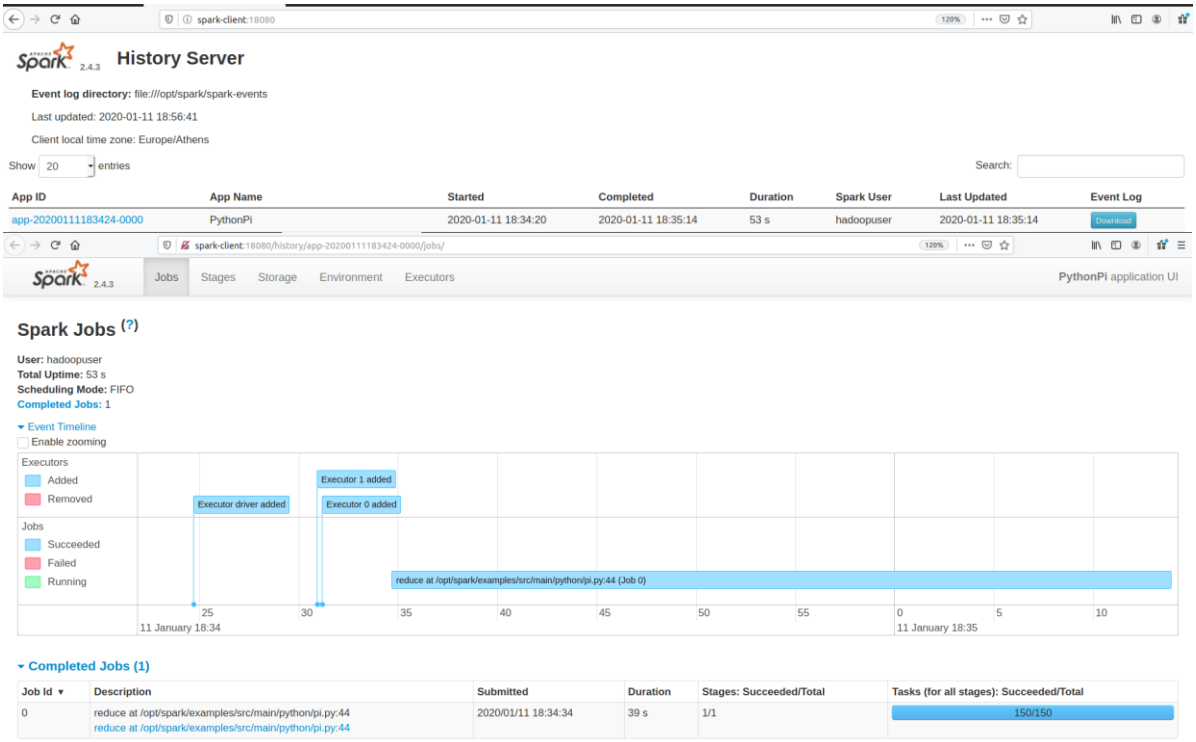
```
(pyspark_env) hadoopuser@spark-client:~$ start-history-server.sh
starting org.apache.spark.deploy.history.HistoryServer, logging to /opt/spark/logs/spark-hadoopuser-org.apache.spark.deploy.hist
.HistoryServer-1-spark-client.out
(pyspark_env) hadoopuser@spark-client:~$ jps
3972 HistoryServer
4012 Jps
```

Σχήμα 6-183: Εκκίνηση του Διακομιστή Ιστορικού Spark στον κόμβο spark-client

```
(pyspark_env) hadoopuser@spark-client:~$ spark-submit --verbose --master spark://master:7077 --deploy-mode client /opt/spark/examp
es/src/main/python/pi.py 150
Using properties file: /opt/spark/conf/spark-defaults.conf
Adding default property: spark.history.fs.logDirectory=file:///opt/spark/spark-events
Adding default property: spark.eventLog.enabled=true
Adding default property: spark.driver.memory=768m
Adding default property: spark.dynamicAllocation.maxExecutors=3
Adding default property: spark.master=spark://master:7077
Adding default property: spark.executor.memory=2688m
Adding default property: spark.eventLog.dir=file:///opt/spark/spark-events
Adding default property: spark.executor.cores=2
Adding default property: spark.dynamicAllocation.minExecutors=0
Adding default property: spark.dynamicAllocation.initialExecutors=1
```

…

Σχήμα 6-184: Υποβολή του script pi.py με το script spark-submit. Παρατηρούμε ότι διαβάζονται οι ιδιότητες του αρχείου διαμόρφωσης Spark spark-defaults.conf



Σχήμα 6-185: Απεικονίζονται μερικές από τις πληροφορίες του Διακομιστής Ιστορικού Spark που τρέχει στον spark-client

Δυναμική κατανομή πόρων σε συστάδα Spark με εγγενή διαχειριστή πόρων (Spark Standalone mode).

Αναλυτικά η δυναμική κατανομή πόρων στο Spark παρουσιάστηκε στην υποενότητα “Δυναμική κατανομή πόρων (Dynamic source allocation) στο Spark σε συστάδα Hadoop YARN”. Παρακάτω παρουσιάζεται η διαμόρφωση του αρχείου spark-defaults.conf τόσο στην πλευρά του client (spark-client) όσο και στην πλευρά των workers (slave1, slave2).

```
#_____________________________________________________________________________
# Στην πλευρά του client (spark-client)
#-----------------------------------------------------------------------------
# spark-defaults.conf
#-----------------------------------------------------------------------------
# Καθορίζει αν θα χρησιμοποιηθεί ή όχι η δυναμική κατανομή πόρων, η οποία κλιμακώνει
# (προς τα πάνω/κάτω) τον αριθμό των εκτελεστών που έχουν εγγραφεί με αυτήν την
# εφαρμογή, βάσει του φόρτου εργασίας.
```

```
spark.dynamicAllocation.enabled                                 true
spark.shuffle.service.enabled                                   true

# Μερικές προαιρετικές ιδιότητες:
# Λαμβάνονται υπόψη μόνο όταν είναι ενεργοποιημένη η Δυναμική Κατανομή
#------------------------------------------------------------------------------------
# Αρχικός αριθμός εκτελεστών
# (προκαθορισμένη τιμή: spark.dynamicAllocation.minExecutors)
spark.dynamicAllocation.initialExecutors                        1

# Ελάχιστος αριθμός εκτελεστών (προκαθορισμένη τιμή: 0)
spark.dynamicAllocation.minExecutors                            0

# Μέγιστος αριθμός εκτελεστών (προκαθορισμένη τιμή: infinity)
spark.dynamicAllocation.maxExecutors                            2

# Εάν ένας εκτελεστής έχει παραμείνει αδρανής για περισσότερο από αυτή τη διάρκεια,
# ο εκτελεστής θα καταργηθεί (προκαθορισμένη τιμή: 60s)
spark.dynamicAllocation.executorIdleTimeout                   60s

# Εάν υπάρχουν καθυστερημένες εργασίες που εκκρεμούν για περισσότερο από αυτή τη
# διάρκεια, θα ζητηθούν νέοι εκτελεστές (προκαθορισμένη τιμή:1s)
spark.dynamicAllocation.schedulerBacklogTimeout               5s

# Όπως η παραπάνω ιδιότητα αλλά χρησιμοποιείται μόνο για μεταγενέστερες αιτήσεις
# εκτελεστή (προκαθορισμένη τιμή: spark.dynamicAllocation.schedulerBacklogTimeout)
spark.dynamicAllocation.sustainedSchedulerBacklogTimeout   5s
```

```
#_____________________________________________________________________________
# Στην πλευρά των workers (slave1, slave2) - Shuffle behavior
#_____________________________________________________________________________
# Καθορίζει εάν θα ρυθμιστεί μια εξωτερική υπηρεσία shuffle, η οποία θα διατηρεί τα
# γραμμένα από τους εκτελεστές αρχεία shuffle ώστε οι εκτελεστές να μπορούν να
# αφαιρεθούν με ασφάλεια:
#*****************************************************************************

spark.shuffle.service.enabled                                   true
```

Ο έλεγχος ορθής λειτουργίας της Δυναμικής Κατανομής Πόρων θα γίνει στην επόμενη ενότητα.

Το Pyspark μέσω του Jupyter Notebook σε Αυτόνομη συστάδα Spark[62] (Spark Standalone mode)

Αναλυτικές πληροφορίες για το Jupyter Notebook και τον τρόπο εγκατάστασής του σε εικονικό περιβάλλον conda (virtual environment) παρέχονται στην υποενότητα "Βήμα 5ο: Το PySpark μέσω του Jupyter Notebook" της ενότητας " Εγκατάσταση του PySpark στο Ubuntu 19.04 σε τοπική λειτουργία (local mode)". Στην υποενότητα αυτή εγκαταστάθηκε το Jupyter Notebook στο περιβάλλον conda *pyspark_env*. Το περιβάλλον conda *pyspark_env* αναδημιουργήθηκε στην υποενότητα "*Λήψη και εγκατάσταση του Spark (σε μια συστάδα Hadoop Yarn*)". Συνεπώς υπάρχουν όλα τα προαπαιτούμενα για να μπορεί ο χρήστης να τρέξει το Jupyter. Το εικονικό περιβάλλον conda *pyspark_env* επειδή έχει τα ίδια εγκατεστημένα

[62] *Με εγγενή διαχειριστή πόρων*

πακέτα σε όλους τους κόμβους της συστάδας επιτρέπει στον χρήστη να χρησιμοποιήσει στο πρόγραμμά του όλα αυτά τα πακέτα χωρίς να απαιτείται η ενσωμάτωση τους μέσω zip ή egg[63] αρχείων με το script *spark_submit*. Με ένα απλό script μπορεί ο διαχειριστής της συστάδας να ενημερώνει το περιβάλλον pyspark_env με νέα πακέτα σε όλους τους κόμβους της συστάδας από τον κόμβο master.

Επειδή έχει ήδη ενσωματωθεί η ενεργοποίηση του περιβάλλοντος conda *pyspark_env*[64] κάθε φορά που ανοίγει η γραμμή εντολών το μόνο που απαιτείται είναι η εκκίνηση του *Jupyter notebook*. Στο Jupyter Notebook δημιουργήθηκαν δύο σημειωματάρια[65] (notebooks) (βλέπε Σχήματα 6-186, 6-187). Τα σημειωματάρια αυτά περιέχουν μια τροποποίηση του script pi.py προκειμένου να ελεγχθεί η λειτουργία της δυναμικής Κατανομής Πόρων στη συστάδα Spark σε Standalone mode. Κάθε εφαρμογή Spark (στιγμιότυπο του SparkContext) εκτελεί ένα ανεξάρτητο σύνολο διεργασιών εκτελεστών. Κατά τη δημιουργία ενός στιγμιότυπου (βλέπε Σχήματα 6-186, 6-187, κελί In[2] των Jupyter notebooks) του *SparkSession* (*SparkSession Instance*) μέσω της μεθόδου *SparkSession.build μπορεί ο χρήστης να* υποβάλλει μέσω αλυσιδωτών μεθόδων (method chaining) config διάφορες επιλογές διαμόρφωσης (configuration options) του Spark που τροποποιούν τις προεπιλεγμένες επιλογές του Spark που περιλαμβάνονται ως ιδιότητες στο αρχείο διαμόρφωσης *spark-defaults.conf*. Οι επιλογές αυτές παρουσιάζονται συγκεντρωτικά στον Πίνακα 6-6 για να είναι ευκολότερη η σύγκρισή τους. Στον πίνακα αυτό παρουσιάζονται με έντονα γράμματα οι ρητές ανά εφαρμογή επιλογές Spark που θα διαμορφώσουν τελικά τη συστάδα Spark. Με τη μέθοδο appName (“name”) καθορίστηκε το όνομα[66] το οποίο θα εμφανίζει η κάθε εφαρμογή στο Spark web UI (βλέπε Σχήμα 6-188.1). Το στιγμιότυπο του SparkSession αποδόθηκε στη μεταβλητή spark (βλέπε Σχήματα 6-186, 6-187, κελί In[2] των Jupyter notebooks).

[63] *Τα αρχεία egg είναι μια παλαιότερη μορφή διανομής Python πακέτων (η νέα μορφή διανομής πακέτων Python ονομάζεται wheel).*
[64] *Ο γράφων εισήγαγε το* script ***conda activate pyspark_env*** *μέσα στο ~/.bashrc αρχείο, σε όλους τους κόμβους της συστάδας και στον spark-client.*
[65] *Τα SparkStandaloneDynAllocTest1.ipynb και SparkStandaloneDynAllocTest2.ipynb.*
[66] *ian_jupyter_DynAlloc1 και iannis _jupyter_DynAlloc2 αντίστοιχα.*

Πίνακας 6-6: Οι επιλογές Spark που ορίζονται μέσω του SparkSession έχουν μεγαλύτερη προτεραιότητα απ΄ αυτές που ορίζονται μέσα στο αρχείο διαμόρφωσης spark-defaults.conf. Στον πίνακα παρουσιάζεται με έντονα μαύρα γράμματα η τελική Κατανομή Πόρων στις εφαρμογές (ian_jupyter_DynAlloc1, iannis_jupyter_DynAlloc2) του παραδείγματος της Δυναμικής Κατανομής πόρων σε αυτόνομη[67] συστάδα Spark (Spark Standalone mode)

App Name	spark-defaults.conf	SparkSession.Builder's config Method
ian_jupyter_ DynAlloc1	**spark.dynamicAllocation.initialExecutors 1** **spark.dynamicAllocation.minExecutors 0** **spark.dynamicAllocation.maxExecutors 2** spark.executor.cores 2 spark.executor.memory 2688m	.config("**spark.executor.cores**", "**1**") .config("**spark.executor.memory**", "**1152m**")
iannis_jupyter_ DynAlloc2	**spark.dynamicAllocation.initialExecutors 1** **spark.dynamicAllocation.minExecutors 0** spark.dynamicAllocation.maxExecutors 2 spark.executor.cores 2 spark.executor.memory 2688m	.config("**spark.executor.cores**", "**1**") .config("**spark.executor.memory**", "**1152m**") .config("**spark.dynamicAllocation.maxExecutors**", "**3**")

Η εφαρμογή με appName: ian_jupyter_DynAlloc1 (βλέπε Σχήμα 186), εκτελεί άμεσα την συνάρτηση υπολογισμού του π. Ο υπολογισμός του π εκτελείται δύο φορές μέσα στη συνάρτηση. Την πρώτη με partitions=1000 και την δεύτερη με partitions =1900. Μεταξύ των δυο επαναλήψεων του υπολογισμού εισάγεται μια καθυστέρηση 100s. Η εφαρμογή με appName: iannis_jupyter_DynAlloc2 (βλέπε Σχήμα 187), εκτελεί την συνάρτηση υπολογισμού του π με μια καθυστέρηση 70s. Ο πρώτος επαναληπτικός υπολογισμός του **π** γίνεται για partitions =1500 ενώ ο δεύτερος για partitions =2100 (βλέπε Σχήμα 6-188, Job ID:0 και Job ID:1 αντίστοιχα) . Η εφαρμογή εισάγει μεταξύ των δυο επαναλήψεων μια καθυστέρηση 120s.

Όπως φαίνεται από τα χρονοδιαγράμματα γεγονότων (event timeline, βλέπε Σχήμα 6-188) των δυο εφαρμογών με την εκκίνηση της πρώτης εργασίας (Job Id:0) προσαρτάται αμέσως ο ένας από τους δυο εκτελεστές λόγω της ιδιότητας Spark *spark.dynamicAllocation.initialExecutors =1* που καθορίζει τον αριθμό των εκτελεστών κατά την εκκίνηση της εφαρμογής. Επειδή στην πρώτη εφαρμογή υπάρχουν στην ουρά αναμονής των έργων, 1000 καθυστερημένα έργα που εκκρεμούν για περισσότερο από το χρονικό διάστημα που ορίζεται στην ιδιότητα Spark *spark.dynamicAllocation.schedulerBacklogTimeout=5s* και λόγω της ιδιότητας Spark

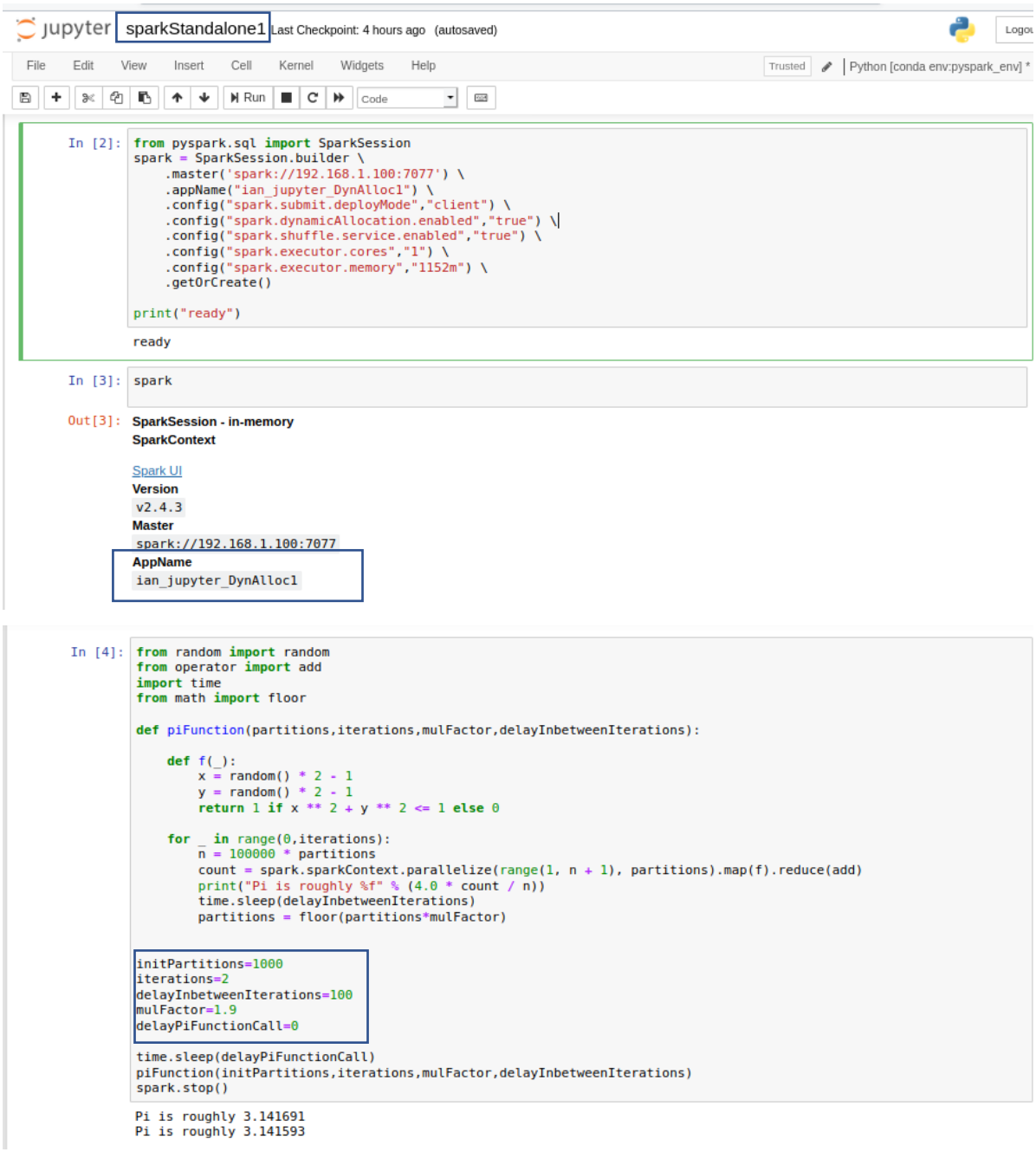
[67] *Με εγγενή διαχειριστή πόρων Spark*

spark.dynamicAllocation.maxExecutors=2 ζητείται νέος εκτελεστής [68] [42] από τον Διαχειριστή Πόρων ο οποίος και εκχωρείται. Αντίθετα στην δεύτερη εφαρμογή επειδή ο εκτελεστής (Executor 0) έμεινε αδρανής για διάστημα μεγαλύτερο από αυτό που ορίζεται στην ιδιότητα Spark *spark.dynamicAllocation.executorIdleTimeout=60s* και λόγω της ιδιότητας Spark *spark.dynamicAllocation.minExecutors=0* επιστρέφεται [69] πίσω στον διαχειριστή πόρων (βλέπε ετικέτα 2 στο Σχήμα 6-188). Μόλις όμως υποβληθεί η εργασία με Job Id:0 και επειδή η εφαρμογή Spark δεν έχει κανένα εκτελεστή ενώ στην ουρά αναμονής των έργων (tasks) υπάρχουν 1500 έργα, ζητά την εκχώρηση νέων εκτελεστών, με τη διαδικασία που περιγράφηκε παραπάνω (έναν κάθε 5s). Αντίθετα όμως με την πρώτη εφαρμογή λόγω της τιμής που έχει ορισθεί στην ιδιότητα Spark *spark.dynamicAllocation.maxExecutors=3* η εφαρμογή ζητά από τον Διαχειριστή Πόρων έναν επιπλέον εκτελεστή το αίτημα όμως απορρίπτεται λόγω έλλειψης διαθέσιμων πόρων (εκτελεστών). Όταν τελειώσει η εκτέλεση της πρώτης εργασίας στην πρώτη εφαρμογή επιστρέφονται όλοι οι εκτελεστές από την εφαρμογή [42] λόγω της ιδιότητας Spark (*spark.dynamicAllocation.minExecutors=0*) μετά από αδράνειά τους για χρονικό διάστημα μεγαλύτερο των 60s (*spark.dynamicAllocation.executorIdleTimeout= 60s*) και ο ένας απ' αυτούς εκχωρείται στην δεύτερη εφαρμογή διότι υπάρχουν αρκετά έργα στην ουρά αναμονής. Μετά από συνολική καθυστέρηση 100s (βλέπε delayInbetweenIterations=100 στο Σχήμα 6-186 και ετικέτα 3 στο Σχήμα 6-188) ξεκινά η δεύτερη εργασία (Job Id:1) της πρώτης εφαρμογής. Η διαδικασία δυναμικής κατανομής πόρων επαναλαμβάνεται όμως αυτή την φορά ο Διαχειριστής εκχωρεί στην εφαρμογή έναν μόνο εκτελεστή λόγω έλλειψης διαθέσιμων εκτελεστών. Ενώ όταν ολοκληρωθεί η πρώτη εργασία της δεύτερης εφαρμογής ο Διαχειριστής Πόρων επιστρέφει τους αδρανείς εκτελεστές μέχρι να φτάσει στον ελάχιστο αριθμό εκτελεστών που έχει ρυθμιστεί να χρησιμοποιεί (*spark.dynamicAllocation.minExecutors=0)* μετά από διάστημα 60s (*spark.dynamicAllocation.executorIdleTimeout= 60s*) και εκχωρεί τον έναν από τους διαθέσιμους εκτελεστές στην δεύτερη εργασία της πρώτης εφαρμογής. Μετά από συνολική καθυστέρηση 120s (βλέπε delayInbetweenIterations=120 στο Σχήμα 6-187 και ετικέτα 4 στο Σχήμα 6-188) ξεκινά η δεύτερη εργασία (Job Id:1) της δεύτερης εφαρμογής. Ο Διαχειριστής εκχωρεί στην εφαρμογή δύο μόνο εκτελεστές λόγω έλλειψης διαθέσιμων εκτελεστών. Μόλις ολοκληρωθεί

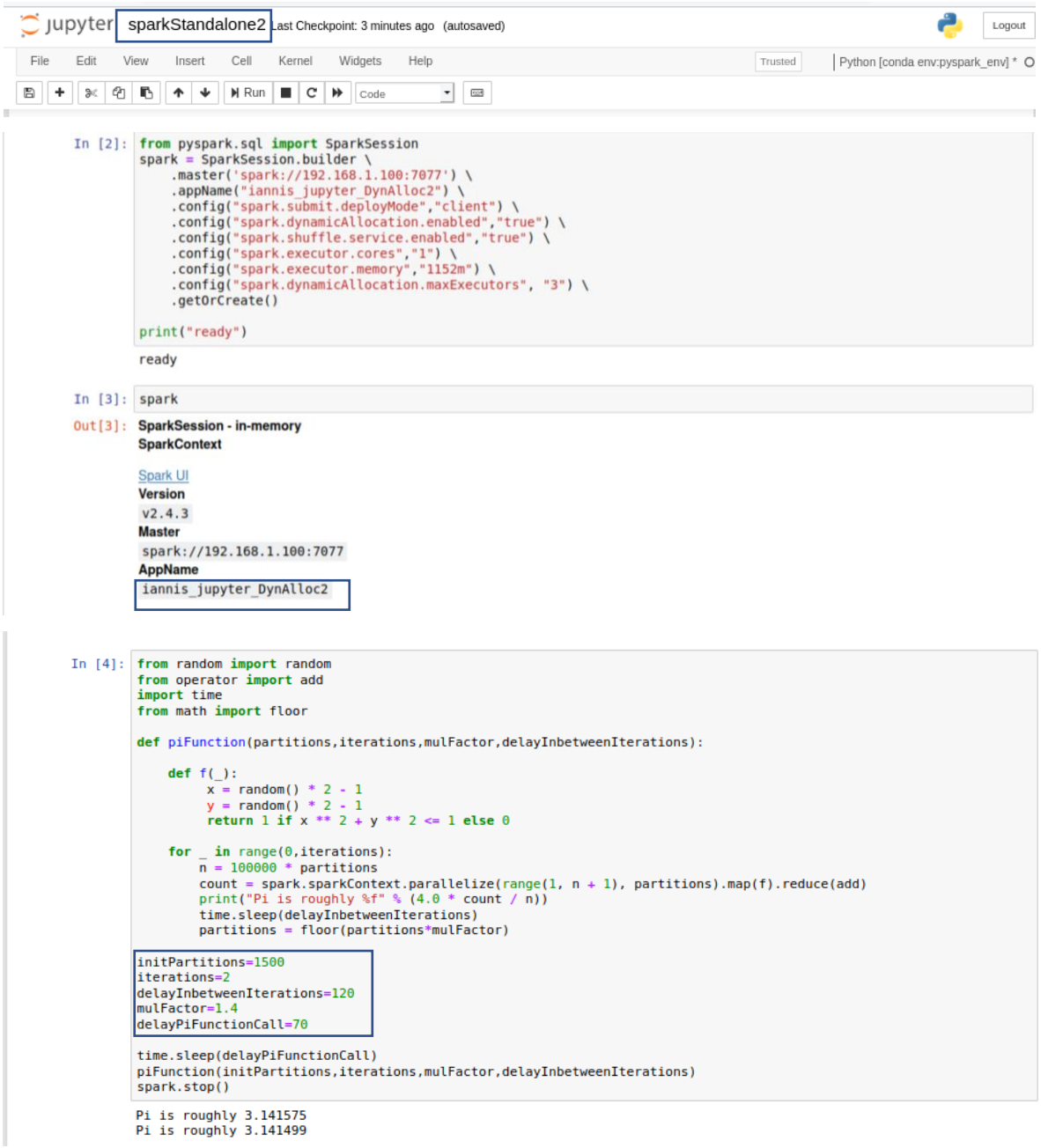
[68] *Ο αριθμός των εκτελεστών που ζητούνται σε κάθε γύρο αυξάνεται εκθετικά από τον προηγούμενο γύρο. Για παράδειγμα, μια εφαρμογή θα προσθέσει 1 εκτελεστή στον πρώτο γύρο, και στη συνέχεια 2, 4, 8 και ούτω καθεξής εκτελεστές στους επόμενους γύρους έως ότου επιτευχθεί ένα ανώτατο όριο που εξαρτάται τόσο από την ιδιότητα διαμόρφωσης του Spark spark.dynamicAllocation.maxExecutors όσο και από τον τρέχοντα αριθμό των έργων (tasks) που εκτελούνται αλλά και αυτών που εκκρεμούν.*

[69] *Ωστόσο, αν ο εκτελεστής που πρόκειται να αφαιρεθεί έχει αποθηκεύσει κάποια block στην cache του, θα αφαιρεθεί μετά από το χρόνο που καθορίζεται στην παράμετρο spark.dynamicAllocation.cachedExecutorIdleTimeout (υπό την προϋπόθεση ότι έχει παραμείνει αδρανής για μεγαλύτερο διάστημα από αυτή τη διάρκεια)[30].*

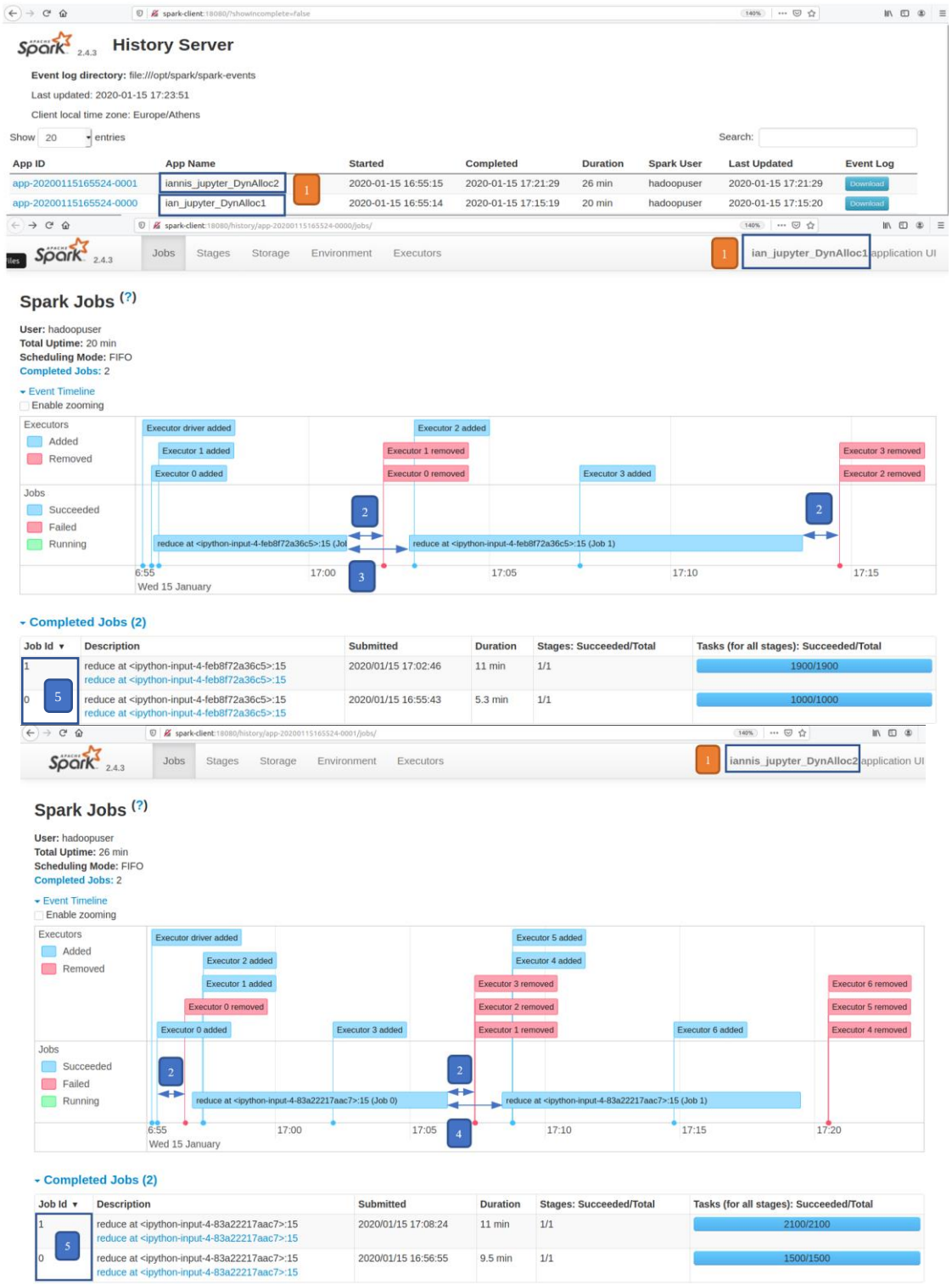
και η δεύτερη εργασία της πρώτης εφαρμογής επιστρέφονται στον Διαχειριστή Πόρων από την εφαρμογή όλοι οι εκτελεστές (*spark.dynamicAllocation.minExecutors=0*) λόγω αδράνειάς τους για χρονικό διάστημα μεγαλύτερο των 60s (*spark.dynamicAllocation.executorIdleTimeout= 60s*) ο οποίος και εκχωρεί τον έναν από τους διαθέσιμους εκτελεστές στην δεύτερη εργασία της δεύτερης εφαρμογής. Ενώ όταν ολοκληρωθεί και η δεύτερη εργασίας της δεύτερης εφαρμογής ο Διαχειριστής Πόρων επιστρέφει τους αδρανείς εκτελεστές μέχρι να φτάσει στον ελάχιστο αριθμό εκτελεστών που έχει ρυθμιστεί να χρησιμοποιεί (*spark.dynamicAllocation.minExecutors=0*) μετά από διάστημα 60s (*spark.dynamicAllocation.executorIdleTimeout= 60s*). Διαπιστώνεται λοιπόν ότι η Δυναμική Κατανομή Πόρων λειτούργησε με τον τρόπο που αναμενόταν.



Σχήμα 6-186: Η εφαρμογή με AppName: ian_jupyter_DynAlloc1, εκτελεί άμεσα την συνάρτηση υπολογισμού του **π.** Ο υπολογισμός του **π** εκτελείται δύο φορές μέσα στη συνάρτηση. Την πρώτη με partitions=1000 και την δεύτερη με partitions =1900. Μεταξύ των δυο επαναλήψεων του υπολογισμού εισάγεται μια καθυστέρηση 100s. Η τελική διαμόρφωση των πόρων που χρησιμοποιεί η εφαρμογή φαίνεται στον Πίνακα 6-6.



Σχήμα 6-187: Η εφαρμογή με AppName: iannis_jupyter_DynAlloc2, εκτελεί την συνάρτηση υπολογισμού του π με μια καθυστέρηση 70s. Ο υπολογισμός του π εκτελείται δύο φορές μέσα στη συνάρτηση. Την πρώτη με partitions=1500 και την δεύτερη με partitions =2100. Μεταξύ των δυο επαναλήψεων του υπολογισμού εισάγεται μια καθυστέρηση 120s. Η τελική διαμόρφωση των πόρων που χρησιμοποιεί η εφαρμογή φαίνεται στον Πίνακα 6-6.



Σχήμα 6-188: Οι δυο εφαρμογές δημιουργούν ταυτόχρονα το αντικείμενο spark (από την κλάση SparkSession) και μέσω αυτού το αντικείμενο SparkContext στο κύριο πρόγραμμα (που ονομάζεται πρόγραμμα οδήγησης. Η αρχική εργασία (Job ID:0) της εφαρμογής iannis_jupyter_DynAlloc2 υποβάλλεται με καθυστέρηση (delayPiFunctionCall) 70s σε σχέση με την αντίστοιχη εργασία της εφαρμογής ian_jupyter_DynAlloc1 (βλέπε τον χρόνο υποβολής (submitted) του Job Id 0 και για τις δυο εφαρμογές).

1: Τα ονόματα των δυο εφαρμογών για το Spark UI, 2:Μετά από χρόνο αδράνειας 60s (spark.dynamicAllocation.executorIdleTimeout) ο εκτελεστής καταργείται. 3,4:Η καθυστέρηση ανάμεσα στις εργασίες (delayInbetweenIterations) 100s και 120s αντίστοιχα. 5: Job Ids .

Παράρτημα

Βιβλιογραφία

[1]	Chambers B., Zaharia M. (2018), “Spark - The Definitive Guide - Big data processing made simple”. O’Reilly Media, Inc., Sebastopol, CA
[2]	Karau H., Warren R. (2017), “High Performance Spark: Best Practices for Scaling and Optimizing Apache Spark”. O’Reilly Media, Inc., Sebastopol, CA.
[3]	Zečević P., Bonaći M. (2017), “Spark in Action”. Manning Publications Co., Shelter Island
[4]	Feng W. (2019), “Learning Apache Spark with Python” [Online]. Available from: https://runawayhorse001.github.io/LearningApacheSpark/pyspark.pdf
[5]	Singh P., (2019). “Machine Learning with PySpark With Natural Language Processing and Recommender Systems”. Apress
[6]	Garillot F., Maas G. (2017), “Learning Spark Streaming”. O’Reilly Media, Inc., Sebastopol, CA.
[7]	Luu H., (2018), “Beginning Apache Spark 2: With Resilient Distributed Datasets, Spark SQL, Structured Streaming and Spark Machine Learning library”. Apress
[8]	Frampton M. (2018), “Complete Guide to Open Source Big Data Stack”. Apress
[9]	Ankam V. (2016), “Big Data Analytics”. Packt Publishing. Birmingham, UK
[10]	McDonald C. (2018), “Getting Started with Apache Spark from Inception to Production” ebook. MapR Technologies, Inc. Santa Clara, CA
[11]	Dushesnay E. (2018), “*Statistics and Machine Learning in. Python”*. Release 0.2. [Online]. Available from: http://120.107.155.180/download/Python/Statistics%20and%20Machine%20Learning%20in%20Python.pdf
[12]	Desjardins J. (2019), *“How much data is generated each day?”*, World Economic Forum [Online], 17 April. Available from: https://www.weforum.org/agenda/2019/04/how-much-data-is-generated-each-day-cf4bddf29f (Accessed 30 July 2019)
[13]	Reinsel D., et al. (2018), “Data Age 2025: The digitization of the world from edge to core”. I.D. Corporation, Editor. p. 28 [Online]. Available from:

	https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf (Accessed 30 July 2019)
[14]	IBM Corporation (2013), *"Veracity of Data"*, Think Finance [Online], 17 April. Available from: https://www.ibm.com/blogs/insights-on-business/ibmix/wp-content/uploads/sites/6/2015/07/veracityofdatafinance_download1.pdf (Accessed 30 July 2019)
[15]	Tejshri A., (2019), "Data Lakes Market 2018–2025". Data Driven Investor [Online], 11 March. Available from: https://medium.com/datadriveninvestor/data-lakes-market-2018-2025-top-key-players-like-microsoft-informatica-teradata-capgemini-b6f6a86e2fc (Accessed 29 July 2019)
[16]	Watson IoT (2017), "Descriptive, predictive, prescriptive: Transforming asset and facilities management with analytics". IBM - Watson Internet of Things [Online], December. Available from: https://www.ibm.com/downloads/cas/3V9AA9Y5
[17]	Woodward A., et al. (2019), "Forecast Snapshot: Prescriptive Analytics Software, Worldwide, 2019" [Online], 23 January. Available from: https://www.gartner.com/en/documents/3899065/forecast-snapshot-prescriptive-analytics-software-worldw (Accessed 2 August)
[18]	Abu-Mostafa, Y. S., Magdon-Ismail, M., & Lin, H. (2012), "Learning from data: A short course". Seattle, WA: AMLbook.com.
[19]	N. Vaidya (2019). "Apache Spark Architecture – Spark Cluster Architecture Explained" 22 May. [Online] 22 May. Available from: https://www.edureka.co/blog/spark-tutorial/ (Accessed 19 January 2020)
[20]	Duvvuri, S., Singhal B. (2016), "Spark for Data Science". Packt Publishing. Birmingham, UK
[21]	Sinha. S. (2019), "Hadoop Tutorial: All you need to know about Hadoop!". [Online] 22 May. Available from: https://www.edureka.co/blog/hadoop-tutorial (Accessed 13 August)
[22]	Holden, K., et al. (2015), "Spark - Learning Spark- Lightning fast data analysis". O'Reilly Media, Inc., Sebastopol, CA
[23]	Apache Spark - [Online] 4 September 2019. Available from: https://spark.apache.org (Accessed 4 September 2019)

[24]	Robert D. Schneider, R., Karmiol , J. (2019). “Spark For Dummies, 2nd IBM Limited Edition”. John Wiley & Sons, Inc., Hoboken, NJ
[25]	Damji, J. (2016). “A Tale of Three Apache Spark APIs: RDDs vs DataFrames and Datasets -When to use them and why” [Online] 14 July. Available from: https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html (Accessed 17 August 2019)
[26]	Sakellis, K. (2015)."Why your Spark Job is Failing". Cloudera. DataWorks Summit. [Online] 19 Jun 2015. Available from: https://www.slideshare.net/Hadoop_Summit/why-your-spark-job-is-failing (Accessed 15 August 2019)
[27]	Breindel, A. (n.d.) “Transformations and Actions. A visual guide of the API” Databricks. Available from: https://training.databricks.com/visualapi.pdf (Accessed 17 August 2019)
[28]	Wikipedia, “Anaconda (Python distribution)” [Online]. Available from: https://en.wikipedia.org/wiki/Anaconda_(Python_distribution) (Accessed 23 August 2019)
[29]	Project Jupyter (n.d.), “The Jupyter Notebook” [Online]. Available from: https://jupyter.org/ (Accessed 25 August 2019)
[30]	Apache Spark (n.d), “Spark Configuration” [Online]. Available from: https://spark.apache.org/docs/latest/configuration.html (Accessed 14 December 2019)
[31]	Apache Spark (n.d), “Submitting Applications” [Online]. Available from: https://spark.apache.org/docs/latest/submitting-applications.html (Accessed 4 September 2019)
[32]	Learning Journal (n.d), “Apache Spark Foundation Course - Spark Architecture Part-1” [Online]. Available from: https://www.learningjournal.guru/courses/spark/spark-foundation-training/spark-architecture-part-1/ (Accessed 4 December 2019)
[33]	Apache Spark (n.d), “RDD Programming Guide” [Online]. Available from: http://spark.apache.org/docs/latest/rdd-programming-guide.html (Accessed 4 December 2019)
[34]	Apache Hadoop (2019), “Overview (File System Shell)” (10 September) [Online]. Available from: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html (Accessed 4 December 2019)

[35]	Apache Hadoop (2019), "HDFS Users Guide" (10 September) [Online]. Available from: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html (Accessed 24 December 2019)
	World Health Organisation 2013, Financial crisis and global health, The United Nations, accessed 1 August 2013, <http://www.who.int/topics/financial_crisis/en/>.
[36]	Apache Spark (n.d), "Running Spark on YARN" [Online]. Available from: https://spark.apache.org/docs/latest/running-on-yarn.html (Accessed 17 December 2019)
[37]	Sankar, K. (2016), "Fast Data Processing with Spark 2" Third Edition. Packt Publishing. Birmingham - Mumbai, UK
[38]	Cloudera (2018), "Determining HDP Memory Configuration Settings" [Online]. Available from: https://docs.cloudera.com/HDPDocuments/HDP2/HDP-2.6.5/bk_command-line-installation/content/determine-hdp-memory-config.html (Accessed 17 November 2019)
[39]	"Distribution of Executors, Cores and Memory for a Spark Application running in Yarn" (n.d) [Online]. Available from: https://spoddutur.github.io/spark-notes/distribution_of_executors_cores_and_memory_for_spark_application.html (Accessed 17 December 2019)
[40]	IBM (n.d), "Apache Spark configuration options " [Online]. Available from: https://www.ibm.com/support/knowledgecenter/en/SS3H8V_1.1.0/com.ibm.izoda.v1r1.azka100/topics/azkic_r_memcpuconfigopts.htm (Accessed 19 December 2019)
[41]	Apache Spark (n.d), "Monitoring and Instrumentation" [Online]. Available from: https://spark.apache.org/docs/latest/monitoring.html (Accessed 17 December 2019)
[42]	Apache Spark (n.d), "Job Scheduling" [Online]. Available from: https://spark.apache.org/docs/latest/job-scheduling.html (Accessed 3 January 2020)
[43]	Apache Spark (n.d), "Spark Standalone Mode" [Online]. Available from: https://spark.apache.org/docs/latest/spark-standalone.html (Accessed 3 January 2020)