ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΣΥΓΚΡΙΤΙΚΗ ΜΕΛΕΤΗ APACHE HADOOP ΜΕ APACHE SPARK ΓΙΑ ΕΠΕΞΕΡΓΑΣΙΑ ΜΕΓΑΛΩΝ ΔΕΔΟΜΕΝΩΝ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΔΡΟΜΟΣ ΔΗΜΗΤΡΙΑΔΗΣ

ΑΡΤΑ, ΙΟΥΛΙΟΣ, 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

ΣΧΟΛΗ: ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ: ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΣΥΓΚΡΙΤΙΚΗ ΜΕΛΕΤΗ APACHE HADOOP ΜΕ APACHE SPARK ΓΙΑ ΕΠΕΞΕΡΓΑΣΙΑ ΜΕΓΑΛΩΝ ΔΕΔΟΜΕΝΩΝ

Πρόδρομος Δημητριάδης

Επιβλέπων:

Χρήστος Γκόγκος,

Αναπληρωτής Καθηγητής

Άρτα, Ιούλιος, 2021

COMPARISON OF APACHE HADOOP WITH APACHE SPARK FOR BIG DATA PROCESSING

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

Άρτα, 2021

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής
   Χρήστος Γκόγκος,
   Αναπληρωτής Καθηγητής

2. Μέλος επιτροπής
   Χρυσόστομος Στύλιος,
   Καθηγητής

3. Μέλος επιτροπής
   Αλέξανδρος Τζάλλας,
   Επίκουρος Καθηγητής

Ο Προϊστάμενος του Τμήματος Ευριπίδης Γλαβάς,

Καθηγητής

Υπογραφή

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα πτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.

Δημητριάδης, Πρόδρομος

Υπογραφή

ΕΥΧΑΡΙΣΤΙΕΣ

Με την ολοκλήρωση των προπτυχιακών σπουδών μου, θα ήθελα πρώτα απ' όλα να ευχαριστήσω την οικογένειά μου για όλη τη βοήθεια και τη στήριξη που μου παρείχαν αυτά τα χρόνια. Έπειτα, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, τον κ. Γκόγκο Χρήστο (Αναπληρωτής Καθηγητής), για τη βοήθεια και την καθοδήγηση που μου πρόσφερε, ώστε να υλοποιήσω την πτυχιακή μου εργασία. Τέλος, θα ήθελα να ευχαριστήσω και τον συμφοιτητή μου Νάστο Βασίλη για τη σημαντική βοήθεια που μου έδωσε κατά τη διάρκειας της πτυχιακής μου εργασίας.

ΠΕΡΙΛΗΨΗ

Κατά τη διάρκεια της πτυχιακή εργασίας αναλύθηκαν τα εργαλεία Apache Hadoop και Apache Spark και δημιουργήθηκαν οι εγκαταστάσεις τους για single-node και multi-node μορφή. Για τη multi-node μορφή δημιουργήθηκε μία συστάδα υπολογιστών, όπου αποτελείται από 5 υπολογιστές, σε ένα από τα εργαστήρια του τμήματος Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Ιωαννίνων στην Άρτα. Με τη χρήση των εργαλείων Apache Hadoop και Apache Spark, καθώς και τις βιβλιοθήκες τους για μηχανική εκμάθηση Apache Mahout και MLlib, αλλά και το Hadoop Streaming, δημιουργήθηκαν τρεις εφαρμογές. Συγκεκριμένα, μία εφαρμογή σύστασης ταινιών με τη χρήση Mahout με δεδομένα χρήστες, ταινίες, αξιολογήσεις και χρονική σήμανση, και ένα παράδειγμα με τη χρήση Hadoop Streaming για ανάλυση βαθμολογίας ταινιών με δεδομένα χρήστες, ταινίες, αξιολογήσεις και χρονική σήμανση. Τέλος, δημιουργήθηκε ακόμα μία εφαρμογή σύστασης ταινιών με δεδομένα ταινίες και κριτικές διάφορων χρηστών με τη χρήση MLlib, Collaborative Filtering και ALS (Alternating Least Squares).

Λέξεις-κλειδιά: συστάδα υπολογιστών, μηχανική εκμάθηση, ανάλυση βαθμολογίας, σύσταση ταινιών

ABSTRACT

During of my thesis, the Apache Hadoop and Apache Spark tools were analyzed and their installations for single-node and multi-node format were created. For the multi-node format, a cluster consisting of 5 computers was created, in one of the laboratories of the Department of Informatics and Telecommunications of the University of Ioannina in Arta. Using the Apache Hadoop and Apache Spark tools, as well as their Apache Mahout and MLlib machine learning libraries, as well as Hadoop Streaming, three applications were designed. Specifically, a movie recommendation system app using Mahout with user data, movies, ratings, and timestamps, and an example using Hadoop Streaming for movie ratings breakdown with user data, movies, ratings, and timestamps. Last, another movie recommendation system app for movies and reviews of various users was designed using MLlib, Collaborative Filtering and ALS (Alternating Least Squares).

Keywords: cluster, machine learning, ratings breakdown, movie recommendation system

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΓΛΩΣΣΑΡΙΟ

- **Framework:** Πλαίσιο όπου μπορεί να αλλάξει τη γενική λειτουργικότητα επιλεκτικά με πρόσθετο κώδικα γραμμένο από τον χρήστη, παρέχοντας έτσι λογισμικό για συγκεκριμένη εφαρμογή.
- **Scale up:** Ένα μηχάνημα υψηλών προδιαγραφών και υψηλής αξιοπιστίας με εκθετική αύξηση κόστους.
- **Virtual Machine:** Εικονική μηχανή όπου επιτρέπεται η πρόσβαση σε οποιοδήποτε λογισμικό.
- **Daemons:** Πρόγραμμα υπολογιστή που λειτουργεί ως διαδικασία παρασκηνίου, αντί να βρίσκεται υπό τον άμεσο έλεγχο ενός διαδραστικού χρήστη.
- **BigTable:** Κατανεμημένο σύστημα αποθήκευσης όπου έχει σχεδιαστεί για την αντιμετώπιση μεγάλων συνόλων δεδομένων.
- **API REST:** Διεπαφή προγραμματισμού εφαρμογών που προσαρμόζεται με τους περιορισμούς της αρχιτεκτονικής REST και επιτρέπει την αλληλεπίδραση με τις υπηρεσίες RESTful.
- **Avro:** Σύστημα σειριοποίησης δεδομένων.
- **Thrift:** Πλαίσιο επεκτάσιμης ανάπτυξης γλωσσικών υπηρεσιών. Συνδυάζει μία στοίβα λογισμικού με μία μηχανή δημιουργίας κώδικα για τη δημιουργία υπηρεσιών που λειτουργούν μεταξύ διάφορων γλωσσών (Python, Java, C++, Ruby κλπ).
- **Data Frames:** Δομή δεδομένων τύπου πίνακα διαθέσιμη σε γλώσσες προγραμματισμού, όπως python.
- **Ping (Packet Internet Groper):** Μέθοδος για τον εντοπισμό της διαθεσιμότητας και της απόδοσης ενός απομακρυσμένου πόρου του δικτύου.
- **DAG (Directed Acyclic Graph):** Κατευθυνόμενο Άκυκλο Γράφημα (γράφημα με βελάκια), όπου μετατρέπει ένα λογικό σχέδιο εκτέλεσης σε ένα σχέδιο φυσικής εκτέλεσης.
- **Script:** Σενάριο όπου επιτρέπει τον έλεγχο μίας ή περισσότερων εφαρμογών.
- **Kerberos:** Πρωτόκολλο ελέγχου ταυτότητας δικτύου υπολογιστών που λειτουργεί βάσει εισιτηρίων για να επιτρέπει στους κόμβους να επικοινωνούν μέσω ενός μη

ασφαλούς δικτύου για να αποδεικνύουν την ταυτότητά του μεταξύ τους με ασφαλή τρόπο.

- **LDAP (Lightweight: Directory Access Control):** Πρωτόκολλο ανοιχτού κώδικα προτύπου για την πρόσβαση σε υπηρεσίες καταλόγου, όπου τρέχει πάνω από το επίπεδο μεταφοράς ενός δικτύου.

ΕΙΣΑΓΩΓΗ

Σκοπός της παρούσας εργασίας είναι η κατανόηση των Big Data και η εξοικείωση με τα λογισμικά Apache Hadoop και Apache Spark. Όσοι θέλουν να ασχοληθούν με τον κλάδο των «Μεγάλων Δεδομένων» η γνώση της χρήσης των παραπάνω εργαλείων θα τους βοηθήσει σε αρκετά μεγάλο βαθμό. Η εργασία αποτελείται από 5 κεφάλαια. Τα 4 πρώτα αφορούν το θεωρητικό κομμάτι, ενώ στο 5ο παρουσιάζονται δύο παραδείγματα και μία εφαρμογή συστάσεων με τη χρήση των παραπάνω εργαλείων.

Στο 1ο κεφάλαιο θα αναφερθούμε στα Big Data ή αλλιώς «Μεγάλα Δεδομένα» και θα περιέχει τον ορισμό τους, τον συμβολισμό και την έννοια των 3 V καθώς και διάφορες τεχνολογίες πάνω στον κλάδο αυτόν. Έπειτα στο 2ο κεφάλαιο περιγράφονται δύο βασικά εργαλεία μοντελοποίησης δεδομένων, το Apache Hadoop και Apache Spark, καθώς και κάποια τεχνικά χαρακτηριστικά τους μαζί με τα υποσυστήματα του οικοσυστήματος του Hadoop, το stack του Spark και τους τρόπους με τους οποίους προγραμματίζονται. Το 3ο κεφάλαιο σχετίζεται με την εγκατάσταση και παραμετροποίηση των εργαλείων αυτών και το περιβάλλον χρήστη και διαχείρισης. Συγκεκριμένα θα δείξουμε τον τρόπο εγκατάστασης των δύο εργαλείων σε single-node και σε multi-node μορφή. Στο 4ο κεφάλαιο γίνονται συγκρίσεις ανάμεσα στο Hadoop με το Spark και τέλος στο 5ο κεφάλαιο θα δούμε μία εφαρμογή συστάσεων ταινιών με τη χρήση της βιβλιοθήκης Apache Mahout για Hadoop, ένα παράδειγμα ανάλυσης βαθμολογίας με τη χρήση Hadoop Streaming για Hadoop και μία εφαρμογή συστάσεων ταινιών με τη χρήση Collaborative Filtering και της βιβλιοθήκης MLlib για το PySpark.

1 BIG DATA

Εικόνα 1 1 Big Data

1.1 ΟΡΙΣΜΟΣ

Μέσω μεγάλων εταιρειών και των μέσων κοινωνικής δικτύωσης, όπως είναι το Facebook, το Twitter και το Instagram και όχι μόνο, καθημερινά παράγονται εκατομμύρια byte δεδομένων. Τα Big Data συνήθως περιλαμβάνουν σύνολα δομημένων, ημιδομημένων και αδόμητων δεδομένων με τεράστια μεγέθη τα οποία είναι πολύ μεγάλα για επεξεργασία, αποθήκευση, διαχείριση και ανάλυση από ένα μόνο υπολογιστικό σύστημα, με αποτέλεσμα τα παραδοσιακά εργαλεία να μην είναι σε θέση να χειριστούν τέτοιες ποσότητες. Γενικά μία συστάδα Μεγάλων Δεδομένων είναι ένα σύστημα πολλών υπολογιστών με αρκετά μεγάλο χώρο στο δίσκο, αλλά και συνήθως με πάρα πολύ RAM. Στα συστήματα αυτά συνήθως υπάρχουν εργαλεία όπως το Apache Hadoop, Hive, Pig, Sqoop, Spark, Impala και Kafka. Το κάθε ένα έχει τον δικό του σκοπό μέσα σε ένα Big Data flow. Στην παρούσα εργασία θα αναλυθούν τα εργαλεία Hadoop και Spark.

1.2 ΤΑ 3 V

Στα Μεγάλα Δεδομένα υπάρχουν τα γνωστά 3 V, τα οποία συμβολίζουν τον Όγκο, την Ταχύτητα και την Ποικιλομορφία.

- **Volume (Όγκος):** Ο Όγκος σχετίζεται περισσότερο με τα Μεγάλα Δεδομένα, διότι ο όγκος τους είναι αρκετά μεγάλος. Στα μέσα κοινωνικής δικτύωσης για παράδειγμα, ο όγκος αναφέρεται στην ποσότητα δεδομένων που δημιουργούνται μέσω ιστότοπων, πυλών και διαδικτυακών εφαρμογών. Το Facebook έχει δισεκατομμύρια χρήστες όπου ο καθένας από αυτούς αποθηκεύει πολλές φωτογραφίες. Το YouTube, το Twitter, το Instagram κι αυτά έχουν επίσης δισεκατομμύρια χρήστες που καθημερινά αναρτούν βίντεο, tweets και εικόνες [1]. Στα Big Data λοιπόν, ο όγκος αφορά την ανάγκη αποθήκευσης και επεξεργασίας Terabytes ακόμα και Petabytes δεδομένων. Καθώς περνάνε τα χρόνια θα υπάρχουν όλο και περισσότερες τεράστιες συλλογές. Ένα άλλο παράδειγμα είναι συστήματα σύστασης ταινιών. Το Netflix από τον Μάιο του 2019 έχει 13.612 τίτλους. Από το 2016 έχει ολοκληρώσει τη μετεγκατάστασή τους στις Amazon Web Services (AWS) με δεκάδες Petabytes δεδομένων να έχουν μεταφερθεί και υπολογίζεται ότι αποθηκεύει περίπου 105 TB δεδομένων μόνο για ένα βίντεο [2].

- **Velocity (Ταχύτητα):** Η ταχύτητα αφορά τον υψηλό ρυθμό με τον οποίο παράγονται νέα δεδομένα. Το Facebook για παράδειγμα καθημερινά έχει να διαχειριστεί μία πληθώρα δημοσιεύσεων, όπου πρέπει να τις επεξεργαστεί, να τις αρχειοθετήσει και να είναι σε θέση να τις ανακτήσει. Ένα άλλο παράδειγμα είναι οι αισθητήρες. Στην καθημερινότητά μας χρησιμοποιούμε αρκετούς αισθητήρες πια, είτε φορώντας μία έξυπνη συσκευή, όπως smartwatch και smartband, είτε έχοντας μέσα στο σπίτι μας διάφορες έξυπνες συσκευές, όπως είναι στα συστήματα φωτισμού, θέρμανσης, κλιματισμού και αρκετά ακόμα άλλα. Όλοι αυτοί οι αισθητήρες των συσκευών παράγουν συνέχεια αρκετά νέα δεδομένα και πρέπει να σε ενημερώνουν ανά τακτά χρονικά διαστήματα [1].

- **Variety (Ποικιλομορφία):** Η ποικιλομορφία αφορά τη συγκέντρωση δεδομένων από διάφορες πηγές και σε διάφορες μορφές. Ορισμένες μορφές δεδομένων είναι

φωτογραφίες, δεδομένα αισθητήρων, tweets, κρυπτογραφημένα πακέτα και πολλά άλλα ακόμη. Καθένα από αυτά είναι διαφορετικά το ένα από το άλλο και μεγάλο μέρος τους δεν είναι δομημένο, με αποτέλεσμα να μην χωράνε εύκολα σε υπολογιστικά φύλλα ή σε ορισμένες βάσεις δεδομένων [1]. Ωστόσο, υπάρχουν και κάποια δεδομένα τα οποία είναι σε δομημένη μορφή, όπως για παράδειγμα το Netflix, το οποίο συλλέγει δεδομένα όπως η ώρα της ημέρας, η διάρκεια της παρακολούθησης, η δημοτικότητα και διάφορες πληροφορίες που σχετίζονται με την αναζήτηση των χρηστών. Όλη αυτή η ποικιλομορφία δεδομένων αποτελεί τον φορέα ποικιλίας των Big Data [2].

Βέβαια, εκτός από τα γνωστά 3 V υπάρχουν ακόμα δύο, το Veracity και το Value. Το Veracity συμβολίζει την φιλαλήθεια και συγκεκριμένα αφορά τη διασφάλιση ποιότητας, αξιοπιστίας και ακρίβειας των δεδομένων. Τα δεδομένα συλλέγονται από πολλές πηγές, οπότε πρέπει ελέγχονται ώστε να μην υπάρχουν στο σύστημα μη έγκυρα δεδομένα που του προσθέτουν «θόρυβο». Το Value συμβολίζει την οικονομική αξία των δεδομένων και στο πόσο χρήσιμα είναι στη λήψη αποφάσεων. Για παράδειγμα, μία εταιρεία συλλέγοντας διάφορα δεδομένα από τους πελάτες της, μπορεί να γνωρίζει τα ενδιαφέροντάς τους ώστε να δημιουργεί τις σωστές προσφορές για τον καθένα, ακόμα και να αποφύγει κάποια απάτη [3].

1.3 ΤΕΧΝΟΛΟΓΙΕΣ

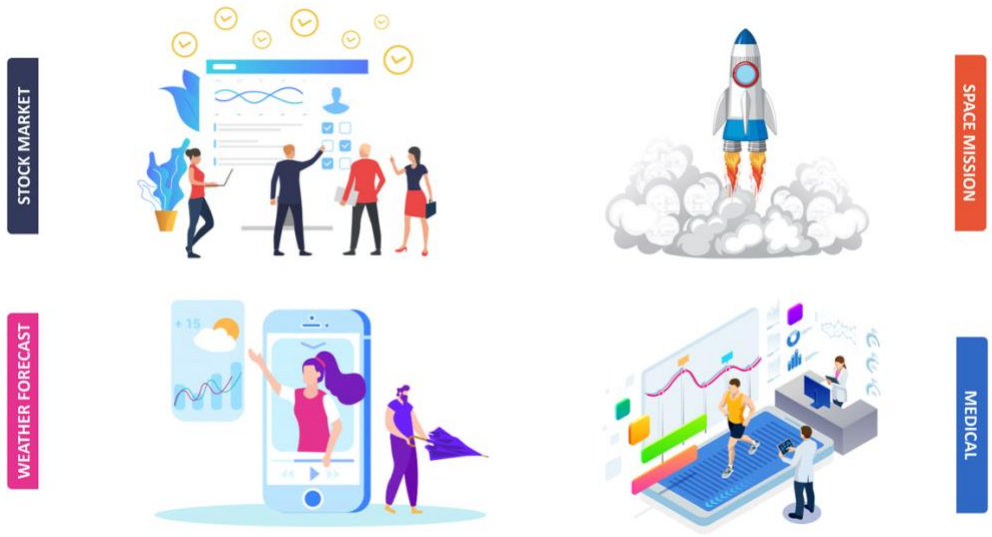
Οι Big Data Τεχνολογίες αφορούν την ανάλυση, την επεξεργασία και την εξαγωγή πληροφοριών από ένα εξαιρετικά περίπλοκο και μεγάλο σύνολο δεδομένων. Είναι απαραίτητες για την ανάλυση τεράστιων δεδομένων σε πραγματικό χρόνο και για προβλέψεις μείωσης κινδύνων. Όλες αυτές οι εργασίες δεν θα μπορούσαν να πραγματοποιηθούν με τα παραδοσιακά εργαλεία επεξεργασίας δεδομένων. Οι Big Data Τεχνολογίες χωρίζονται σε 2 κατηγορίες, στις Τεχνολογίες Μεγάλων Δεδομένων που χρησιμοποιούνται στην πράξη και στην Αναλυτική Μεγάλων Δεδομένων [4]

- **Τεχνολογίες Μεγάλων Δεδομένων που χρησιμοποιούνται στην πράξη:** Οι Τεχνολογίες Μεγάλων Δεδομένων που χρησιμοποιούνται στην πράξη, αφορούν τα

καθημερινά δεδομένα που δημιουργούμε. Στην παρακάτω εικόνα υπάρχουν κάποια συγκεκριμένα παραδείγματα, όπως είναι οι ηλεκτρονικές κρατήσεις εισιτηρίων, οι ηλεκτρονικές αγορές, τα δεδομένα από ιστότοπους μέσων κοινωνικής δικτύωσης και τα στοιχεία του υπαλλήλου οποιασδήποτε πολυεθνικής εταιρείας.

Εικόνα 1.3 1 Operational Big Data Technologies (Πηγή:https://www.edureka.co/blog/top-big-data-technologies/)

- **Αναλυτική Μεγάλων Δεδομένων:** Η Αναλυτική Μεγάλων Δεδομένων αφορά τις κρίσιμες επιχειρηματικές αποφάσεις που λαμβάνονται σε πραγματικό χρόνο με την ανάλυση Μεγάλων Δεδομένων. Στην παρακάτω εικόνα υπάρχουν κάποια παραδείγματα, όπως αυτό του χρηματιστηρίου, της διεξαγωγής διαστημικών αποστολών, τις πληροφορίες πρόγνωσης του καιρού και τα ιατρικά επίπεδα όπου μπορεί να παρακολουθείται μια συγκεκριμένη κατάσταση της υγείας των ασθενών.



Εικόνα 1.3 2 Analytical Big Data Technologies (Πηγή:https://www.edureka.co/blog/top-big-data-technologies/)

Κάποιες από τις Big Data Τεχνολογίες είναι οι εξής:

- Hadoop, MongoDB και RainStor, οι οποίες ανήκουν στο πεδίο Αποθήκευσης Δεδομένων.
- Presto, RapidMiner και Elasticsearch, οι οποίες ανήκουν στο πεδίο Εξόρυξης Δεδομένων.
- Spark, Kafka, Splunk, KNIME, R-Language, Flume και Blockchain, οι οποίες ανήκουν στο πεδίο Ανάλυσης Δεδομένων.
- Tableau και Plotly, οι οποίες ανήκουν στο πεδίο Οπτικοποίησης Δεδομένων.

Εικόνα 1.3 3 Big Data Technologies (Πηγή:https://www.edureka.co/blog/top-big-data-technologies/)

2 ΕΡΓΑΛΕΙΑ ΜΟΝΤΕΛΟΠΟΙΗΣΗΣ ΔΕΔΟΜΕΝΩΝ

2.1 APACHE HADOOP

Εικόνα 2.1 1 Apache Hadoop

Το Apache Hadoop είναι ένα framework ανοιχτού κώδικα το οποίο επιτρέπει στον χρήστη την αποθήκευση και επεξεργασία μεγάλων συνόλων δεδομένων σε συστάδες υπολογιστών χρησιμοποιώντας απλά μοντέλα προγραμματισμού. Είναι έτσι σχεδιασμένο για να μπορεί να κάνει scale up από μεμονωμένους διακομιστές σε χιλιάδες μηχανήματα, προσφέροντας στο καθένα υπολογιστική ισχύ και αποθηκευτικό χώρο τοπικά. Έχει σχεδιαστεί για να εντοπίζει και να χειρίζεται αποτυχίες στο επίπεδο των εφαρμογών, παρέχοντας μια διαθέσιμη υπηρεσία πάνω από μία συστάδα υπολογιστών και όχι να βασίζεται στο hardware για να το πετύχει αυτό. Επιπρόσθετα, αποτελείται από τέσσερις κύριες ενότητες, το Hadoop Common, το οποίο περιέχει εργαλεία ώστε να υποστηρίζονται οι υπόλοιπες λειτουργικές μονάδες, το HDFS (Hadoop Distributed File System), το οποίο είναι ένα κατανεμημένο

Εικόνα 2.1 2 Η έμπνευση του logo του Hadoop

σύστημα αρχείων που παρέχει πρόσβαση υψηλής απόδοσης σε δεδομένα εφαρμογών, το Hadoop YARN, το οποίο είναι ένα πλαίσιο για προγραμματισμό εργασιών και διαχείρισης πόρων σε μια συστάδα υπολογιστών και το Hadoop MapReduce, το οποίο είναι ένα σύστημα για παράλληλη επεξεργασία μεγάλων συνόλων δεδομένων. Τέλος, ιδρυτές του Hadoop είναι ο Doug Cutting και ο Mike Cafarella και συγκεκριμένα ο Doug Cutting ήταν εκείνος που σκέφτηκε το logo και το όνομα του Hadoop, εμπνευσμένος από ένα λούτρινο παιχνίδι του γιου του, έναν κίτρινο ελέφαντα που το φώναζε Hadoop [5].

2.1.1 ΤΕΧΝΙΚΑ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΚΑΙ ΤΡΟΠΟΙ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

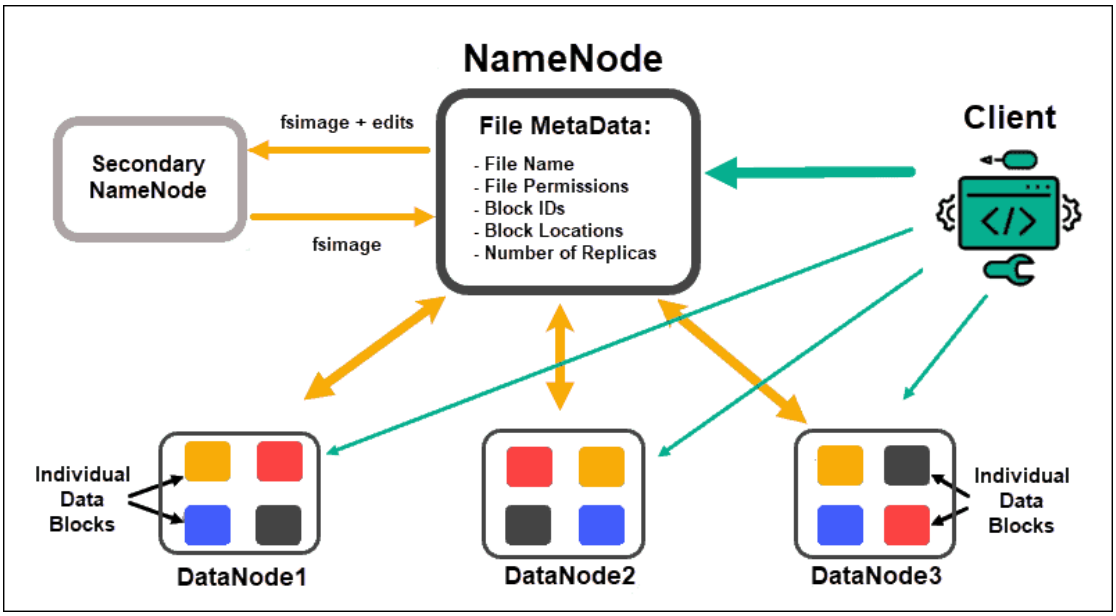
Το Apache Hadoop είναι κυρίως ανεπτυγμένο σε γλώσσα προγραμματισμού Java. Βέβαια υπάρχουν και διάφοροι άλλοι τρόποι προγραμματισμού, όπως είναι το Hadoop Streaming, Cascading, Pig, Scalding και Hive. Τα αρχεία παραμετροποίησης είναι σε μορφή XML. Η εγκατάστασή του επιτυγχάνεται σε λογισμικό Linux, Windows και MacOS, βέβαια ο συνηθέστερος τρόπος είναι σε Linux λογισμικό ή στα υπόλοιπα χρησιμοποιώντας ένα virtual machine. Υπάρχουν και διάφορες εταιρείες που προσφέρουν έτοιμο το περιβάλλον του Hadoop. Κάθε virtual machine περιέχει ένα μεγάλο αριθμό από όλα τα εργαλεία του Hadoop. Δύο τέτοιες εταιρείες είναι η Cloudera σε συνεργασία με τη Hortonworks και η MapR. Βέβαια απαιτούν αρκετά μεγάλη ποσότητα μνήμης, για παράδειγμα το Hortonworks HDP 2.6.5 έχει μέγεθος 15 GB. Μερικά στοιχεία του Apache Hadoop εμπνεύστηκαν από τις εργασίες της Google πάνω στο δικό της MapReduce αλγόριθμο και στο δικό τους σύστημα αρχείων, το Google File System.

2.1.2 ΒΑΣΙΚΑ ΥΠΟΣΥΣΤΗΜΑΤΑ ΟΙΚΟΣΥΣΤΗΜΑΤΟΣ HADOOP

Τα βασικά υποσυστήματα του οικοσυστήματος του Hadoop είναι τρία, το HDFS, το YARN και το MapReduce [6].

- **HDFS (Hadoop Distributed File System):** Το HDFS είναι ένα σύστημα κατανεμημένων αρχείων που έχει σχεδιαστεί για να τρέχει πάνω από το σύστημα αρχείων ενός Linux συστήματος. Έχει πολλές ομοιότητες με τα υπάρχοντα κατανεμημένα συστήματα αρχείων, ωστόσο οι διαφορές από άλλα τέτοια συστήματα

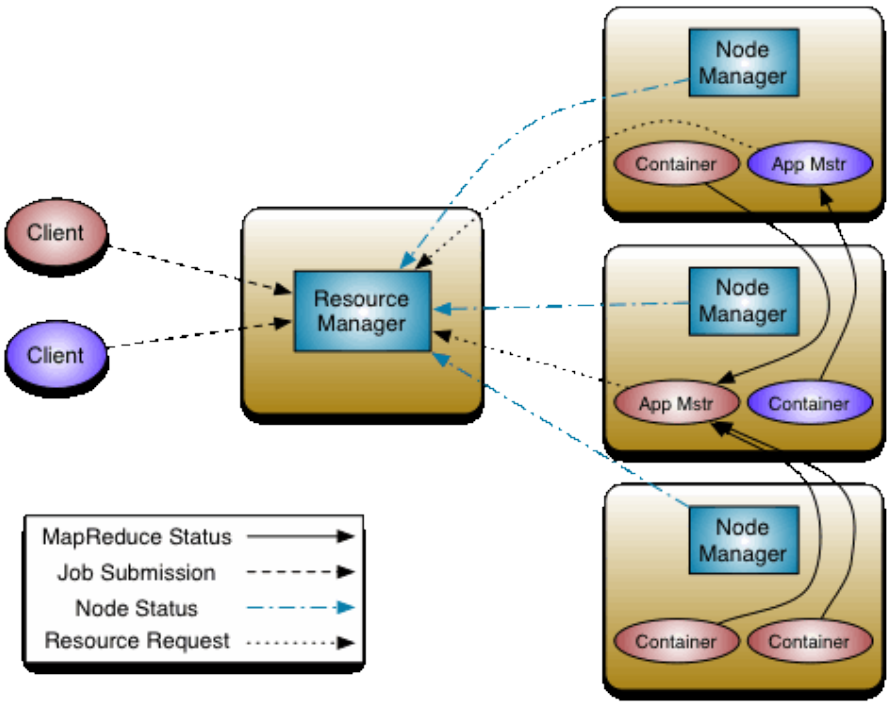
είναι σημαντικές. Το HDFS είναι το βασικότερο υποσύστημα του Hadoop, έχει εξαιρετικά μεγάλη ανοχή σε σφάλματα υλικού και έχει σχεδιαστεί για την ανάπτυξη σε υλικό χαμηλού κόστους. Επίσης, παρέχει πρόσβαση υψηλής απόδοσης σε δεδομένα εφαρμογών και είναι κατάλληλο για εφαρμογές που διαθέτουν μεγάλα σύνολα δεδομένων. Επιπρόσθετα, βοηθά στην αποθήκευση των δεδομένων σε διάφορους κόμβους και στη διατήρηση του αρχείου καταγραφής σχετικά με τα αποθηκευμένα δεδομένα. Η αρχιτεκτονική του είναι master/slave και αποτελείται από τρία στοιχεία, το NameNode, το Secondary NameNode και τα DataNodes. Το NameNode είναι ο κύριος κόμβος (master server) και δεν αποθηκεύει τα πραγματικά δεδομένα. Περιέχει μεταδεδομένα, με αποτέλεσμα να απαιτεί λιγότερη αποθήκευση και υψηλούς υπολογιστικούς πόρους. Το Secondary NameNode παρακολουθεί τις αλλαγές που πραγματοποιούνται στο NameNode και του παρέχει μία γρήγορη εκκίνηση. Όλα τα δεδομένα αποθηκεύονται στα DataNodes, με αποτέλεσμα να απαιτεί περισσότερους πόρους αποθήκευσης. Αυτά τα DataNodes είναι βασικό υλικό (όπως για παράδειγμα οι επιτραπέζιοι υπολογιστές) στο κατανεμημένο περιβάλλον και είναι ο λόγος για τον οποίο οι λύσεις με το Hadoop είναι αρκετά αποδοτικές. Τέλος, το HDFS υποστηρίζει την εργασία με πολύ μεγάλα δεδομένα χωρίζοντάς τα σε μπλοκ. Το μέγεθος των μπλοκ μπορεί να διαμορφωθεί κατά την εγκατάσταση και είναι από προεπιλογή 128 MB [7].



Εικόνα 2.1.2 1 HDFS Architecture (Πηγή:https://phoenixnap.com/kb/apache-hadoop-architecture-explained)

- **YARN (Yet Another Resource Negotiator):** Το YARN είναι ένα σύστημα το οποίο είναι υπεύθυνο για τον διαχωρισμό των λειτουργιών της διαχείρισης πόρων, του προγραμματισμού εργασιών καθώς και της παρακολούθησης εργασιών σε ξεχωριστά daemons. Η κατανομή των πόρων επιτυγχάνεται με το ResourceManager, ενώ το NodeManager είναι υπεύθυνο ανά σύστημα για την παρακολούθηση των πόρων (cpu, memory, disk, network) και για την αναφορά που θα στείλει στο ResourceManager προκειμένου να γίνει σωστά η κατανομή αυτών. Το NodeManager έχει ένα βασικό συστατικό, το ApplicationMaster. Το ApplicationMaster είναι υπεύθυνο για τη διαπραγμάτευση των πόρων από το ResourceManager και τη συνεργασία του με το NodeManager για την εκτέλεση και την παρακολούθηση των εργασιών. Το ResourceManager έχει δύο βασικά συστατικά, τον Scheduler και το ApplicationsManager. Ο Scheduler είναι υπεύθυνος για την κατανομή των πόρων σε διάφορες εφαρμογές που εκτελούνται εκείνη τη στιγμή. Βέβαια δεν προσφέρει καμία εγγύηση για την επανεκκίνηση αποτυχημένων εργασιών λόγω αστοχίας της εφαρμογής ή λόγω σφάλματος του υλικού, καθώς δεν είναι υπεύθυνος για την παρακολούθηση και τον εντοπισμό των εργασιών. Τέλος, εκτελεί τη λειτουργία προγραμματισμού βάσει των απαιτήσεων σε πόρους των εφαρμογών, όπου πραγματοποιείται με βάση τους πόρους που ενσωματώνουν στοιχεία όπως μνήμη, cpu, δίσκος και δίκτυο. Το ApplicationsManager είναι υπεύθυνο για την αποδοχή της εργασίας η οποία του υποβλήθηκε και για τη διαπραγμάτευση του πρώτου πλαισίου πόρων για την εκτέλεση της συγκεκριμένης εφαρμογής ApplicationMaster και την παροχή της υπηρεσίας επανεκκίνησης του ApplicationMaster σε περίπτωση αποτυχίας. Τέλος, είναι υπεύθυνο για τη

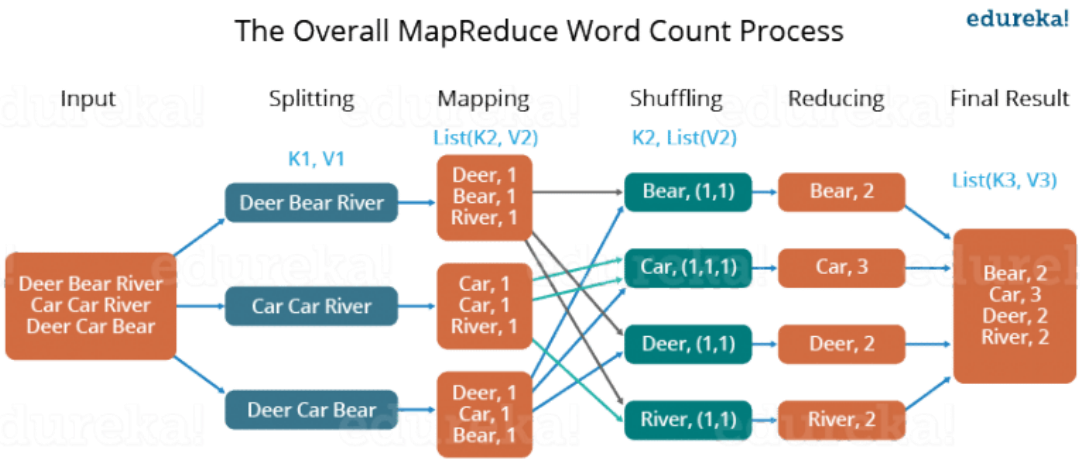
διαπραγμάτευση των κατάλληλων πόρων από τον Scheduler, ώστε να μπορεί να ανιχνεύσει την κατάστασή τους και να παρακολουθεί την πρόοδο της εκτέλεσης [8].



Εικόνα 2.1.2 2 YARN Architecture (Πηγή:https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html)

- **MapReduce:** Το MapReduce είναι ένα framework που βοηθά να γράφονται εφαρμογές οι οποίες επεξεργάζονται τεράστιες ποσότητες δεδομένων παράλληλα σε μεγάλες συστάδες, παρέχοντας αξιοπιστία και ανεκτικότητα σε σφάλματα. Χωρίζει το σύνολο των δεδομένων εισόδου σε ανεξάρτητα κομμάτια, ώστε να δημιουργείται επεξεργασία αντιστοίχισης (Map) και μετά να ταξινομεί τις εξόδους ώστε να επιτευχθεί η εργασία της μείωσης (Reduce). Οι εργασίες εισόδου εξόδου αποθηκεύονται σε ένα σύστημα αρχείων [9]. Συγκεκριμένα μία εργασία MapReduce αποτελείται από τέσσερις φάσεις. Πρώτη φάση είναι αυτή του split, όπου τα δεδομένα χωρίζονται σε πολλούς κόμβους υπολογιστών, δεύτερη του map, όπου εκεί εφαρμόζεται μία λειτουργία αντιστοίχισης και υπάρχει το RecordReader το οποίο επεξεργάζεται κάθε εγγραφή εισόδου και δημιουργεί το αντίστοιχο ζεύγος key-value. Έπειτα, στην τρίτη φάση είναι το sort&shuffle, όπου η έξοδος των αντιστοιχίσεων ταξινομείται και κατανέμεται στους μειωτές και τέλος το reduce, όπου εφαρμόζεται η λειτουργία μείωσης στα δεδομένα και παράγεται μία έξοδος

[10]. Η όλη ιδέα του MapReduce ήταν της Google το 2004. Ένα γνωστό MR πρόβλημα είναι η καταμέτρηση λέξεων (WordCount). Σε ένα txt αρχείο βάζουμε διάφορες λέξεις και τρέχοντας τον αλγόριθμο βλέπουμε τις λέξεις του txt αρχείου και πόσες φορές υπάρχει η κάθε μία. Όπως φαίνεται και στην παρακάτω εικόνα, συγκεκριμένα διαιρούμε την είσοδο σε τρία κομμάτια και στη συνέχεια οριοθετούμε τις λέξεις σε καθέναν από τους mappers, δίνοντας μια τιμή κωδικοποίησης 1 σε κάθε λέξη. Δημιουργείται μία λίστα ζεύγους key-value όπου οι μεμονωμένες λέξεις και η τιμή τους είναι μία. Άρα για την πρώτη γραμμή, Deer Bear River, υπάρχουν τρία ζεύγη key-value ([Deer, 1] [Bear, 1] [River, 1]). Η διαδικασία του mapper παραμένει ίδια σε όλους τος κόμβους. Έπειτα περνάμε στη διαδικασία του ανακατέματος (shuffle) όπου γίνεται η ταξινόμηση και η αναδιάταξη ώστε όλες οι πλειάδες με το ίδιο κλειδί να αποστέλλονται στον αντίστοιχο reducer. Οπότε, κάθε reducer έχει ένα σύνολο κλειδιών και μία λίστα τιμών που αντιστοιχούν σε κάθε κλειδί, [Bear, (1,1)] [Car, (1,1,1)] [Dear, (1,1)] [River, (1,1)]. Κάθε reducer μετρά τις τιμές που υπάρχουν σε αυτή τη λίστα τιμών και λαμβάνει μια λίστα μορφής (1,1) για τη λέξη Bear. Μετρώντας τον αριθμό αυτών στην ίδια λίστα δίνει την τελική έξοδο ως Bear, 2. Τέλος, όλα τα ζεύγη key-value συλλέγονται και γράφονται στο αρχείο εξόδου [11].



Εικόνα 2.1.2 3 MapReduce WordCount Process (Πηγή:https://www.edureka.co/blog/mapreduce-tutorial/)

2.1.3 ΥΠΟΛΟΙΠΑ ΥΠΟΣΥΣΤΗΜΑΤΑ ΟΙΚΟΣΥΣΤΗΜΑΤΟΣ HADOOP

Το οικοσύστημα του Hadoop, εκτός από τα βασικά τρία υποσυστήματα HDFS, YARN και MapReduce, διαθέτει και κάποια άλλα. Κάποια από αυτά είναι το Hive, το Pig, το HBase, το Zookeeper, το Ambari, το Lucene/Solr, το Oozie, το Flume και το Sqoop [6] [12].

- **Hive:** Το Hive διευκολύνει στην αναζήτηση και τη διαχείριση μεγάλων συνόλων δεδομένων, όπου βρίσκονται σε κατανεμημένο χώρο αποθήκευσης και παρέχει έναν μηχανισμό για την αναζήτηση δεδομένων, χρησιμοποιώντας μία γλώσσα προγραμματισμού τύπου SQL, την HQL (Hive Query Language). Επιπρόσθετα, έχει δύο βασικά στοιχεία, το Hive Command Line και τους JDBC (Java Database Connectivity)/ODBC (Object Database Connectivity) drivers. Συγκεκριμένα, το Hive Command Line χρησιμοποιείται για την εκτέλεση εντολών HQL, ενώ το JDBC και το ODBC χρησιμοποιούνται για τη δημιουργία σύνδεσης από την αποθήκευση δεδομένων. Τέλος, λόγω της επεκτασιμότητας όπου έχει, εκτός ότι μπορεί να επεξεργάζεται μεγάλα σύνολα δεδομένων, μπορεί να τα επεξεργάζεται και σε πραγματικό χρόνο.

- **Pig:** Το Pig χωρίζεται σε δύο μέρη, τη γλώσσα Pig Latin (γλώσσα τύπου SQL) και το Pig Runtime για το περιβάλλον εκτέλεσης. Συγκεκριμένα, ο μεταγλωττιστής μετατρέπει τη Pig Latin σε MapReduce, παράγοντας ένα διαδοχικό σύνολο MapReduce εργασιών. Επίσης, το Pig παρέχει μία πλατφόρμα για τη δημιουργία ροής δεδομένων για ETL (Extract-Transform-Load) διαδικασίες, έτσι ώστε να επεξεργάζεται και να αναλύει μεγάλα σύνολα δεδομένων.

- **HBase:** Το HBase είναι ένα έργο ανοιχτού κώδικα, όπου δεν σχετίζεται με κάποια κατανεμημένη βάση δεδομένων. Στην ουσία είναι μία NoSQL βάση δεδομένων. Υποστηρίζει όλους του τύπους δεδομένων και χειρίζεται οτιδήποτε υπάρχει μέσα στο οικοσύστημα του Hadoop. Σχεδιάστηκε για να λειτουργεί πάνω από το HDFS και παρέχει δυνατότητες όπως το BigTable. Το HBase είναι γραμμένο σε γλώσσα προγραμματισμού Java και οι εφαρμογές του μπορούν να γραφτούν σε API REST, Avro και Thrift.

- **Zookeeper:** Το Zookeeper είναι ο συντονιστής οποιασδήποτε Hadoop εργασίας που περιλαμβάνει συνδυασμό διάφορων υπηρεσιών μέσα στο οικοσύστημα. Εξοικονομεί αρκετό χρόνο εκτελώντας συγχρονισμό δεδομένων, συντήρηση διαμόρφωσης των υπηρεσιών και ομαδοποίηση. Παρόλο που είναι μία απλή υπηρεσία, μπορεί και χρησιμοποιείται για τη δημιουργία ισχυρών λύσεων.

- **Ambari:** Το Ambari είναι ένα έργο όπου σαν στόχο έχει να κάνει πιο εύχρηστο το οικοσύστημα του Hadoop. Παρέχει τροφοδοσία στο Hadoop cluster, δίνοντας βήμα προς βήμα τη διαδικασία εγκατάστασης των υπηρεσιών του Hadoop σε μία συστάδα υπολογιστών. Διαχειρίζεται επίσης το cluster, παρέχοντας μία κεντρική υπηρεσία διαχείρισης για εκκίνηση, διακοπή και επαναδιαμόρφωση των υπηρεσιών. Τέλος, παρέχοντας έναν πίνακα εργαλείων παρακολουθεί την υγεία και την κατάσταση του cluster και με το πλαίσιο Amber Alert ειδοποιεί τον χρήστη όποτε απαιτείται προσοχή.

- **Lucene/Solr:** Το Lucene και το Solr είναι δύο υπηρεσίες για την αναζήτηση και την ευρετηρίαση στο οικοσύστημα του Hadoop. Συγκεκριμένα, το Lucene βασίζεται στη γλώσσα προγραμματισμού Java, όπου βοηθά και στον ορθογραφικό έλεγχο. Το Solr από την άλλη είναι μία πλήρης εφαρμογή που δημιουργήθηκε γύρω από το Lucene.

- **Oozie:** Το Oozie είναι ένα από τα διαθέσιμα εργαλεία του Hadoop για τον προγραμματισμό ροών εργασιών. Υπάρχουν δύο είδη εργασιών Oozie, η ροή εργασιών και ο συντονιστής. Η ροή εργασιών Oozie, πρόκειται για ένα διαδοχικό σύνολο ενεργειών που πρέπει να εκτελεστούν, ενώ ο συντονιστής Oozie, πρόκειται για τις εργασίες που ενεργοποιούνται όταν τα δεδομένα διατίθενται σε αυτό.

- **Flume:** Το Flume είναι μία υπηρεσία που βοηθά στην αξιοποίηση μη δομημένων και ημιδομημένων δεδομένων στο HDFS. Προσφέρει μία αξιόπιστη και διανεμημένη λύση ως προς τη συλλογή, τη συγκέντρωση, αλλά και τη μετακίνηση μεγάλου όγκου συνόλων δεδομένων. Επιπρόσθετα, βοηθά και στην αξιοποίηση δεδομένων συνεχούς ροής από διάφορες πηγές, όπως κυκλοφορία δικτύου, μέσα κοινωνικής δικτύωσης, email, αρχεία καταγραφής και διάφορα άλλα στο HDFS.

- **Sqoop:** Το Sqoop, όπως και το Flume, είναι μία υπηρεσία που βοηθά στην αξιοποίηση δεδομένων. Κύρια διαφορά τους είναι ότι το Sqoop μπορεί να εισάγει και να εξάγει δομημένα δεδομένα από RDBMS ή επιχειρησιακές αποθήκες δεδομένων στο HDFS και αντίστροφα.

2.2 APACHE SPARK

Εικόνα 2.2 1 Apache Spark

Το Apache Spark είναι μία μηχανή επεξεργασίας και ανάλυσης δεδομένων ανοιχτού κώδικα και χρησιμοποιείται για την αποθήκευση, την επεξεργασία και ανάλυση δεδομένων σε πραγματικό χρόνο σε διάφορες συστάδες υπολογιστών χρησιμοποιώντας απλές κατασκευές προγραμματισμού. Επεκτείνει το MapReduce και μπορεί να προγραμματιστεί σε διάφορες γλώσσες προγραμματισμού (θα αναφερθούν παρακάτω). Επίσης, περιλαμβάνει βιβλιοθήκες για μηχανική εκμάθηση, επεξεργασία ροής δεδομένων και επεξεργασία γράφων. Ξεκίνησε το 2009 ως ερευνητικό έργο στο Πανεπιστήμιο της Καλιφόρνια στο AMPLab του UC Berkley από την Matei Zaharia και το 2010 πήρε άδεια από το BSD. Το 2013 δωρίστηκε στο Apache Software Foundation και άλλαξε την άδειά του σε Apache 2.0 και τον Φεβρουάριο του 2014 έγινε πρωτεύον έργο της Apache. Τέλος, τον Νοέμβριο του 2014 η εταιρεία Databricks δημιούργησε ένα ρεκόρ στην ταξινόμηση μεγάλου όγκου χρησιμοποιώντας το Spark [13].

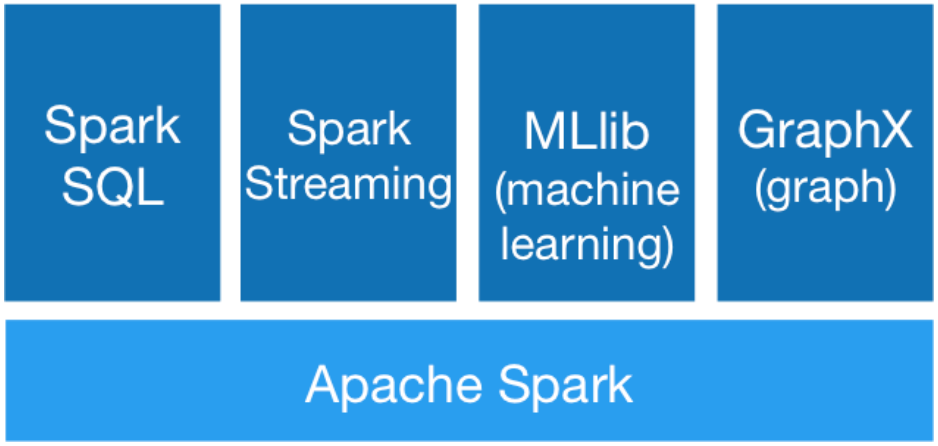
2.2.1 ΤΕΧΝΙΚΑ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΚΑΙ ΤΡΟΠΟΙ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Το Spark μπορεί να προγραμματιστεί σε τέσσερις γλώσσες προγραμματισμού, Java, Python, Scala και R και ορισμένα αρχεία παραμετροποίησής του είναι σε μορφή XML. Η εγκατάστασή του επιτυγχάνεται σε λογισμικό Linux, Windows και MacOS. Μπορεί να τρέξει σε αρκετά συστήματα, όπως Hadoop, Apache Mesos, Kubernetes, standalone ή σε cloud και να έχει πρόσβαση σε διάφορες πηγές δεδομένων. Εκτός από τις βιβλιοθήκες που

έχει, αποτελείται και από το Spark Core, το οποίο είναι υπεύθυνο για την παροχή κατανεμημένης μετάδοσης εργασιών, προγραμματισμού και I/O λειτουργιών. Επίσης, το Spark Core χρησιμοποιεί ένα κατανεμημένο σύνολο δεδομένων τα RDDs (Resilient Distributed Dataset), τα οποία μοιάζουν με τους πίνακες των βάσεων δεδομένων. Επιπρόσθετα, τα RDDs υποστηρίζουν κατανεμημένη αποθήκευση δεδομένων στις μνήμες των μηχανημάτων μιας συστάδας υπολογιστών, ώστε να επιτυγχάνεται ανοχή σε σφάλματα υλικού καταγράφοντας το ιστορικό των μετασχηματισμών που εφαρμόζονται στα δεδομένα και να επιτυγχάνεται και υψηλή απόδοση με παραλληλισμό επεξεργασίας στους κόμβους των συστάδων [13].

Εικόνα 2.2.1 1 Συστήματα όπου τρέχει το Spark (Πηγή:https://spark.apache.org/)

2.2.2 ΤΟ STACK ΤΟΥ SPARK



Εικόνα 2.2.2 1 Το Stack του Spark (Πηγή:https://spark.apache.org/)

Το stack του Spark αποτελείται από τέσσερις βιβλιοθήκες, την Spark SQL, Spark Streaming, MLlib για Machine Learning και GraphX για γράφους. Μάλιστα αυτές οι βιβλιοθήκες μπορούν να συνδυαστούν και σε μία ίδια εφαρμογή [14].

- **Spark SQL:** Το Spark SQL χρησιμοποιείται για επεξεργασία δομημένων δεδομένων και παρέχει ένα API, το DataFrames που μπορεί να λειτουργήσει σαν κατανεμημένη μηχανή SQL ερωτήσεων, με αποτέλεσμα να είναι προσβάσιμο σε περισσότερους χρήστες. Παρέχεται επίσης και ένα module, το spark session, το οποίο χρησιμεύει στη σύνδεση και στην παραμετροποίηση του cluster δικτύου μας. Το Spark SQL τρέχει πάνω από το Spark Core, επιτρέποντας στους χρήστες να εισάγουν σχεσιακά δεδομένα από πίνακες Hive και αρχεία Parquet. Μπορεί ακόμα να συνδυαστεί και με το υπόλοιπο stack, όπως για παράδειγμα την ενσωμάτωση επεξεργασίας SQL ερωτημάτων με μηχανική εκμάθηση (Machine Learning). Επιπρόσθετα, παρέχει και το επεκτάσιμο εργαλείο βελτιστοποίησης, το Catalyst, το οποίο βοηθά στην υποστήριξη ενός ευρέος φάσματος πηγών δεδομένων και αλγορίθμων στα Big Data [15].

- **Spark Streaming:** Το Spark Streaming προστέθηκε στο Apache Spark το 2013. Είναι μία επέκταση του βασικού API του Spark, όπου χρησιμοποιείται για streaming analytics. Συγκεκριμένα χρησιμοποιείται για τη ροή νέων δεδομένων σε πραγματικό χρόνο και επιτρέπει ισχυρές διαδραστικές αναλυτικές εφαρμογές, τόσο σε ροή όσο

και σε ιστορικά δεδομένα, έχοντας και τα χαρακτηριστικά ευκολίας στη χρήση και στην ανοχή σφαλμάτων υλικού του Spark. Μπορεί και ενσωματώνεται με δημοφιλείς πηγές δεδομένων, όπως είναι το HDFS, το Flume, το Kafka και το Twitter. Πραγματοποιεί επεξεργασία δεδομένων με τη χρήση σύνθετων αλγορίθμων που εκφράζονται με λειτουργίες υψηλού επιπέδου, όπως map και reduce και έπειτα αυτά τα δεδομένα μπορούν να προωθηθούν σε συστήματα αρχείων, βάσεις δεδομένων και dashboards. Γενικά, αντί να επεξεργάζεται τα δεδομένα ροής ως μία εγγραφή κάθε φορά, τα διαχωρίζει σε μικρές παρτίδες, με αποτέλεσμα να δέχεται δεδομένα παράλληλα. Αποθηκεύει τα δεδομένα στη μνήμη των workers και στη συνέχεια εκτελούνται σύντομες εργασίες για την επεξεργασία αυτών των παρτίδων και την παραγωγή των αποτελεσμάτων στα συστήματα που αναφέρθηκαν προηγουμένως (συστήματα αρχείων, βάσεις δεδομένων, dashboards). Αυτό που καταφέρνει το Spark, είναι να αναθέτει τις εργασίες στους workers, βάσει της τοποθεσίας των δεδομένων και των διαθέσιμων πόρων, με αποτέλεσμα την καλύτερη εξισορρόπηση φορτίου και την ταχύτερη αποκατάσταση σφαλμάτων υλικού. Τέλος, κάθε παρτίδα υλικού είναι ένα RDD, ένα κατανεμημένο σύνολο δεδομένων δηλαδή, όπου επιτρέπει την επεξεργασία των δεδομένων ροής χρησιμοποιώντας οποιονδήποτε κώδικα ή βιβλιοθήκη του Spark [16].

- **MLlib (Machine Learning):** Το MLlib είναι μία επεκτάσιμη βιβλιοθήκη μηχανικής εκμάθησης (Machine Learning), η οποία παρέχει αλγόριθμους υψηλής ποιότητας και ταχύτητα εκκίνησης. Χρησιμοποιείται για έλεγχο υποθέσεων, συσταδοποίηση, ανάλυση κύριων συνιστωσών κλπ. Επιπρόσθετα, μπορεί να χρησιμοποιηθεί σε Java, Scala και Python ως μέρος των εφαρμογών του Spark. Παρέχει εργαλεία όπως [17],

    - ML Αλγόριθμοι, όπου είναι κοινοί αλγόριθμοι μάθησης (ταξινόμηση, παλινδρόμηση, ομαδοποίηση).
    - Pipelines, όπου είναι εργαλεία για κατασκευή, αξιολόγηση και συντονισμό των ML Pipelines.
    - Βοηθητικά προγράμματα, όπως γραμμική άλγεβρα, στατιστικά στοιχεία και διαχείριση δεδομένων.

Το MLlib επίσης, υποστηρίζει και αρκετούς τύπους δεδομένων, όπως για παράδειγμα τοπικά διανύσματα και πίνακες που αποθηκεύονται σε ένα μόνο μηχάνημα και κατανεμημένους πίνακες όπου υποστηρίζονται από ένα ή περισσότερα RDDs. Τα τοπικά διανύσματα και οι τοπικοί πίνακες είναι απλά μοντέλα δεδομένων, παρόλα αυτά λειτουργούν ως δημόσιες διεπαφές [18]. Τέλος, σύμφωνα με τη Databricks το MLlib είναι έως και 100 φορές πιο γρήγορο από το MapReduce [19].

- **GraphX (graph):** Το GraphX χρησιμοποιείται για την ανάλυση γράφων και έχει δημιουργηθεί πάνω από το Spark επιτρέποντας στους χρήστες να αλληλοεπιδρούν διαδραστικά. Επεκτείνει τα RDD με ένα γράφημα ανθεκτικών κατανεμημένων ιδιοτήτων. Το συγκεκριμένο γράφημα, είναι ένα κατευθυνόμενο πολύγραμμα, όπου έχει πολλές άκρες παράλληλα. Ένα χαρακτηριστικό τους είναι πως μοιράζονται την ίδια πηγή και την κορυφή προορισμού. Επιπρόσθετα, μπορούν να δημιουργήσουν και πολλαπλές σχέσεις μεταξύ των ίδιων κορυφών. Όπως τα RDD, έτσι και τα γραφήματα ιδιοτήτων είναι επίσης αμετάβλητα, κατανεμημένα και ανεκτικά σε σφάλματα υλικού. Ο χρήστης έχει τη δυνατότητα να δημιουργήσει ένα νέο γράφημα, αλλάζοντας τις τιμές ή η δομή του. Ακόμα, μπορεί να επαναχρησιμοποιήσει τα μέρη του αρχικού γραφήματος στο νέο γράφημα, με αποτέλεσμα τη μείωση του κόστους αυτής της λειτουργικής δομής δεδομένων [20]. Τέλος, το GraphX συνοδεύεται από μία βιβλιοθήκη κοινών αλγορίθμων, όπως για παράδειγμα οι PageRank και Triangle Count. Το PageRank μετρά τη σημασία κάθε κορυφής σε ένα γράφημα, υποθέτοντας ότι ένα άκρο από το x στο y αντιπροσωπεύει μία έγκριση της σημασίας του y στο x. Για παράδειγμα, αν ένας χρήστης στο Instagram ακολουθείται από πολλούς άλλους χρήστες, τότε θα κατατάσσεται σε υψηλή θέση. Το Triangle Count είναι ένας αλγόριθμος μέτρησης τριγώνων που καθορίζει τον αριθμό των τριγώνων που διέρχονται από κάθε κορυφή, παρέχοντας ένα μέτρο ομαδοποίησης. Ο υπολογισμός των αριθμών των τριγώνων του συνόλου δεδομένων πραγματοποιείται από το PageRank. Επιπρόσθετα, το Triangle Count απαιτεί τα άκρα να έχουν κανονικό προσανατολισμό και το γράφημα να χωρίζεται χρησιμοποιώντας το Graph.partionBy. Υπάρχουν ακόμα και κάποιοι άλλοι αλγόριθμοι, όπως για παράδειγμα οι Label propagation και SVD++ [21].

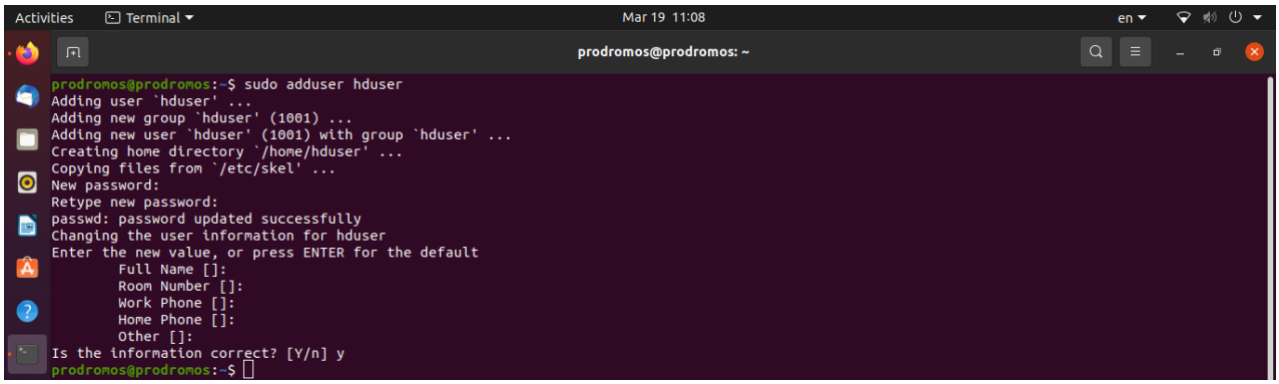
3 ΕΓΚΑΤΑΣΤΑΣΗ ΛΟΓΙΣΜΙΚΟΥ

Η εγκατάσταση του λογισμικού σε single node μορφή παρουσιάστηκε σε ένα macOS σύστημα με επεξεργαστή 1,6 GHz Dual-Core Intel Core i5, με 4 Threads και μία μνήμη RAM 8 GB 2133 MHz LPDDR3. Για την εγκατάσταση και την εκτέλεση των Apache Hadoop και Apache Spark χρησιμοποιήθηκε ένα virtual machine (μέσω του προγράμματος virtual box) με λειτουργικό Linux Ubuntu 20.04.2 LTS, τύπο 64-bit και μνήμη 4 GB. Οι εγκαταστάσεις σε multi node μορφή πραγματοποιήθηκαν από τον συνάδερφο Βασίλη Νάστο σε 5 υπολογιστές του εργαστηρίου του τμήματος, με λειτουργικό Linux Ubuntu 20.04.2 LTS, επεξεργαστή Intel Core i5-6500 CPU @ 3.20 GHz, μνήμη RAM 8 GB και δύο δίσκους, έναν με 256 GB SSD και έναν με 500 GB HDD .

3.1 ΕΓΚΑΤΑΣΤΑΣΗ APACHE HADOOP

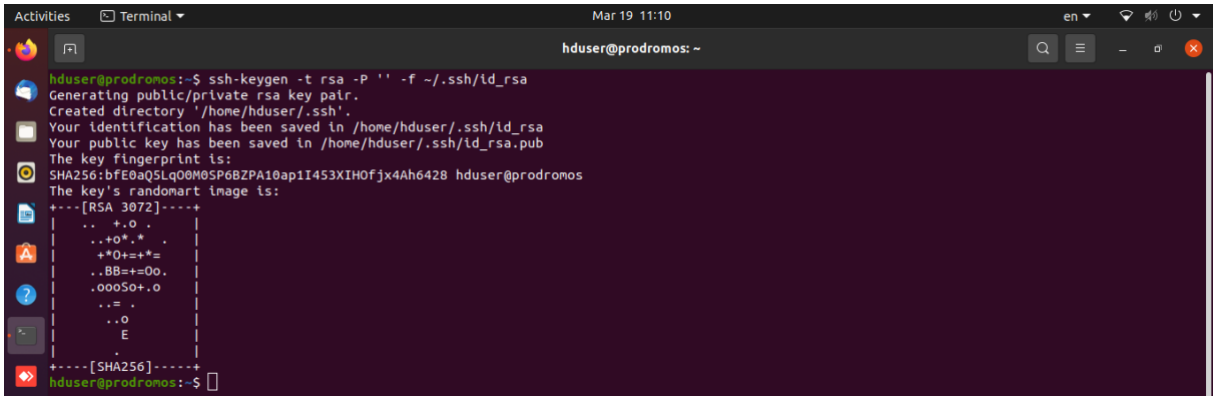
3.1.1 ΕΓΚΑΤΑΣΤΑΣΗ ΚΑΙ ΠΑΡΑΜΕΤΡΟΠΟΙΗΣΗ ΣΕ SINGLE NODE

Πρώτα από όλα χρειάζεται να γίνει μία ενημέρωση του συστήματος από το terminal χρησιμοποιώντας την εντολή **“sudo apt update”**. Στη συνέχεια πρέπει να γίνει η εγκατάσταση μας JAVA OPENJDK 8 χρησιμοποιώντας την εντολή **“sudo apt install openjdk-8-jdk -y”** και με την εντολή **“java -version; Javac -version”** ελέγχουμε ότι έχει κατέβει η έκδοση που θέλουμε. Έπειτα, χρειάζεται να δημιουργήσουμε έναν Non-Root χρήστη για το περιβάλλον του Hadoop. Εγκαθιστούμε οπότε έναν OpenSSH server και έναν πελάτη με την εντολή **“sudo apt install openssh-server openssh-client -y”** και φτιάχνουμε έναν Hadoop χρήστη (στη δικιά μας περίπτωση τον ονομάσαμε hduser**)** με την εντολή **“sudo adduser hduser”**.



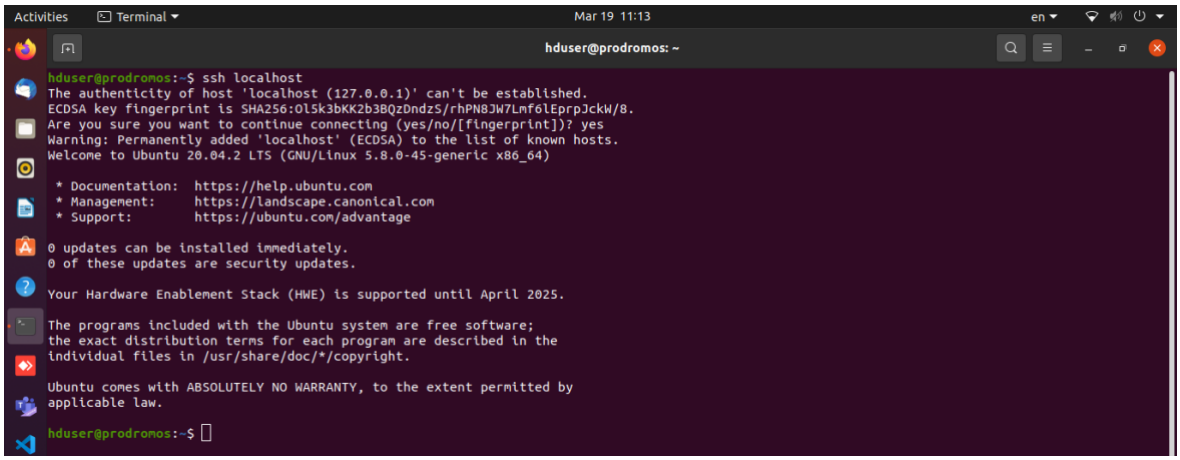
Εικόνα 3.1.1 1 Δημιουργία χρήστη Hadoop

Έχοντας δημιουργήσει τον νέο χρήστη δίνοντας την εντολή **“su – hduser”** μπαίνουμε μέσα σε αυτόν. Τώρα πρέπει να ενεργοποιήσουμε ένα ssh χωρίς κωδικό πρόσβασης για τον νέο χρήστη. Όπως φαίνεται παρακάτω στην Εικόνα 3.1.1 2 δημιουργούμε ένα ζεύγος κλειδιών ssh και ορίζουμε την τοποθεσία που πρέπει να αποθηκευτεί δίνοντας την εντολή **“ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa”**.



Εικόνα 3.1.1 2 Δημιουργία ζεύγος κλειδιών ssh και ορισμός της τοποθεσίας αποθήκευσης

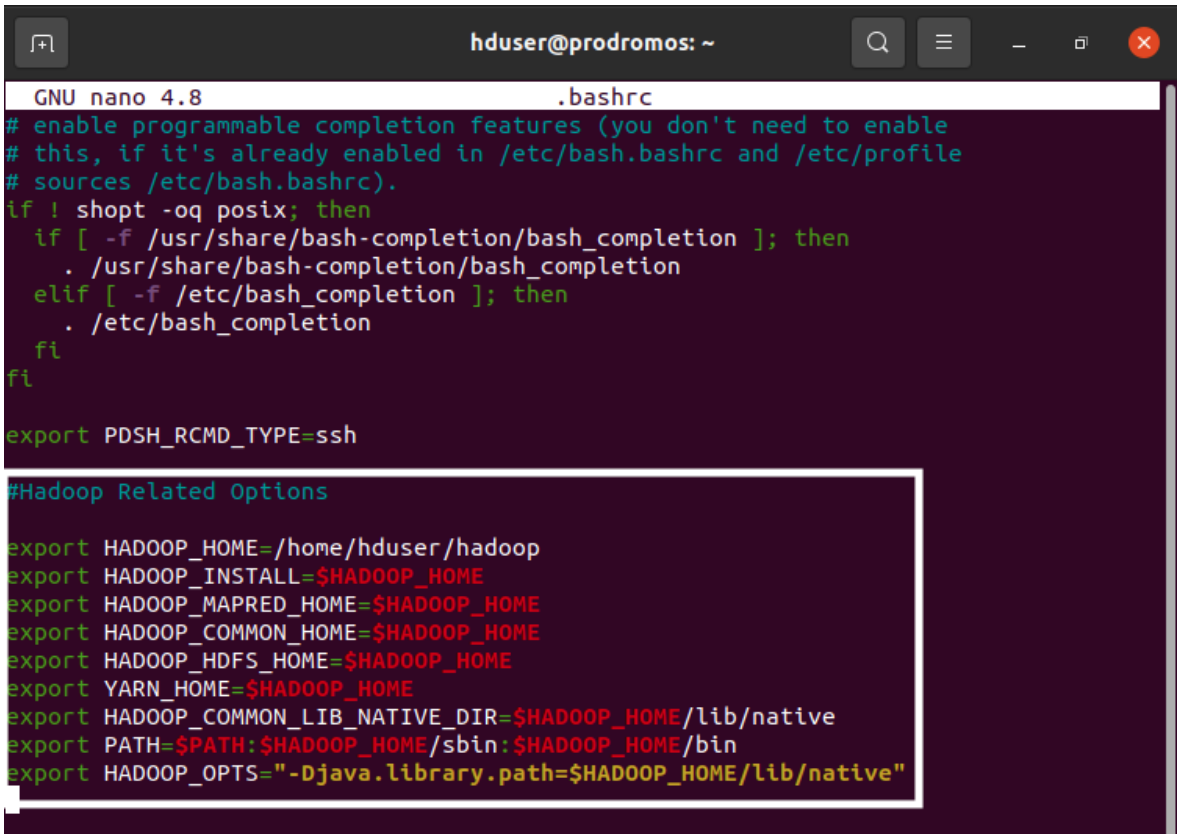
Έπειτα, με την εντολή **“cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys”** πραγματοποιούμε την εξουσιοδότηση του νέου κλειδιού στον κατάλογο ssh και με την εντολή **“chmod 0600 ~/.ssh/authorized_keys”** ορίζουμε τα δικαιώματα για τον χρήστη. Πλέον ο νέος χρήστης έχει τη δυνατότητα ssh χωρίς να χρειάζεται να εισάγει κωδικό πρόσβασης κάθε φορά. Για την επαλήθευση ότι όλα έχουν δημιουργηθεί σωστά δίνουμε την εντολή **“ssh localhost”** και αν δεν υπάρχει κάποιο λάθος θα εμφανιστεί το παρακάτω μήνυμα.



Εικόνα 3.1.1 3 ssh localhost

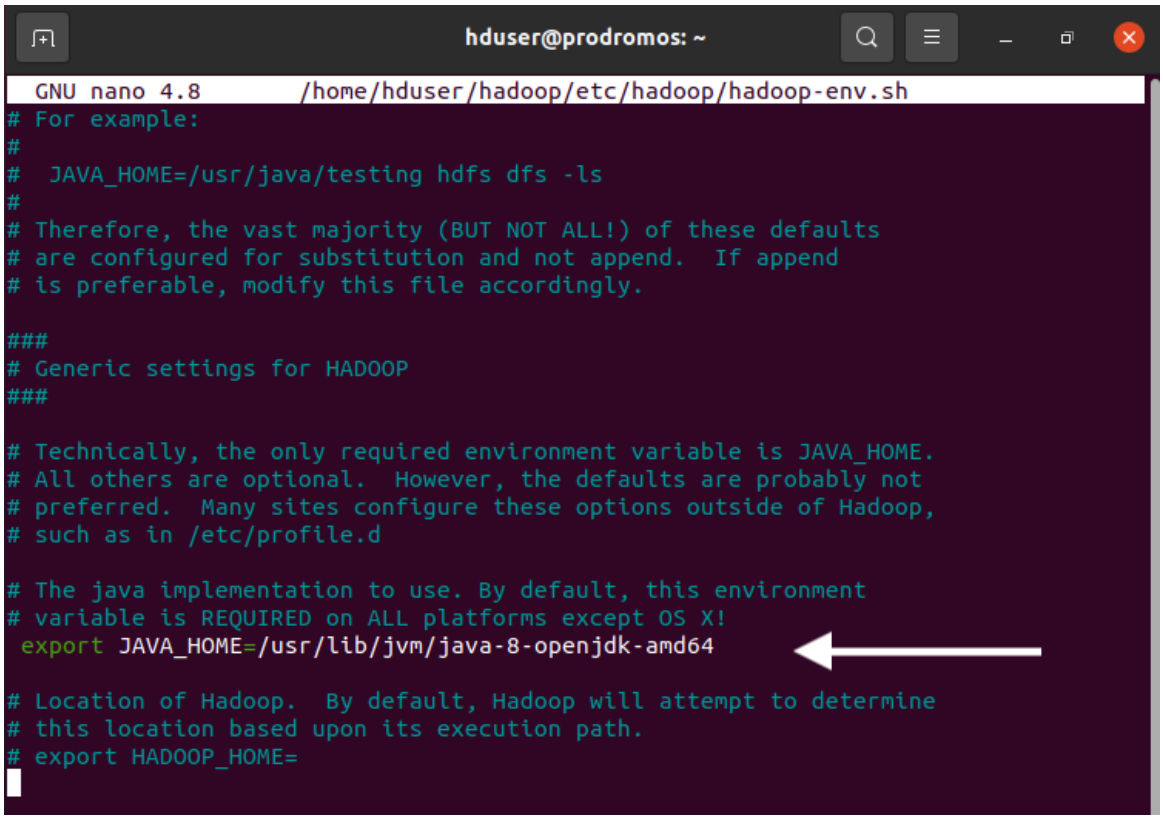
Μετά την παραπάνω προεργασία είμαστε έτοιμοι να εγκαταστήσουμε το Apache Hadoop. Αυτό μπορεί να γίνει με δύο τρόπους, είτε πηγαίνοντας στη σελίδα **https://hadoop.apache.org/** και από την ενότητα Download διαλέξουμε την binary έκδοση που προτιμάμε (στην προκειμένη περίπτωση το Hadoop-3.2.1) ή δίνοντας την εντολή **"wget https://downloads.apache.org/hadoop/common/hadoop-3.2.1/hadoop-3.2.1.tar.gz".**

Έπειτα με την εντολή **"tar xzf hadoop-3.2.1.tar.gz"** αποσυμπιέζουμε το Hadoop αρχείο. Ώστε να είναι πιο εύκολη η πλοήγηση αργότερα στους φακέλους του Hadoop δίνουμε την εντολή **"mv ./hadoop-3.2.1 ./hadoop"** και έτσι μετονομάζουμε το αρχείο σε hadoop. Για να μην υπάρχει αργότερα πρόβλημα όταν θα πάμε να εκκινήσουμε τους κόμβους, πηγαίνουμε στο κέλυφος bashrc με την εντολή **"sudo nano .bashrc"** ώστε να τροποποιήσουμε το προεπιλεγμένο rcmd pdsh σε ssh και τοποθετούμε την εντολή **"export PDSH_RCMD_TYPE=ssh".** Βγαίνοντας από το bashrc πρέπει να δώσουμε την εντολή **"source ~/.bashrc"** ώστε να εφαρμοστούν οι αλλαγές που πραγματοποιήσαμε. Έπειτα θα φτιάξουμε το Hadoop σε Single-Node ή αλλιώς Pseudo-Distributed Mode. Μεταβαίνουμε ξανά στο bashrc με την εντολή **"sudo nano .bashrc"** και κάνουμε export τις εντολές που φαίνονται στην Εικόνα 3.1.1 4 (Δεν ξεχνάμε ότι βγαίνοντας από το bashrc δίνουμε την εντολή **"source ~/.bashrc"**).



Εικόνα 3.1.1 4 Hadoop Related Options

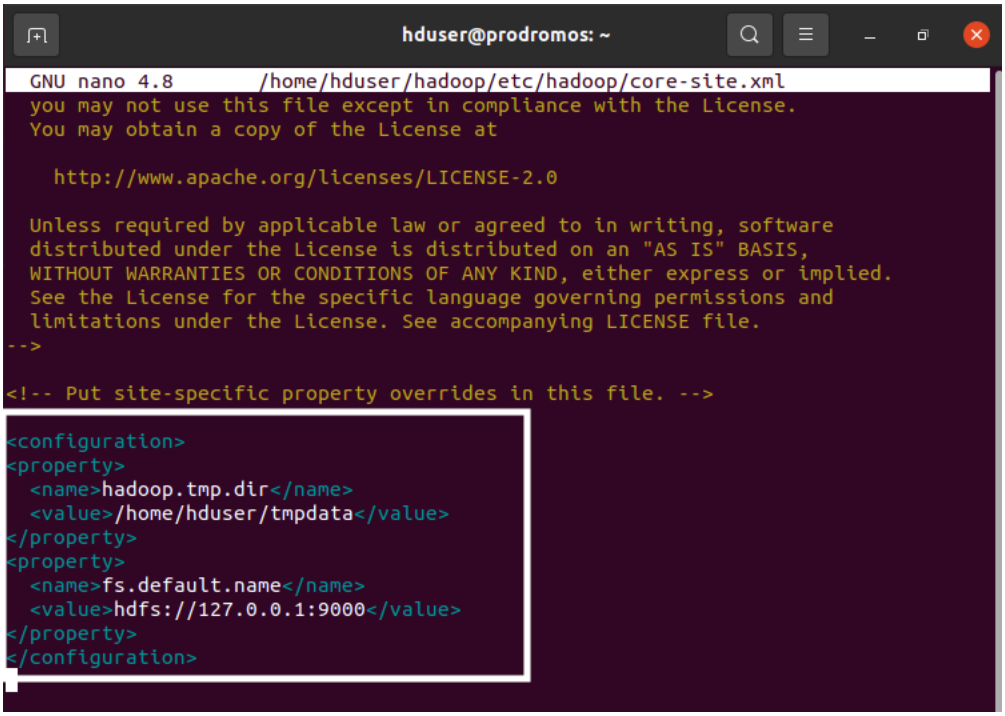
Μετά πρέπει να γίνει η επεξεργασία του αρχείου hadoop-env.sh, διότι αυτό το αρχείο χρησιμεύει ως κύριο αρχείο για τη διαμόρφωση των ρυθμίσεων που σχετίζονται με το YARN, το HDFS και το MapReduce. Για να μπούμε στο αρχείο δίνουμε την εντολή **“sudo nano $HADOOP_HOME/etc/hadoop/hadoop-env.sh”** και βρίσκουμε τη γραμμή που λέει **#export JAVA_HOME=** και βάζουμε την τοποθεσία της JAVA (**“export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64”**) όπως φαίνεται στην Εικόνα 3.1.1 5



Εικόνα 3.1.1 5 Παραμετροποίηση αρχείου hadoop-env.sh

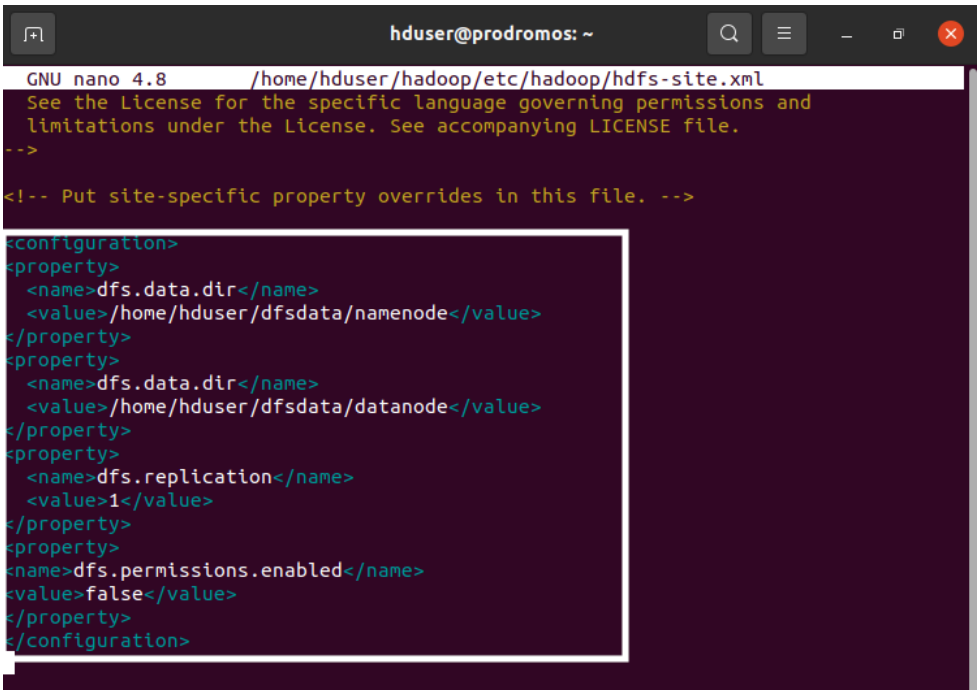
Μόλις τελειώσουμε και με την παραμετροποίηση του hadoop-env.sh αρχείου πάμε να πραγματοποιήσουμε επεξεργασία στα αρχεία core-site.xml, hdfs-site.xml, mapred-site.xml και yarn-site.xml.

- **core-site.xml:** Το αρχείο core-site.xml καθορίζει τις ιδιότητες του HDFS και Hadoop Core. Για να ρυθμιστεί το Hadoop σε Pseudo-Distributed λειτουργία πρέπει να καθοριστεί η διεύθυνση URL για το NameNode και τον προσωρινό κατάλογο που χρησιμοποιεί το Hadoop για τη διαδικασία του MapReduce. Δίνοντας την εντολή **"sudo nano $HADOOP_HOME/etc/hadoop/core-site.xml"** μεταβαίνουμε μέσα στο αρχείο και τοποθετούμε τις εντολές παραμετροποίησης που φαίνονται στην Εικόνα 3.1.1 6 (σημαντικό ότι θα πρέπει να δημιουργηθεί και ένας κατάλογος tmpdata για τα προσωρινά δεδομένα, πραγματοποιείται με την εντολή **"mkdir tmpdata"**).



Εικόνα 3.1.1 6 Παραμετροποίηση core-site.xml

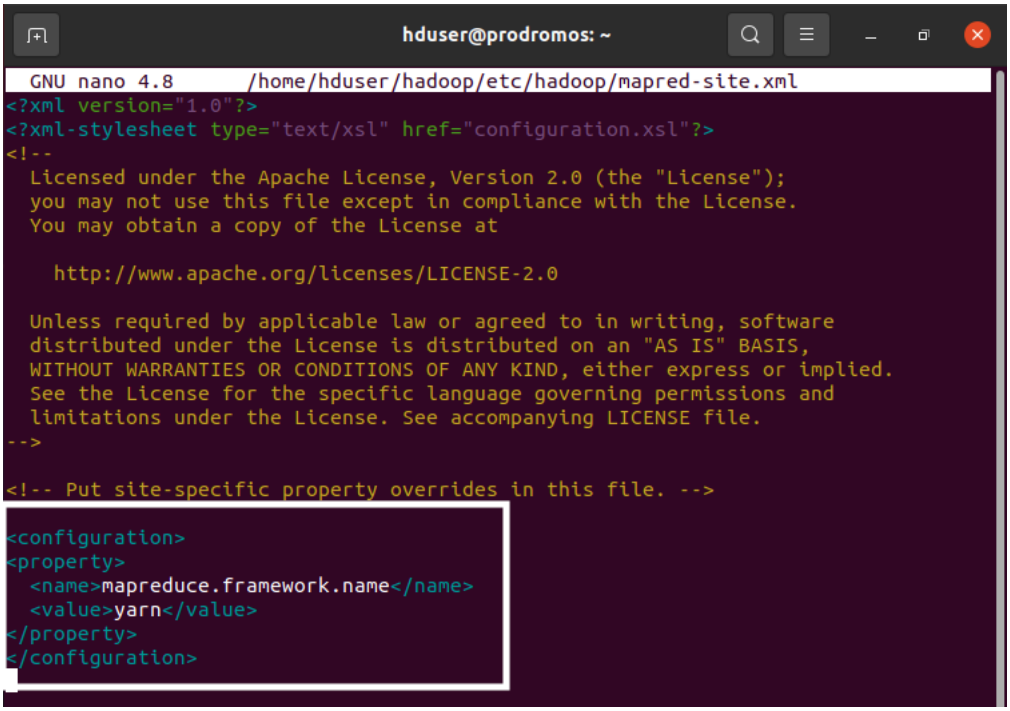
- **hdfs-site.xml:** Το αρχείο hdfs-site.xml χρησιμεύει για τη θέση αποθήκευσης των μεταδεδομένων κόμβων, αρχείων fsimage και επεξεργασίας αρχείων καταγραφής. Μέσα στο αρχείο καθορίζουμε τους καταλόγους αποθήκευσης NameNode και DataNode.Δίνοντας την εντολή **"sudo nano $HADOOP_HOME/etc/hadoop/hdfs-site.xml"** μεταβαίνουμε μέσα στο αρχείο και τοποθετούμε τις εντολές παραμετροποίησης που φαίνονται στην Εικόνα 3.1.1 7 (σημαντικό ότι θα πρέπει να δημιουργηθούν και δύο κατάλογοι dfsdata, ένα για το namenode και ένα για το datanode).



Εικόνα 3.1.1 7 Παραμετροποίηση hdfs-site.xml

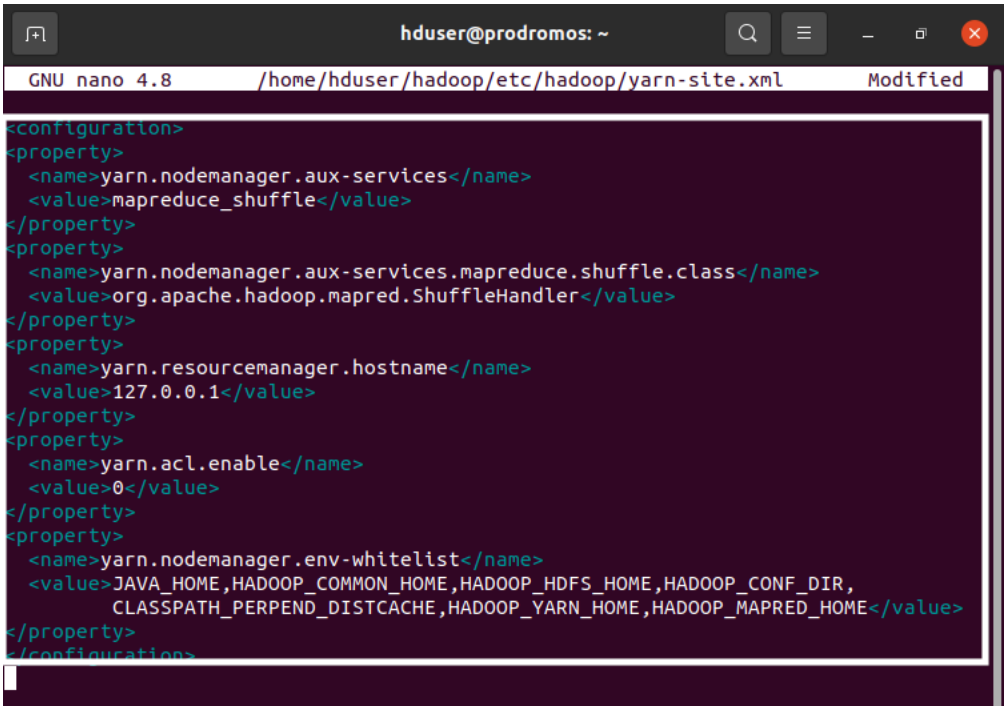
- **mapred-site.xml:** Για την επεξεργασία του αρχείου mapred-site.xml χρησιμοποιούμε την εντολή “sudo nano $HADOOP_HOME/etc/hadoop/mapred-

site.xml" και προσθέτουμε τις εντολές παραμετροποίησης της Εικόνας 3.1.1 8 ώστε να αλλαχτεί η προεπιλεγμένη τιμή ονόματος πλαισίου MapReduce σε νήματα.



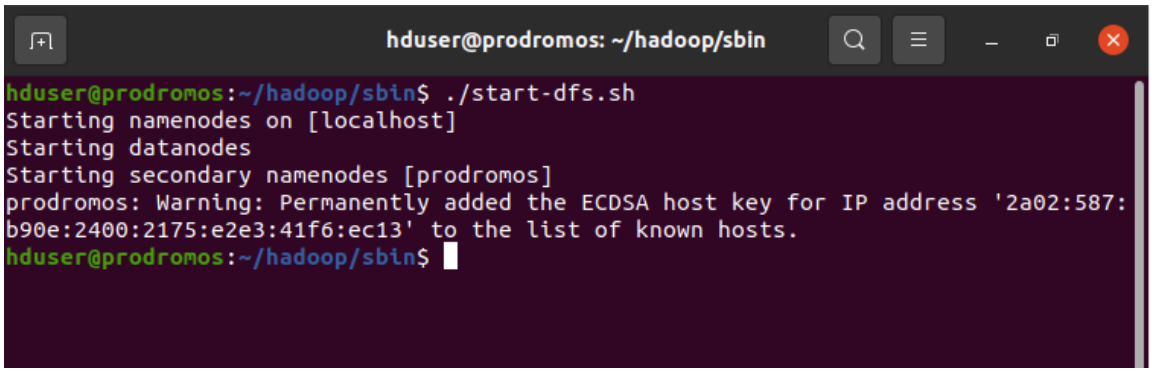
Εικόνα 3.1.1 8 Παραμετροποίηση mapred-site.xml

- **yarn-site.xml:** Το αρχείο yarn-site.xml χρησιμοποιείται για τον καθορισμό ρυθμίσεων σχετικά με το YARN. Περιέχει διαμορφώσεις για το NodeManager, Resource Manager, Containers και Application Master. Για την επεξεργασία του χρησιμοποιούμε την εντολή **"sudo nano $HADOOP_HOME/etc/hadoop/yarn-site.xml"** και προσθέτουμε τις εντολές παραμετροποίησης της Εικόνας 3.1.1 9.



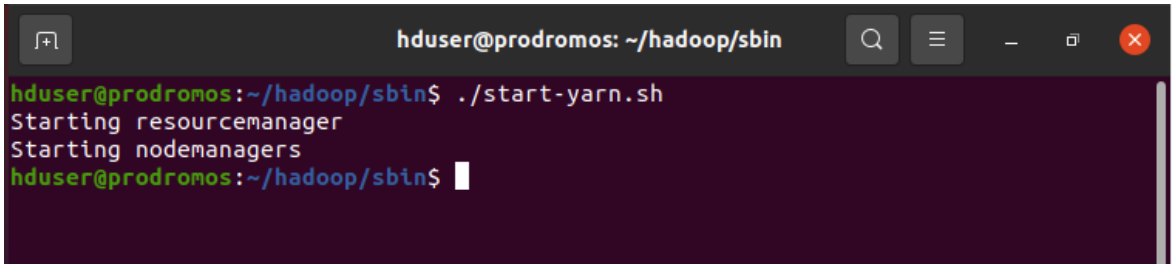
Εικόνα 3.1.1 9 Παραμετροποίηση yarn-site.xml

Μετά τις παραμετροποιήσεις των παραπάνω αρχείων πρέπει να γίνει μία μορφοποίηση για το NameNode, πριν ξεκινήσει τις υπηρεσίες του Hadoop για πρώτη φορά. Αυτό επιτυγχάνεται με την εντολή **"hdfs namenode -format"**. Έπειτα πρέπει να κατευθυνθούμε στον φάκελο hadoop/sbin και να εκτελέσουμε την εντολή **"./start-dfs.sh"** ώστε να ξεκινήσει το NameNode και το DataNode.



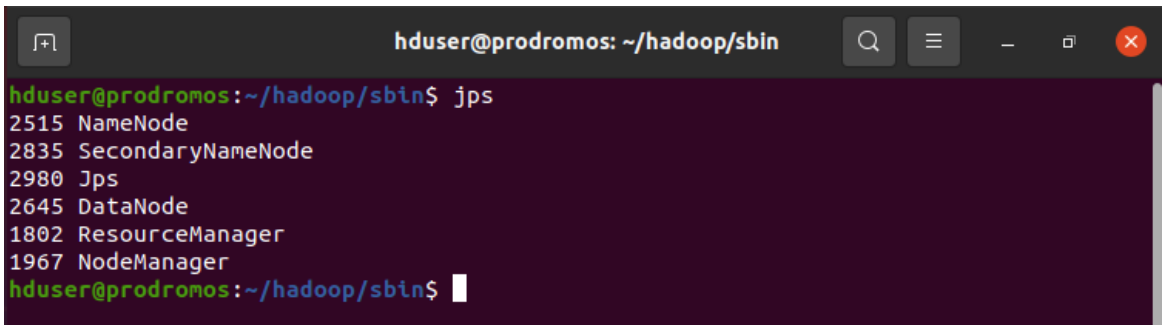
Εικόνα 3.1.1 10 Εκκίνηση namenode, datanode και secondary namenodes

Μόλις ξεκινήσουν και τρέχουν οι namenode, datanode και secondary namenode, δίνουμε την εντολή **"./start-yarn.sh"** ώστε να ξεκινήσουν και τα resourcemanager και nodemanagers του YARN.



Εικόνα 3.1.1 11 Εκκίνηση resourcemanager και nodemanages

Τέλος, πληκτρολογώντας την εντολή **“jps”** βλέπουμε πως όλα τρέχουν σωστά και δεν υπάρχει κανένα λάθος.

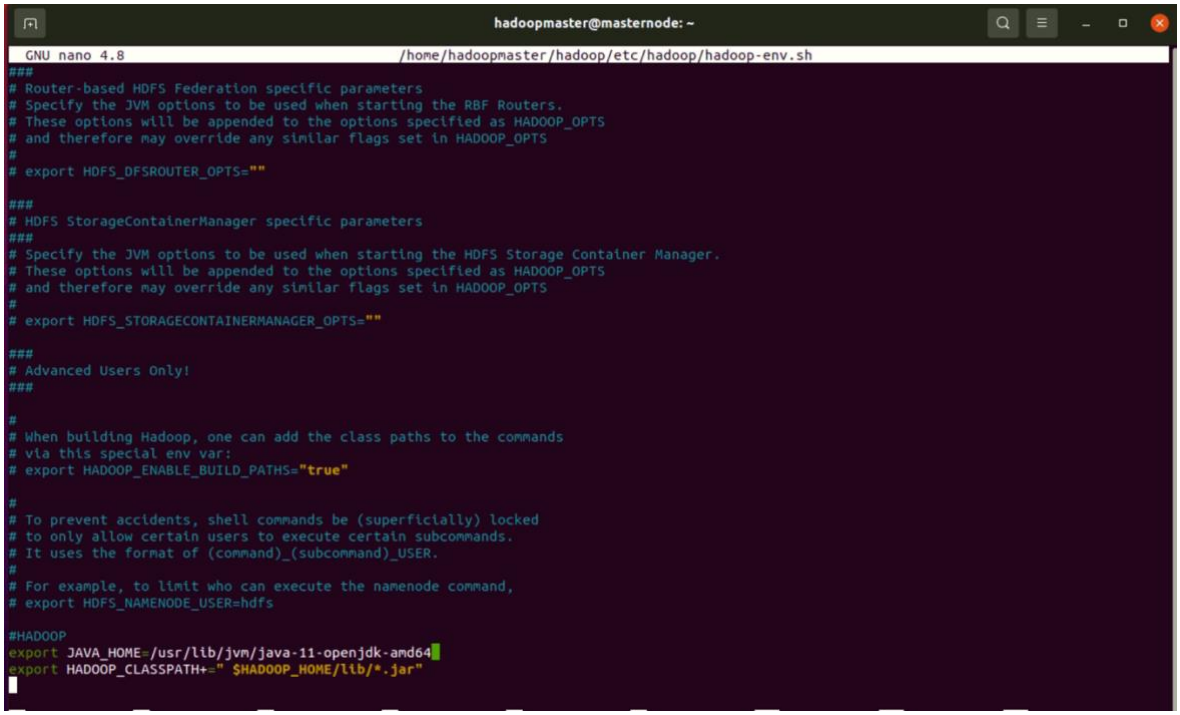


Εικόνα 3.1.1 12 Επαλήθευση πως όλα τρέχουν σωστά

3.1.2 ΕΓΚΑΤΑΣΤΑΣΗ ΚΑΙ ΠΑΡΑΜΕΤΡΟΠΟΙΗΣΗ ΣΕ MULTI NODE

Αρχικά με την εντολή **“sudo apt install ssh”** εγκαθιστούμε το ssh και με την εντολή **“sudo apt install pdsh”** εγκαθιστούμε το pdsh. Για να μην υπάρχει αργότερα πρόβλημα όταν θα πάμε να εκκινήσουμε τους κόμβους, πηγαίνουμε στο κέλυφος bashrc με την εντολή **“sudo nano .bashrc”** ώστε να τροποποιήσουμε το προεπιλεγμένο rcmd pdsh σε ssh και τοποθετούμε την εντολή **“export PDSH_RCMD_TYPE=ssh”.** Βγαίνοντας από το bashrc πρέπει να δώσουμε την εντολή **“source ~/.bashrc”** ώστε να εφαρμοστούν οι αλλαγές που πραγματοποιήσαμε. Για να δημιουργήσουμε μία σύνδεση χωρίς να χρειάζεται να βάζουμε συνέχεια κάποιον κωδικό πρόσβασης δημιουργούμε ένα ζεύγος κλειδιών ssh και ορίζουμε την τοποθεσία που πρέπει να αποθηκευτεί δίνοντας την εντολή **“ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa”**. Έπειτα, με την εντολή **“cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys”** πραγματοποιούμε την εξουσιοδότηση του νέου κλειδιού στον κατάλογο ssh. Για την επαλήθευση ότι όλα έχουν δημιουργηθεί σωστά δίνουμε την εντολή **“ssh localhost”**. Μετά

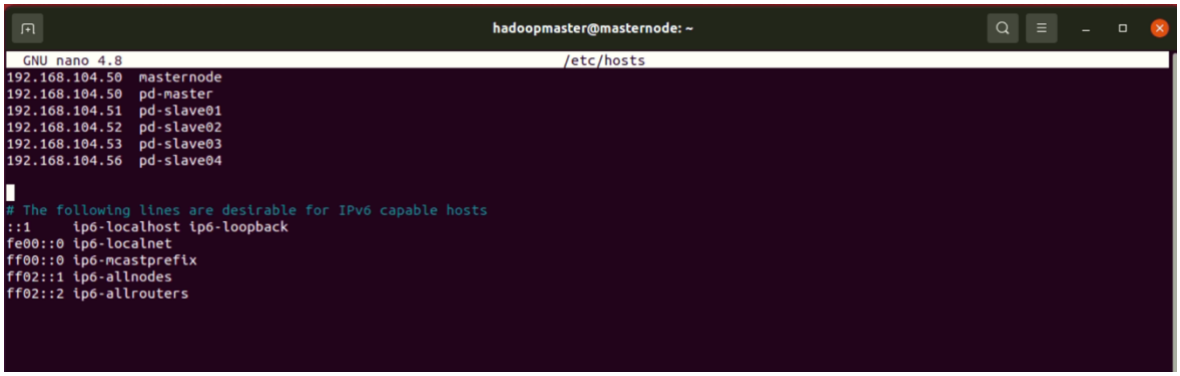
πρέπει να κάνουμε έναν έλεγχο εάν υπάρχει η java, αυτό γίνεται με την εντολή **“java -version; javac -version”**, αν δεν υπάρχει την εγκαθιστούμε με την εντολή **“sudo apt install openjdk-11-jdk”**. Τώρα είμαστε έτοιμοι να κατεβάσουμε το Hadoop αρχείο. Αυτό μπορεί να γίνει με δύο τρόπους, είτε πηγαίνοντας στη σελίδα **https://hadoop.apache.org/** και από την ενότητα Download διαλέξουμε την binary έκδοση που προτιμάμε (στην προκειμένη περίπτωση το Hadoop-3.2.1) ή δίνοντας την εντολή **“sudo wget http://apache.cs.utah.edu/hadoop/common/hadoop-3.2.1/hadoop-3.2.1.tar.gz”**. Έπειτα με την εντολή **“tar xzf hadoop-3.2.1.tar.gz”** αποσυμπιέζουμε το hadoop αρχείο. Ώστε να είναι πιο εύκολη η πλοήγηση αργότερα στους φακέλους του Hadoop δίνουμε την εντολή **“mv ./hadoop-3.2.1 ./hadoop”** και έτσι μετονομάζουμε το αρχείο σε hadoop. Επόμενο βήμα, είναι να πάμε στο περιβάλλον του hadoop, αυτό πραγματοποιείται με την εντολή **“nano ~/hadoop/etc/hadoop/hadoop-env.sh”** και να ορίσουμε το Java Home, με την εντολή **“export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64/”**.



Εικόνα 3.1.2 1 Παραμετροποίηση αρχείου hadoop-env.sh

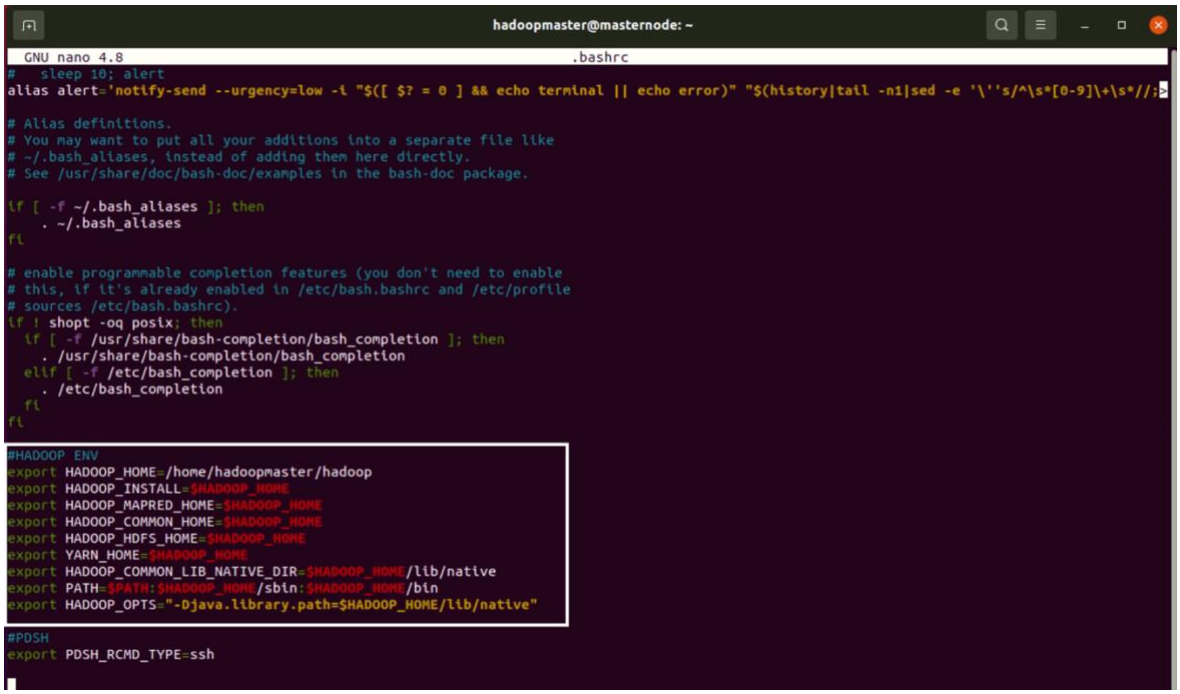
Έπειτα με την εντολή **“sudo mv hadoop /usr/local/hadoop”** θα μετακινήσουμε το hadoop στον κατάλογο /usr/local και με την εντολή **“sudo nano /etc/environment”** θα μεταβούμε στο περιβάλλον του συστήματος ώστε να βάλουμε το παρακάτω path **“PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/usr/local/hadoop/bin:/usr/local/hadoop/sbin"JAVA_HOME="/usr/lib/jvm/j**

ava-11-openjdk-amd64/jre" ”. Τώρα με την εντολή **“sudo adduser hadoopuser”** θα δημιουργήσουμε έναν hadoop χρήστη και στη συνέχεια θα δώσουμε τις παρακάτω τέσσερις εντολές **“sudo usermod -aG hadoopuser hadoopuser”**, **“sudo chown hadoopuser:root -R /usr/local/hadoop/”**, **“sudo chmod g+rwx -R /usr/local/hadoop/”**, **“sudo adduser hadoopuser sudo”**. Επόμενο βήμα είναι, με την εντολή **“ip addr”** να βρούμε την ip του συστήματός μας και έπειτα με την εντολή **“sudo nano /etc/hosts”** να μεταβούμε στο αρχείο των hosts και να ορίσουμε τα ψευδώνυμα master/slaves για κάθε συσκευή και τις αντίστοιχες ip.



Εικόνα 3.1.2 2 Ανάθεση ονομάτων master/slaves στις IP

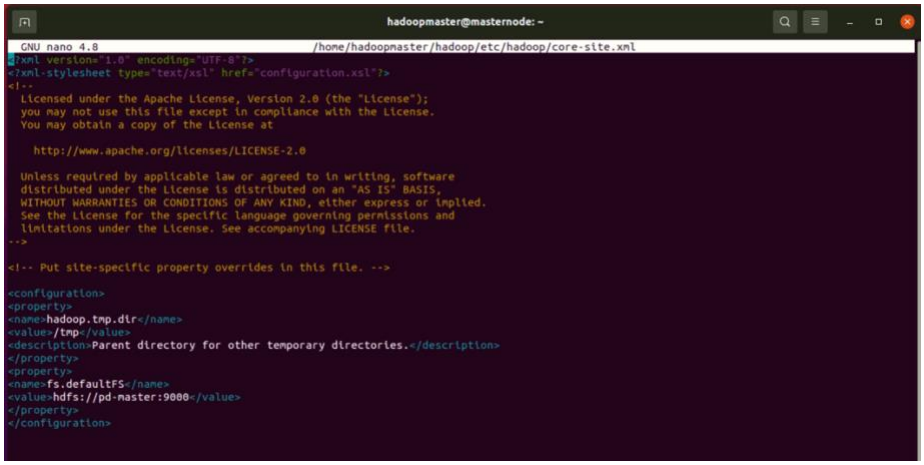
Έπειτα αντιγράφουμε το ssh κλειδί σε όλους τους χρήστες με τις εντολές **“ssh-copy-id hadoopuser@hadoop-master”**, **“ssh-copy-id hadoopuser@hadoop-slave1”**, **“ssh-copy-id hadoopuser@hadoop-slave2”**, συνεχίζουμε τη διαδικασία για όσους slaves έχουμε. Μεταβαίνουμε μετά στο bashrc με την εντολή **“sudo nano .bashrc”** και κάνουμε export τις παρακάτω εντολές (βγαίνοντας από το bashrc δίνουμε την εντολή **“source ~/.bashrc”** ώστε να εφαρμοστούν οι αλλαγές)



Εικόνα 3.1.2 3 Παραμετροποίηση bashrc αρχείου

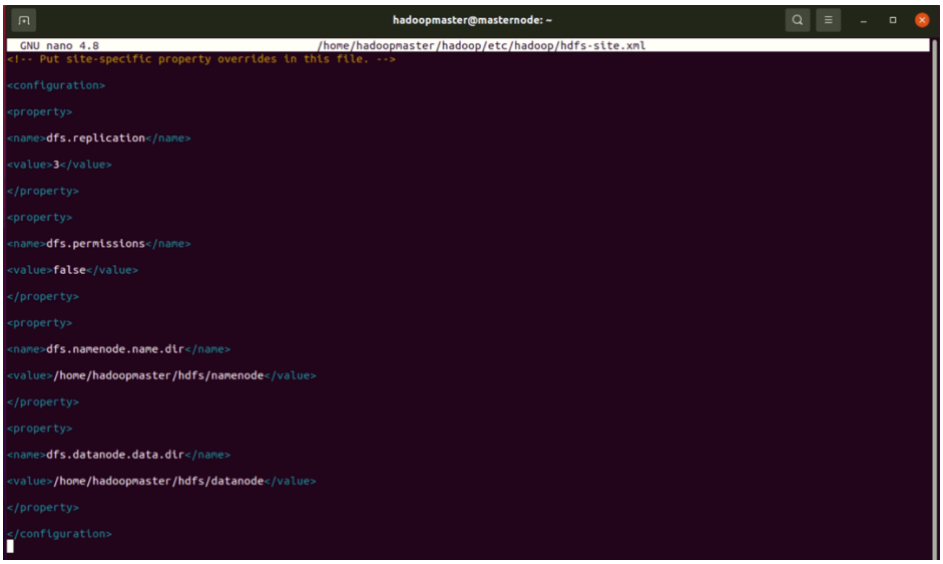
Τώρα ήρθε η ώρα των παραμετροποιήσεων των xml αρχείων και συγκεκριμένα των core-site.xml, hdfs-site.xml, mapred-site.xml και yarn-site.xml.

- **core-site.xml:** Το αρχείο core-site.xml καθορίζει τις ιδιότητες του HDFS και Hadoop Core. Δίνοντας την εντολή
  "sudo nano $HADOOP_HOME/etc/hadoop/core-site.xml" μεταβαίνουμε μέσα στο αρχείο και τοποθετούμε τις εντολές παραμετροποίησης που φαίνονται στην Εικόνα 3.1.2 4, ώστε να ρυθμίσουμε το port για το hadoop-master.



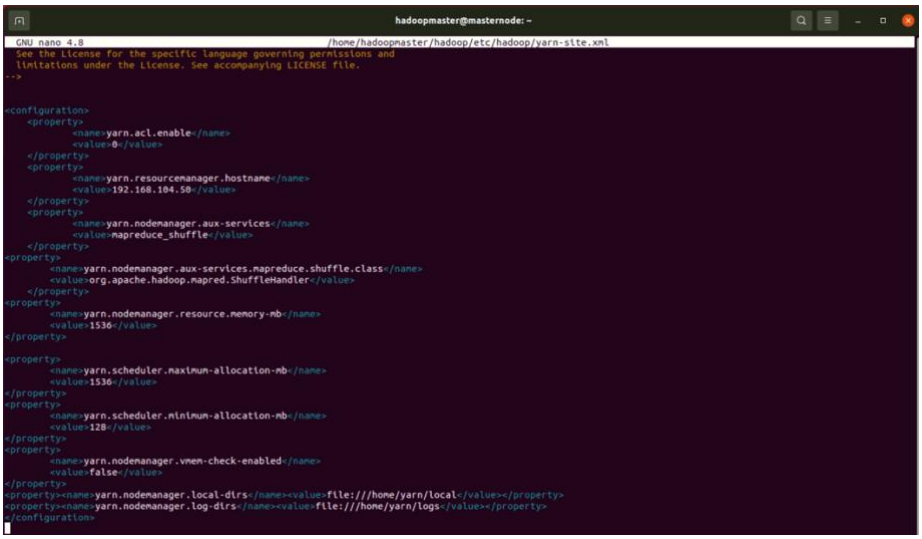
Εικόνα 3.1.2 4 Παραμετροποίηση core-site.xml

- **hdfs-site.xml:** Το αρχείο hdfs-site.xml χρησιμεύει για τη θέση αποθήκευσης των μεταδεδομένων κόμβων, αρχείων fsimage και επεξεργασίας αρχείων καταγραφής. Μέσα στο αρχείο καθορίζουμε τους καταλόγους αποθήκευσης NameNode και DataNode. Δίνοντας την εντολή
  "sudo nano $HADOOP_HOME/etc/hadoop/hdfs-site.xml" μεταβαίνουμε μέσα στο αρχείο και τοποθετούμε τις εντολές παραμετροποίησης της Εικόνας 3.1.2 5 (σημαντικό ότι θα πρέπει να δημιουργηθούν και δύο κατάλογοι dfsdata, ένα για το namenode και ένα για το datanode).



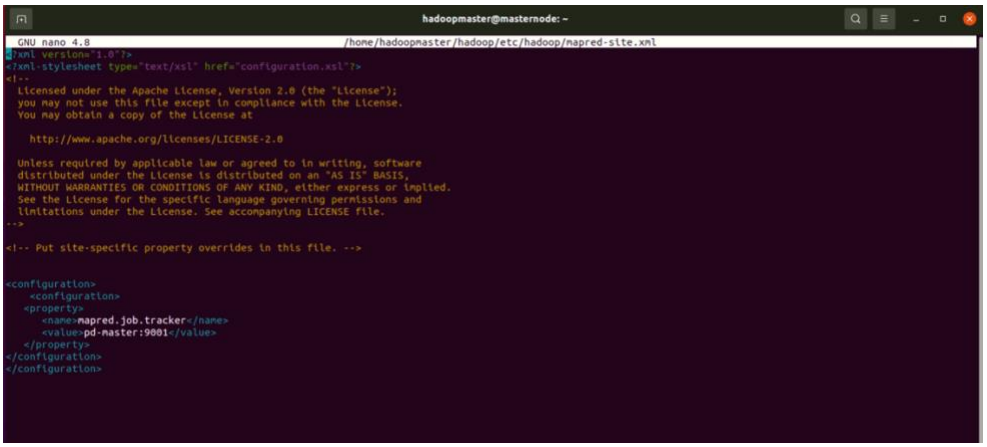
Εικόνα 3.1.2 5 Παραμετροποίηση hdfs-site.xml

- **yarn-site.xml:** Το αρχείο yarn-site.xml χρησιμοποιείται για τον καθορισμό ρυθμίσεων σχετικά με το YARN. Περιέχει διαμορφώσεις για το NodeManager, Resource Manager, Containers και Application Master. Για την επεξεργασία του χρησιμοποιούμε την εντολή **"sudo nano $HADOOP_HOME/etc/hadoop/yarn-site.xml"** και προσθέτουμε τις εντολές παραμετροποίησης της Εικόνας 3.1.2 6



Εικόνα 3.1.2 6 Παραμετροποίηση yarn-site.xml

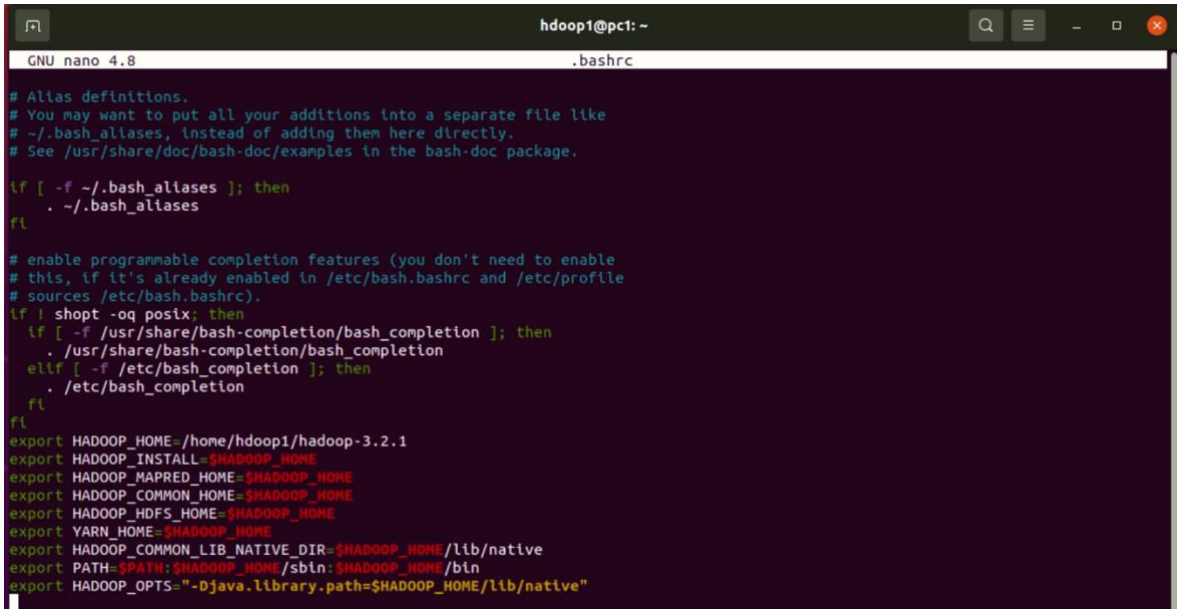
- **mapred-site.xml:** Για την επεξεργασία του αρχείου mapred-site.xml χρησιμοποιούμε την εντολή
  "sudo nano $HADOOP_HOME/etc/hadoop/mapred-site.xml" και προσθέτουμε τις εντολές παραμετροποίησης που φαίνονται στην Εικόνα 3.1.2 7



Εικόνα 3.1.2 7 Παραμετροποίηση mapred-site.xml

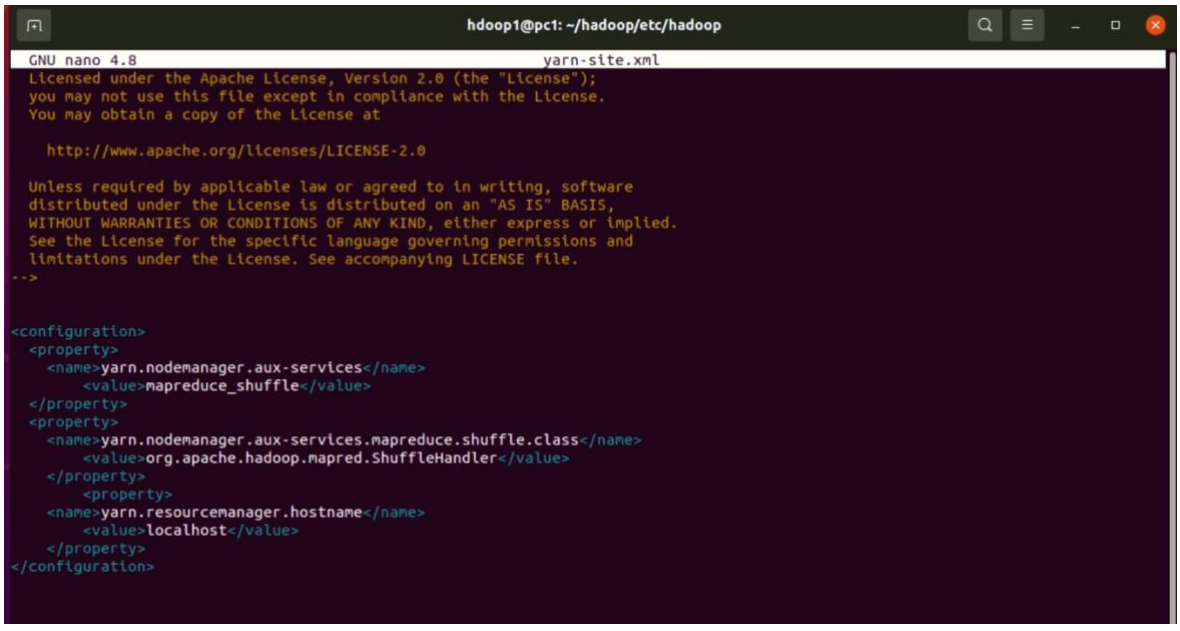
Έπειτα, με την εντολή **"sudo nano ~/hadoop/etc/hadoop/workers"** θα πάμε στο αρχείο των workers και θα τοποθετήσουμε τις εντολές **"hdoop1@pd-slave01"**, **"hdoop1@pd-slave01"**, συνεχίζουμε τη διαδικασία για όσους slaves έχουμε. Μετά τις παραμετροποιήσεις των παραπάνω αρχείων πρέπει να γίνει μία μορφοποίηση για το NameNode, πριν ξεκινήσει τις υπηρεσίες του Hadoop για πρώτη φορά. Αυτό επιτυγχάνεται με την εντολή **"hdfs namenode -format"**. Έπειτα πρέπει να κατευθυνθούμε στον φάκελο hadoop/sbin και να εκτελέσουμε την εντολή **"./start-dfs.sh"** ώστε να ξεκινήσει το NameNode και το DataNode.

Καλό είναι να δώσουμε και την εντολή **“jps”** ώστε να δούμε πως τρέχουν σωστά και πως δεν υπάρχει κάποιο πρόβλημα. Μετά χρειάζεται να ορίσουμε τις εντολές της Εικόνας 3.1.2 8 για να ρυθμίσουμε το yarn.



Εικόνα 3.1.2 8 Ρύθμιση Yarn μέσα στον 1ο slave

Έπειτα, πάμε στο yarn-site.xml των slaves, με την εντολή **“sudo nano hadoop/etc/hadoop/yarn-site.xml”** και προσθέτουμε τις εντολές της Εικόνας 3.1.2 9

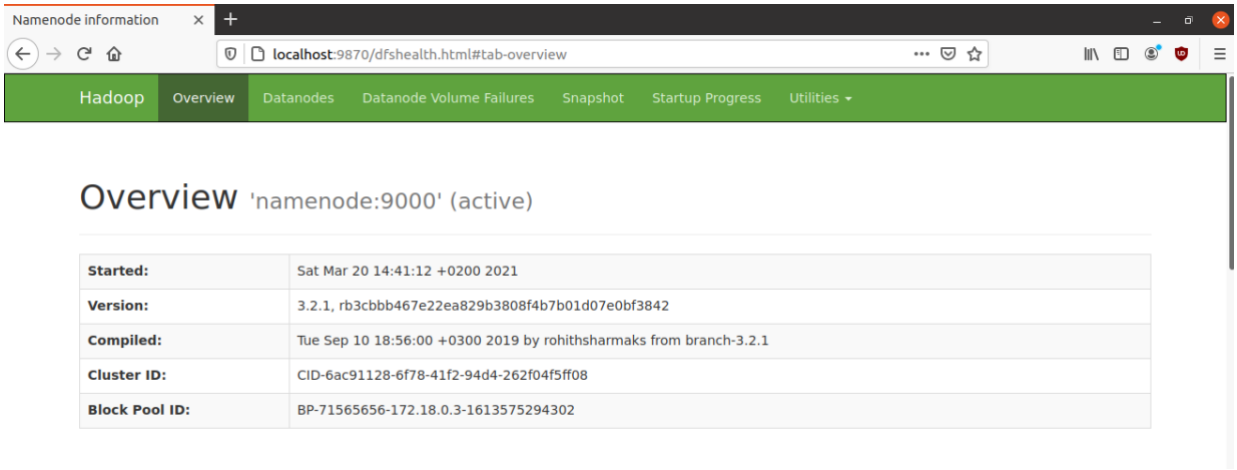


Εικόνα 3.1.2 9 Ρύθμιση yarn-site.xml στον 1ο slave

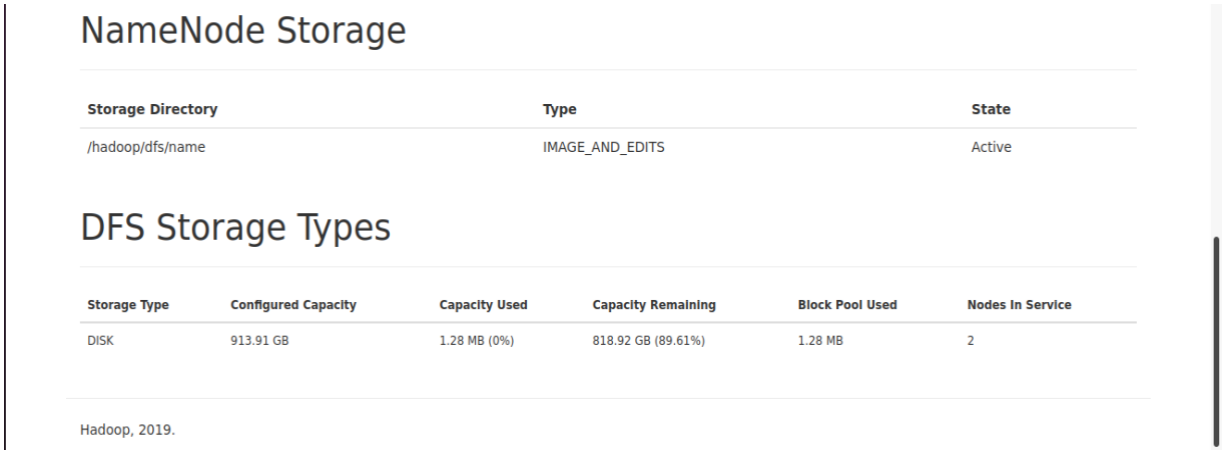
Τέλος, από τον φάκελο hadoop/sbin δίνουμε την εντολή **“./start-yarn.sh”** ώστε να ξεκινήσουν και τα resourcemanager και nodemanagers του YARN.

3.1.3 ΠΕΡΙΒΑΛΛΟΝ ΧΡΗΣΤΗ ΚΑΙ ΔΙΑΧΕΙΡΙΣΗΣ

Για την πρόσβαση στο περιβάλλον του χρήστη στο Hadoop UI, πρέπει να ανοίξουμε τον φυλλομετρητή (browser) που χρησιμοποιούμε και να μεταβούμε στη διεύθυνση URL ή τη διεύθυνση IP του localhost. Ο προεπιλεγμένος αριθμός θύρας 9870 δίνει την πρόσβαση στο Hadoop NameNode UI και μπαίνουμε πληκτρολογώντας **http://localhost:9870**. Στην Εικόνα 3.1.3 1 βλέπουμε κάποιες πληροφορίες σχετικά με την ημερομηνία και την ώρα που μπήκαμε, την έκδοση του Hadoop, το πότε συντάχθηκε, το αναγνωριστικό συστάδας (Cluster ID) και το αναγνωριστικό μπλοκ ομάδας (Block Pool ID).

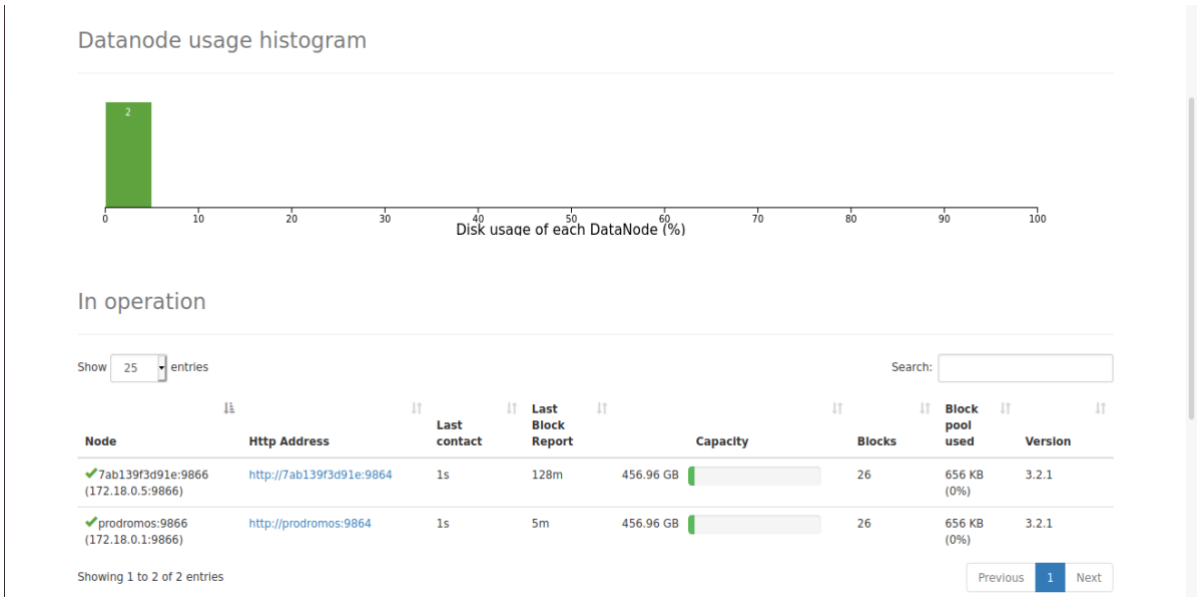


Εικόνα 3.1.3 10 Hadoop NameNode UI Overview



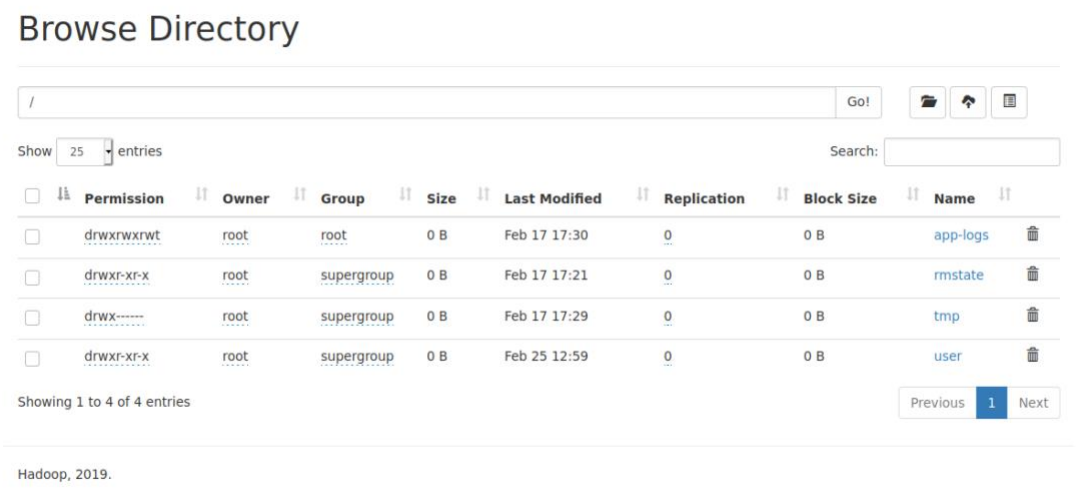
Εικόνα 3.1.3 11 NameNode Storage & DFS Storage Types

Στην παραπάνω εικόνα (Εικόνα 3.1.3 2), βλέπουμε κάποια στοιχεία για τον αποθηκευτικό χώρο του NameNode και τους τύπους αποθήκευσης του dfs. Έπειτα στην Εικόνα 3.1.3 3 απεικονίζονται διάφορα στοιχεία ως προς τη χρήση δίσκου κάθε DataNode, ποια Node βρίσκονται σε λειτουργία και διάφορα ακόμα.



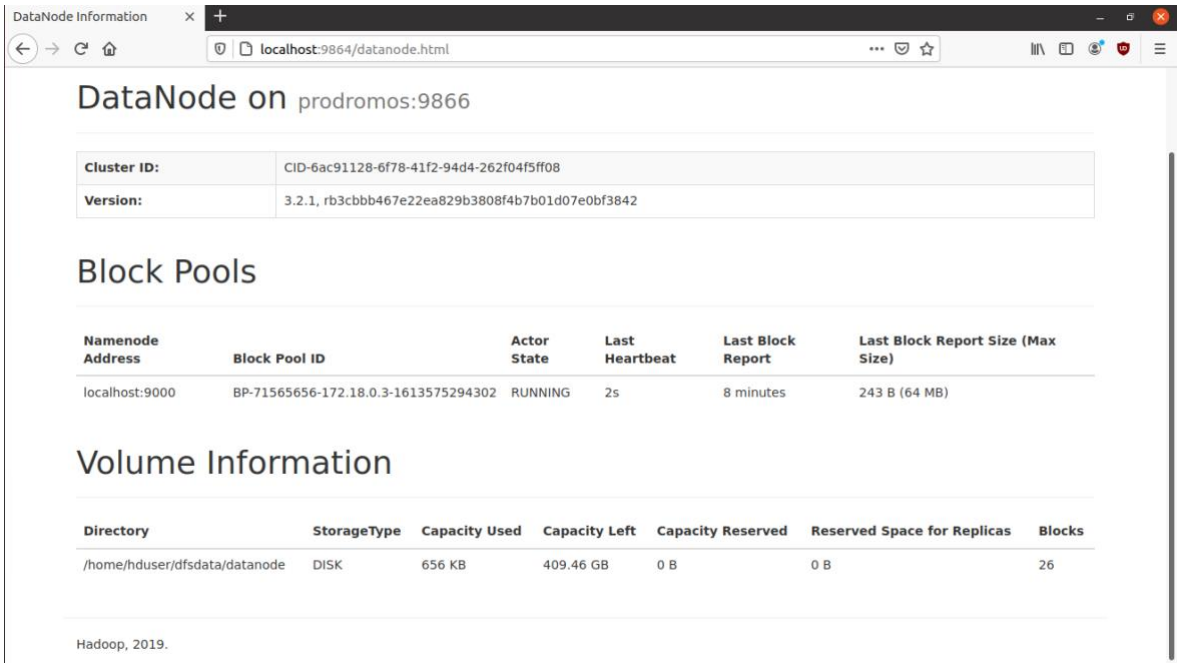
Εικόνα 3.1.3 12 DataNode usage histogram

Στην Εικόνα 3.1.3 4 βλέπουμε τον κατάλογο αναζήτησης, όπου εδώ θα βρίσκονται οι εργασίες που θα δημιουργηθούν.



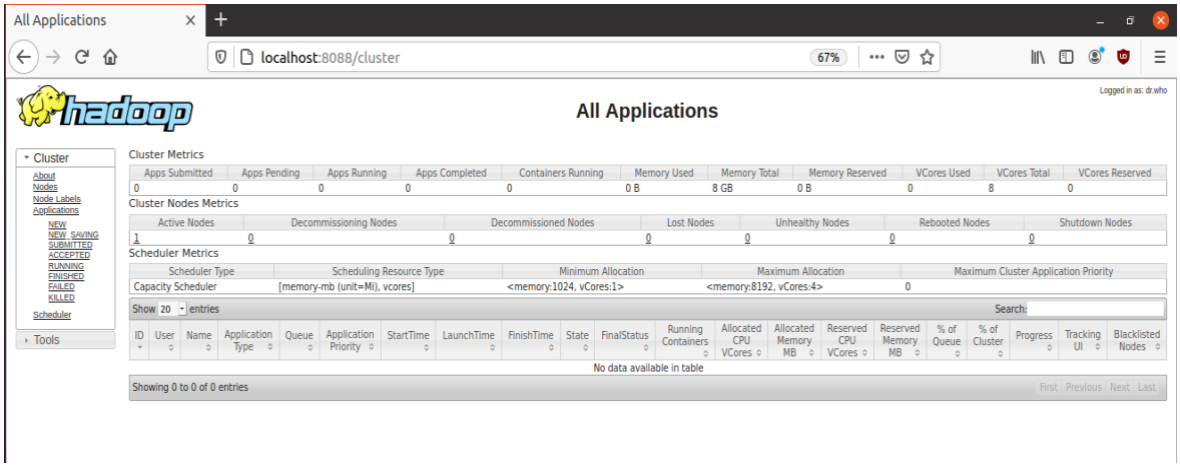
Εικόνα 3.1.3 13 Browse Directory

Η επόμενη θύρα είναι η 9864 και χρησιμοποιείται για πρόσβαση σε μεμονωμένους DataNodes. Για να μεταβούμε θα πρέπει να πληκτρολογήσουμε **http://localhost:9864**. Στην Εικόνα 3.1.3 5 απεικονίζεται το περιβάλλον παρακολούθησης του DataNode, όπου παρέχει κάποιες πληροφορίες για το DataNode, Block Pools και Volume.



Εικόνα 3.1.3 14 Περιβάλλον παρακολούθησης DataNode

Τέλος το YARN Resource Manager βρίσκεται στη θύρα 8088 και πηγαίνουμε πληκτρολογώντας **http://localhost:8088**. Όπως φαίνεται και παρακάτω στην Εικόνα 3.1.3 6 εκεί είναι το περιβάλλον όπου δείχνει τις εργασίες που πραγματοποιούνται στο cluster.

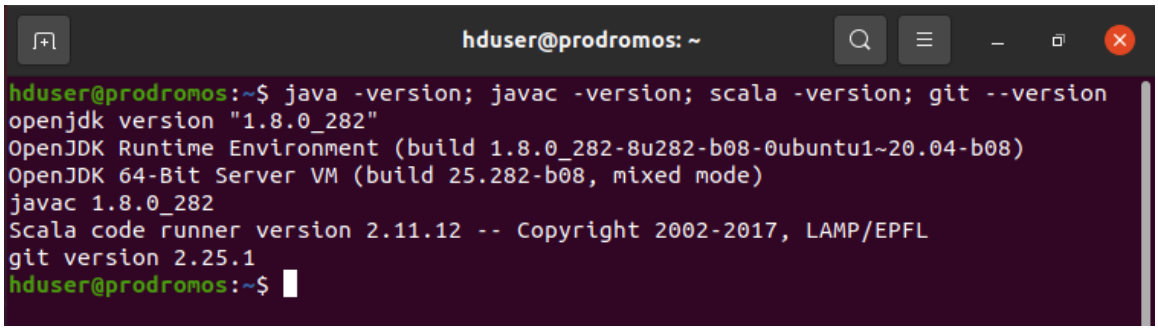


Εικόνα 3.1.3 15 Περιβάλλον παρακολούθησης εργασιών cluster

3.2 ΕΓΚΑΤΑΣΤΑΣΗ APACHE SPARK

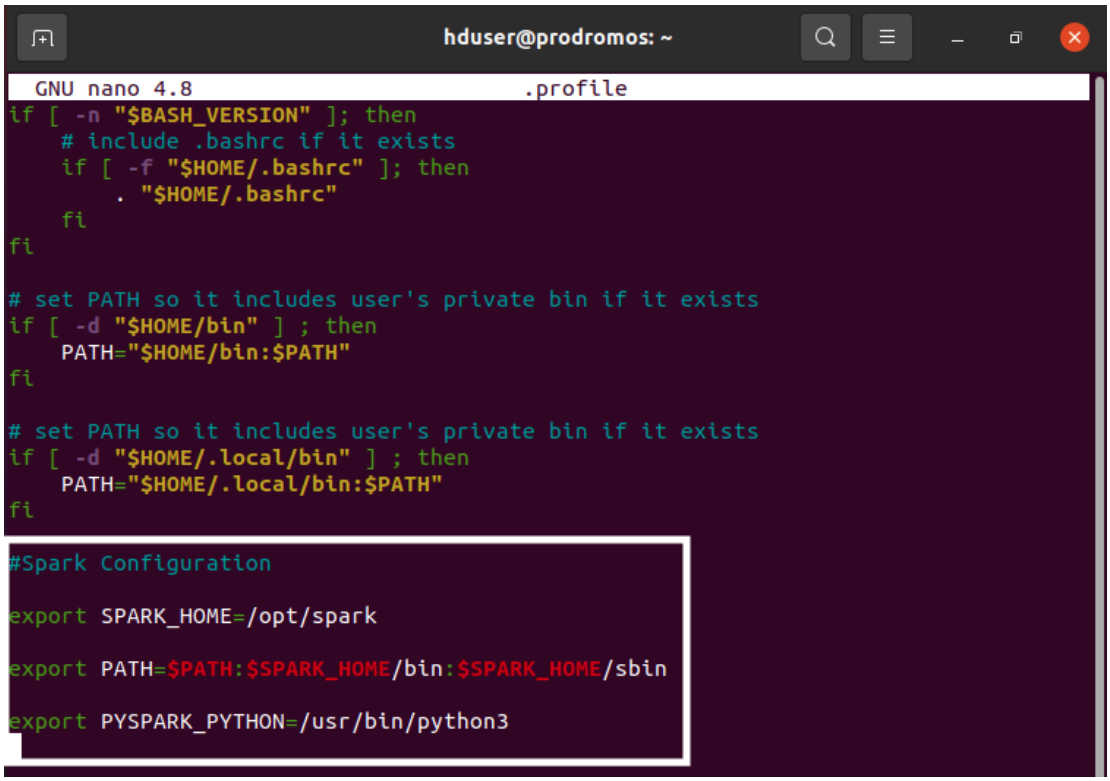
3.2.1 ΕΓΚΑΤΑΣΤΑΣΗ ΚΑΙ ΠΑΡΑΜΕΤΡΟΠΟΙΗΣΗ ΣΕ SINGLE NODE

Για την εγκατάσταση του Apache Spark πρέπει να κατεβάσουμε τρία πακέτα, Java, Scala και Git. Αυτό επιτυγχάνεται πληκτρολογώντας στο terminal την εντολή **"sudo apt install default-jdk scala git -y"**. Για την επιβεβαίωση ότι κατέβηκαν σωστά δίνουμε την εντολή **"java -version; javac -version; scala -version; git –version"** και αν δούμε το μήνυμα της Εικόνας 3.2.1 1 σημαίνει ότι όλα πήγαν σωστά.



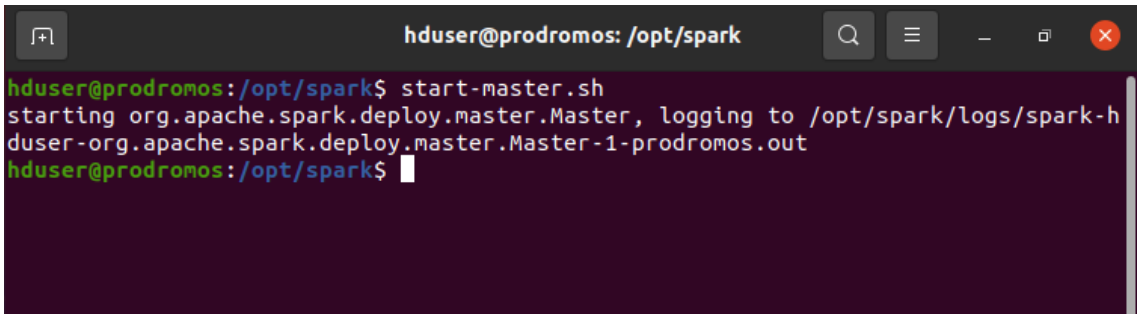
Εικόνα 3.2.1 1 Επιβεβαίωση έκδοσης Java, Scala & Git

Έπειτα πάμε να κατεβάσουμε το Apache Spark. Αυτό μπορεί να γίνει με δύο τρόπους, είτε πηγαίνοντας στη σελίδα **https://spark.apache.org/** και από την ενότητα Download διαλέξουμε την έκδοση που προτιμάμε (στην προκειμένη περίπτωση το Spark 3.1.1 με Pre-Built για Apache Hadoop 3.2) είτε δίνοντας την εντολή **"wget https://downloads.apache.org/spark/spark-3.1.1/spark-3.1.1-bin-hadoop3.2.tgz"**. Μετά με την εντολή **"tar xvf spark-*"** αποσυμπιέζουμε το Spark αρχείο. Για να είναι πιο εύκολη η πλοήγηση αργότερα στους φακέλους του Spark δίνουμε την εντολή **"mv ./spark-3.0.1-bin-hadoop2.7 ./spark"** και έτσι μετονομάζουμε το αρχείο σε spark. Τέλος, μετακινούμε το αρχείο στον κατάλογο opt/spark με την εντολή **"sudo mv spark /opt/spark"**. Το επόμενο βήμα είναι η παραμετροποίηση του περιβάλλοντος του Spark. Με την εντολή **"sudo nano .profile"** πηγαίνουμε στο profile και κάνουμε export τις εντολές που απεικονίζονται στην Εικόνα 3.2.1 2 (Σημαντικό μόλις βγούμε από το profile να δώσουμε την εντολή **"source ~/.profile"** ώστε να εφαρμοστούν οι αλλαγές που πραγματοποιήσαμε).



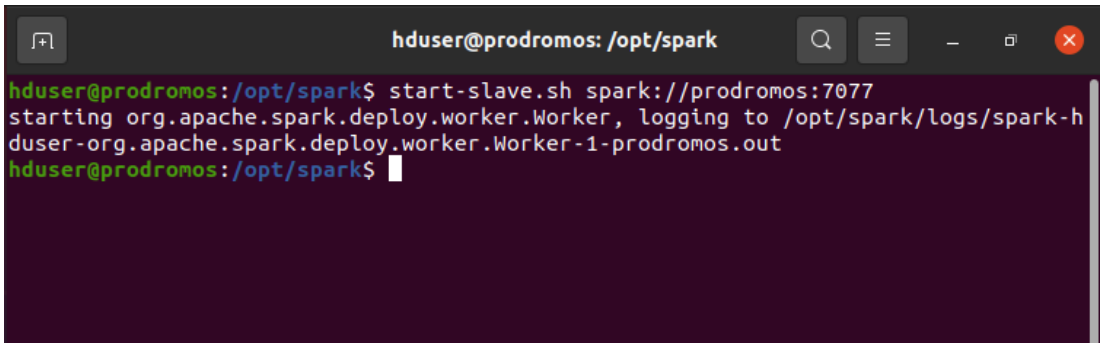
Εικόνα 3.2.1 2 Spark Configuration

Μόλις τελειώσουμε την παραπάνω παραμετροποίηση μπορούμε να ξεκινήσουμε τον master server. Δίνουμε την εντολή **“cd ../../opt/spark”** για να μεταβούμε στον κατάλογο του Spark και πληκτρολογούμε την εντολή **“start-master.sh”**



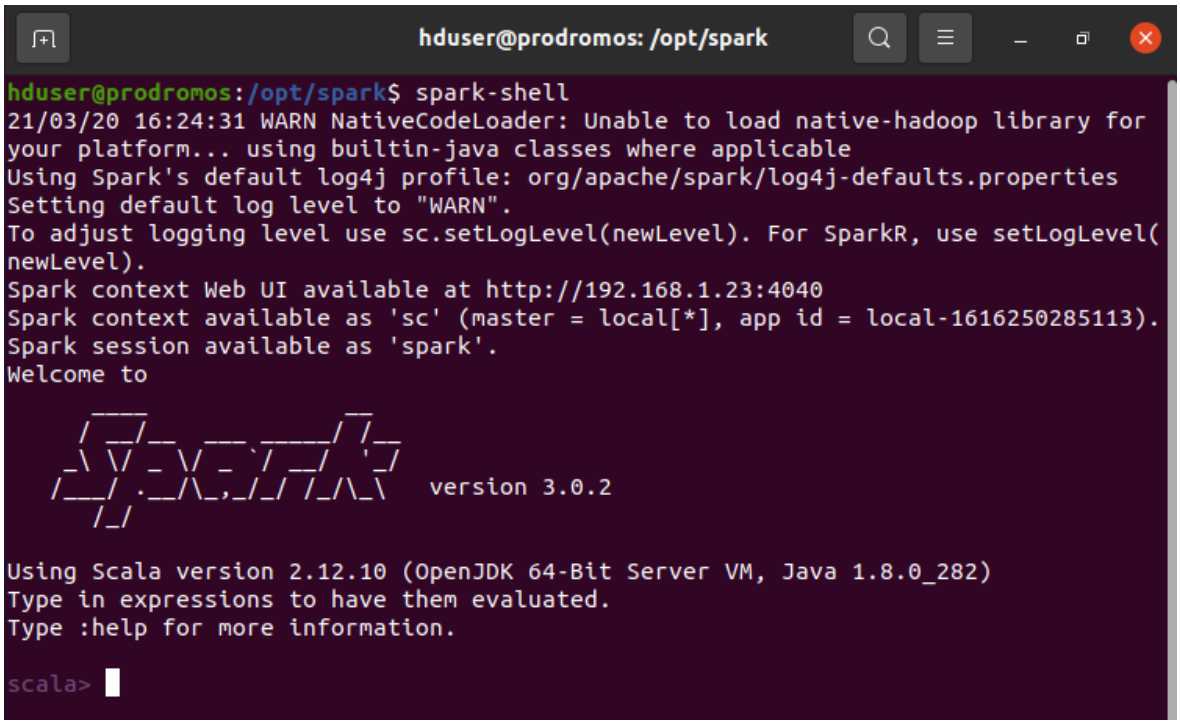
Εικόνα 3.2.1 3 Εκκίνηση master server

Για να ξεκινήσουμε τον slave server δίνουμε την εντολή **“start-slave.sh spark://prodromos:7077”** όπου το prodromos:7077 είναι το master port.

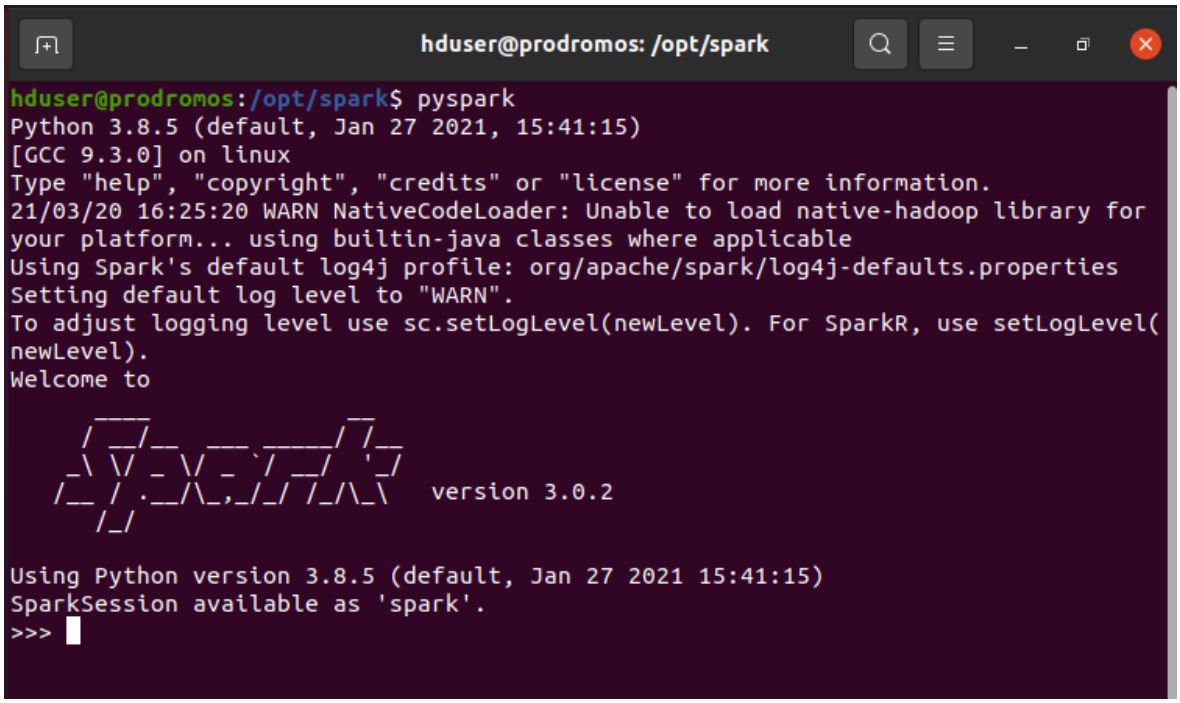


Εικόνα 3.2.1 4 Εκκίνηση slave server

Μόλις τελειώσουμε με την παραμετροποίησης του Spark μπορούμε να αρχίσουμε να χρησιμοποιούμε το spark-shell, εάν θέλουμε να προγραμματίσουμε σε γλώσσα προγραμματισμού Scala ή το pyspark , εάν θέλουμε να προγραμματίσουμε σε γλώσσα προγραμματισμού Python. Το spark-shell ξεκινάει με την εντολή **"spark-shell"** και αν θέλουμε να ορίσουμε εμείς πόσοι πυρήνες θα χρησιμοποιηθούν θα πρέπει να συμπληρώσουμε την εντολή **"-master local[x]"**, όπου το χ συμβολίζει τον αριθμό των πυρήνων. Για να βγούμε από το spark-shell πληκτρολογούμε **":q"**. Το pyspark ξεκινάει με την εντολή **"pyspark"** και αν θέλουμε να βγούμε πληκτρολογούμε **"quit()"**. Στις Εικόνες 3.2.1 5 και 3.2.1 6 απεικονίζεται το spark-shell και pyspark.



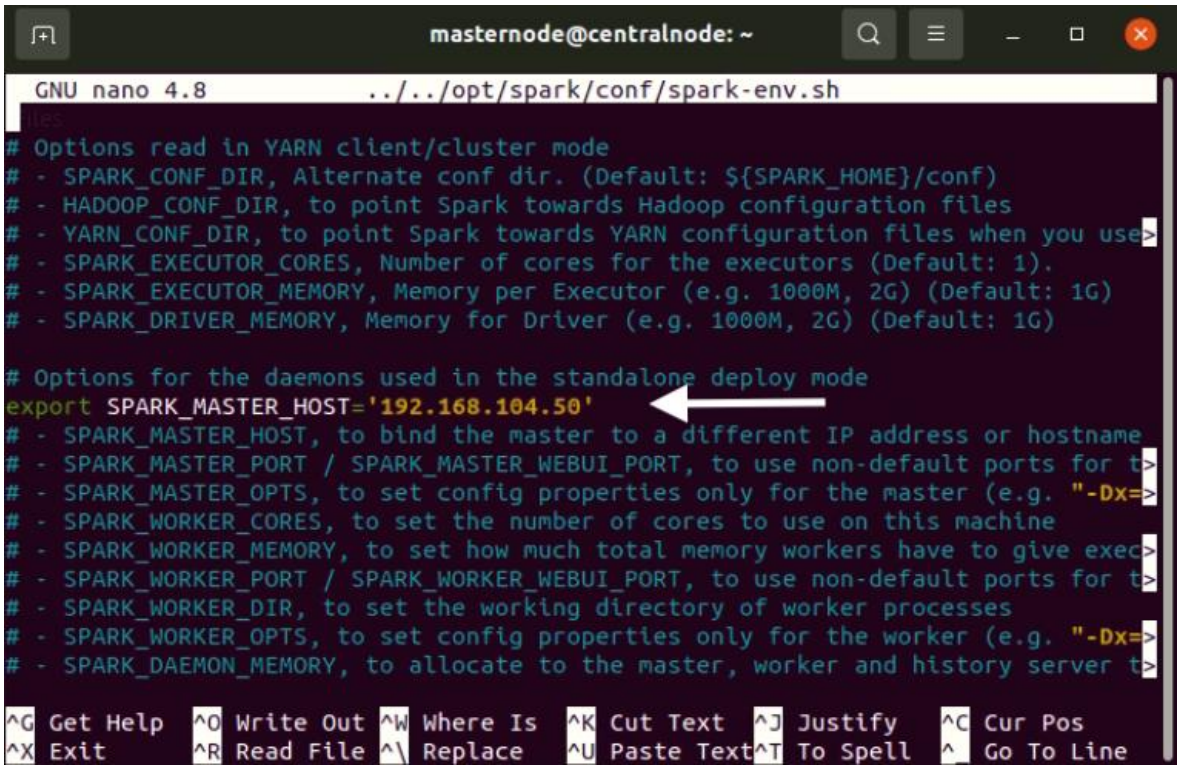
Εικόνα 3.2.1 5 spark-shell



Εικόνα 3.2.1 6 pyspark

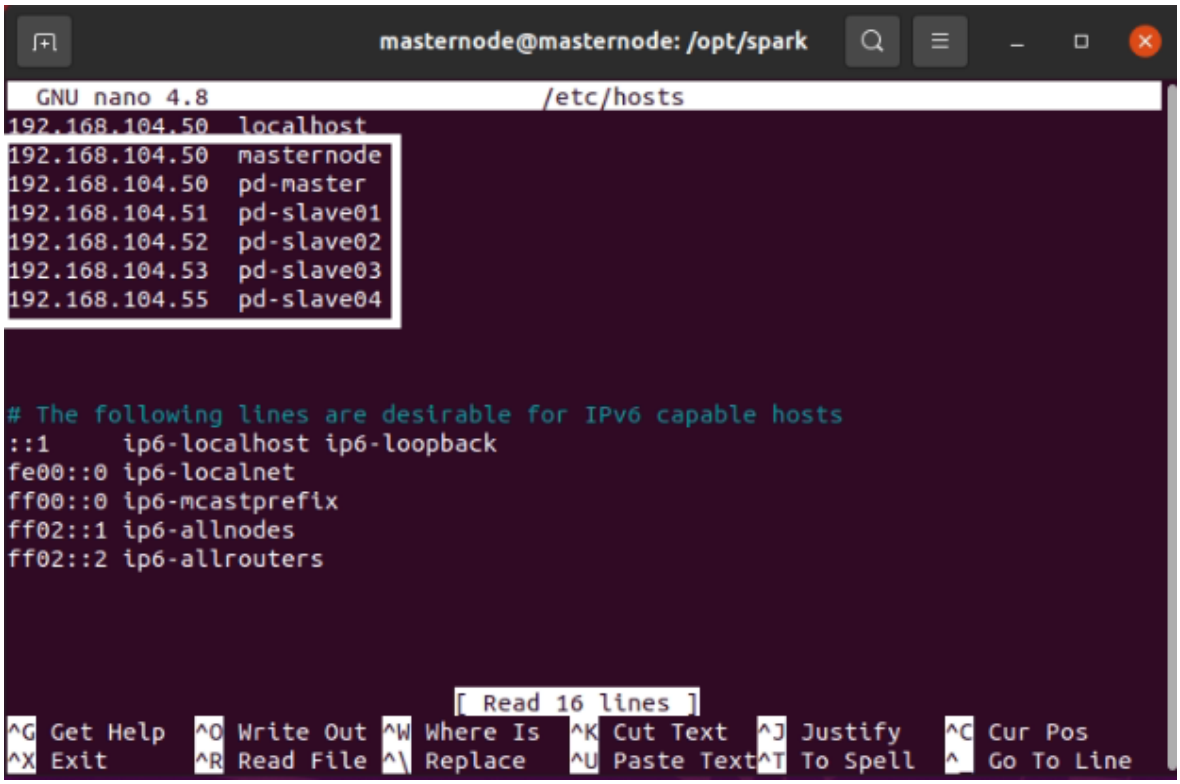
3.2.2 ΕΓΚΑΤΑΣΤΑΣΗ ΚΑΙ ΠΑΡΑΜΕΤΡΟΠΟΙΗΣΗ ΣΕ MULTI NODE

Αρχικά πρέπει να κάνουμε αντιγραφή των περιεχομένων των template αρχείων σε καινούργια shell αρχεία (.sh) με το ίδιο όνομα, αυτό γίνεται με την εντολή **“sudo nano spark-env.template spark-env.sh”**. Εκτελούμε επίσης τις εντολές **“sudo cp spark-defaults.conf.template spark-defaults.conf”** και **“sudo cp slaves.template slaves”**. Έπειτα θα φτιάξουμε τον Master. Για τη διαδικασία αυτή, πρώτα από όλα πρέπει να γίνει επεξεργασία των μεταβλητών του Spark περιβάλλοντος. Πηγαίνουμε οπότε στον φάκελο conf όπου υπάρχει η εγκατάσταση του Spark, δίνοντας την εντολή **“sudo nano ../../opt/spark/conf/spark-env.sh”**. Έπειτα πρέπει να γίνει η ρύθμιση για την ip του master host. Πηγαίνουμε οπότε στο αρχείο spark-env.sh και ορίζουμε το όνομα του master host μαζί με την ip του.



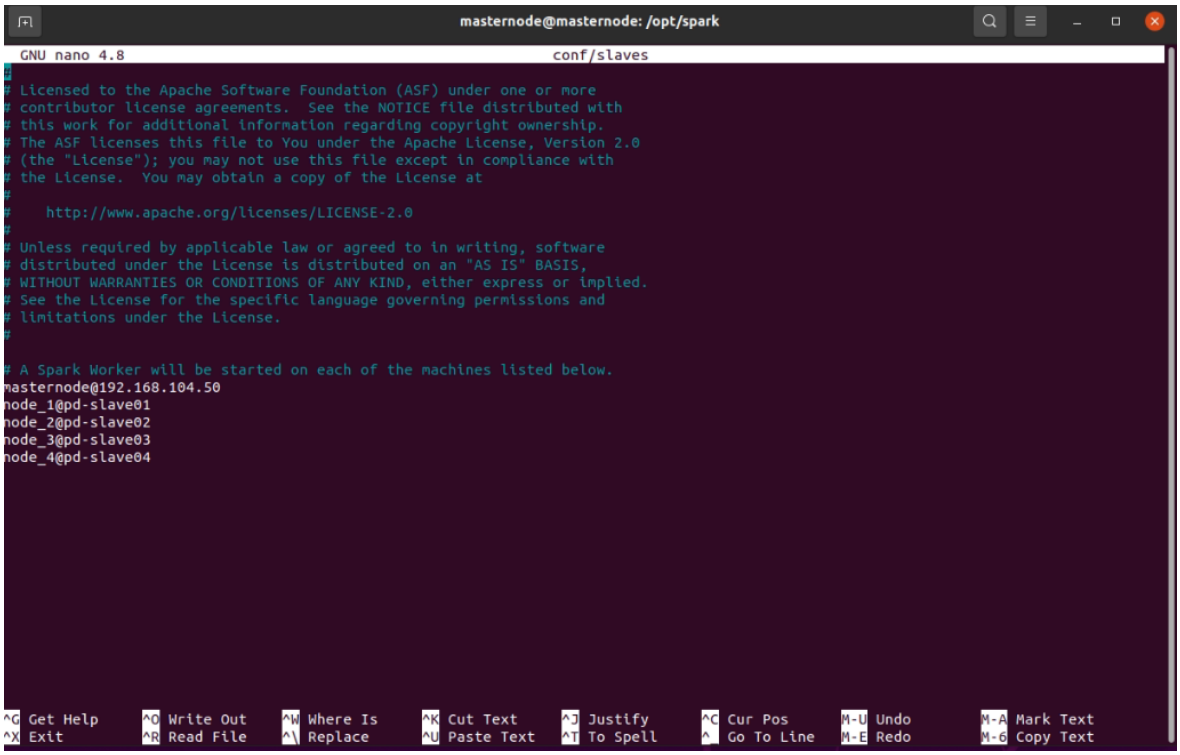
Εικόνα 3.2.2 1 Παραμετροποίηση αρχείου spark-env.sh

Στο επόμενο βήμα πρέπει να ορίσουμε για κάθε μία ip και από ένα όνομα. Δίνοντας την εντολή **“ sudo nano ../../etc/hosts ”** μεταβαίνουμε μέσα στο αρχείο και βάζουμε τα ονόματα master/slaves για τις ip που αντιστοιχούν. Η ίδια διαδικασία πρέπει να γίνει και στους υπολογιστές που θα χρησιμοποιηθούν ως workers. Για να βρούμε τις ip δίνουμε την εντολή **“ip addr”** και η ip που βρίσκουμε (στην περίπτωσή μας 192.168.104.50) θα είναι ο master. Έπειτα με την εντολή **“sudo reboot”** γίνεται επανεκκίνηση του συστήματος ώστε να εφαρμοστούν οι αλλαγές.



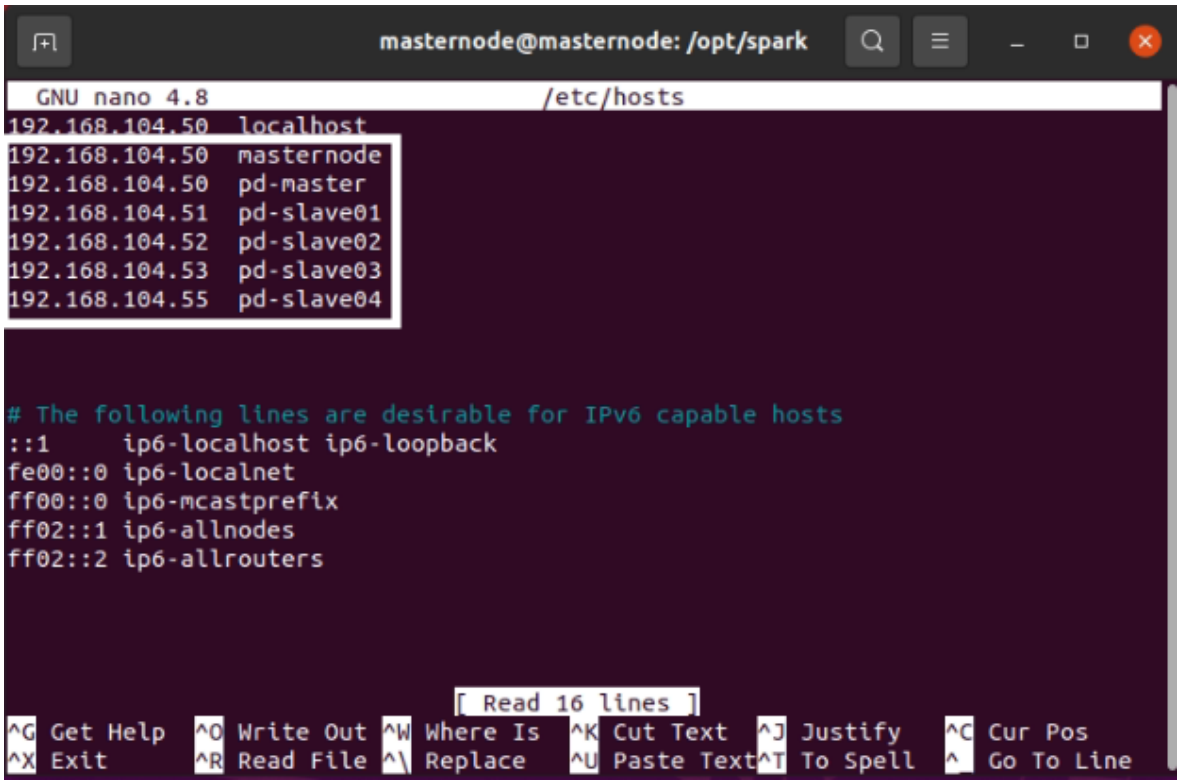
Εικόνα 3.2.2 2 Ανάθεση ονομάτων master/slaves στις IP

Μετά πρέπει να ελέγξουμε εάν έχουμε εγκατεστημένη τη Java και τη Scala. Αυτό πραγματοποιείται με την εντολή **“java -version; javac -version; scala -version”** (στο Single Node τις είχαμε εγκαταστήσει). Έπειτα, με την εντολή **“ssh”** ελέγχουμε εάν το ssh είναι ρυθμισμένο στο μηχάνημά μας. Εάν δεν υπάρχει το εγκαθιστούμε με την εντολή **“sudo apt-get install openssh-server openssh-client”**. Μετά με την εντολή **“ssh-keygen -t rsa -P '' ”** δημιουργούμε ένα νέο ζεύγος κλειδιών και με την εντολή **“ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys”** εξουσιοδοτούμε το νέο κλειδί. Ως τελικό βήμα, πρέπει να αντιγράψουμε το περιεχόμενο του .ssh/id_rsa.pub στο .ssh/authorized_keys όλων των workers. Αυτό επιτυγχάνεται με τις εντολές **“ssh-copy-id user@pd-master”** και **“ ssh-copy-id user@pd-slave1”**. Με την εντολή **“ssh user@pd-slave1”** μπαίνουμε μέσα στον πρώτο slave, ώστε να ελέγξουμε τη σύνδεση με τους slaves. Για να επιστρέψουμε πίσω στον master δίνουμε την εντολή **“exit”**. Έπειτα, πρέπει να προσθέσουμε τη διαδρομή με την τοποθεσία που βρίσκεται το Spark. Δίνουμε την εντολή **“sudo nano ~/.bashrc”** για να μπούμε στο bashrc και κάνουμε **“export PATH=$PATH:/opt/spark/bin”**. Βγαίνοντας δίνουμε την εντολή **“source ~/.bashrc”** για να εφαρμοστεί η αλλαγή. Μετά με την εντολή **“sudo nano ../../opt/spark/conf/slaves”** μπαίνουμε στο slaves και εκεί προσθέτουμε τους τους workers, ώστε ο master node να μπορεί να τους αναγνωρίζει.



Εικόνα 3.2.2 3 Ορισμός των workers

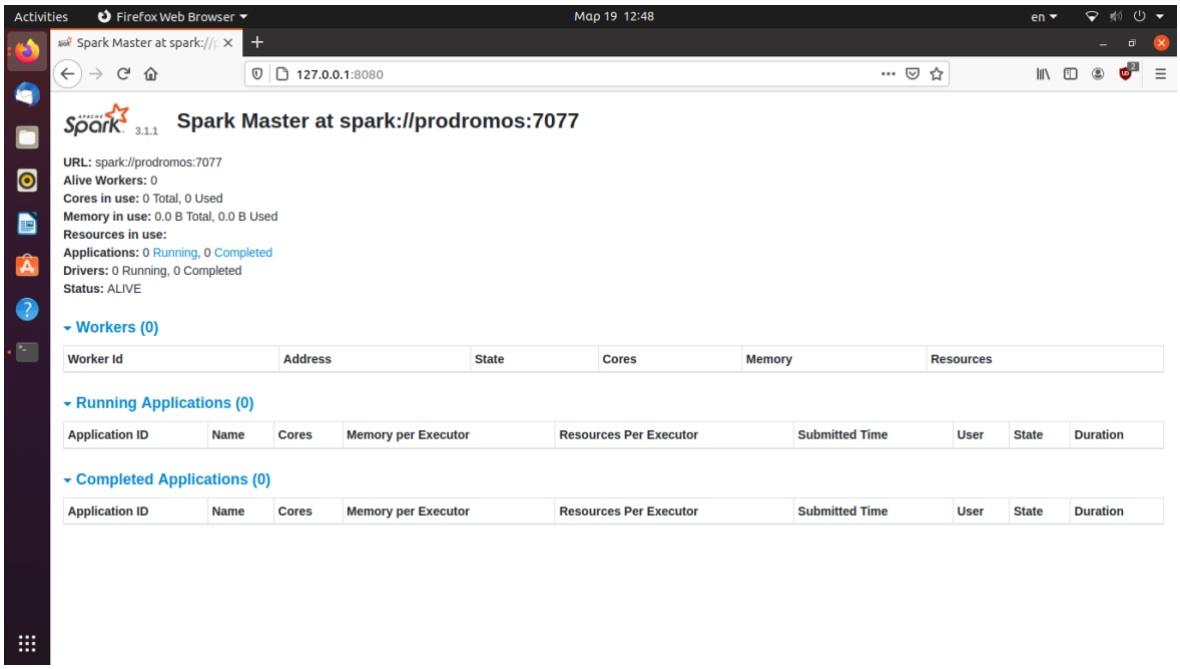
Μόλις τελειώσουμε με τη ρύθμιση του Master, πρέπει να κάνουμε τη ρύθμιση ενός worker. Αρχικά πραγματοποιούμε αντιγραφή των template αρχείων παρόμοια με την αντιγραφή που έγινε στον κεντρικό κόμβο. Έπειτα πηγαίνουμε στο αρχείο spark-env.sh με την εντολή **“sudo nano ../../opt/spark/conf/spark-env.sh”** και ορίζουμε τον master host μαζί με την ip του κάνοντας, **“export SPARK_MASTER_HOST = ‘YOUR_MASTER_HOST_IP’ ”**. Επιπρόσθετα, πρέπει να τοποθετήσουμε τα μηχανήματα στο αρχείο hosts. Δίνουμε την εντολή **“sudo nano ../../etc/hosts”** για να μπούμε στο αρχείο και προσθέτουμε τις ip με τα ονόματα των υπολογιστών που απεικονίζονται στην Εικόνα 3.2.2 4, ώστε να φτιάξουμε ίδια ονόματα για τα μηχανήματα του cluster network με αυτά του master node. Εάν θέλουμε παραπάνω από έναν worker επαναλαμβάνουμε τη διαδικασία και για τους υπόλοιπους. Μόλις τελειώσουμε και τη ρύθμιση του worker μπορούμε να ξεκινήσουμε το cluster network, δίνοντας την εντολή **“../../opt/spark/sbin/start-all.sh”**



Εικόνα 3.2.2 4 Διαμόρφωση host αρχείου

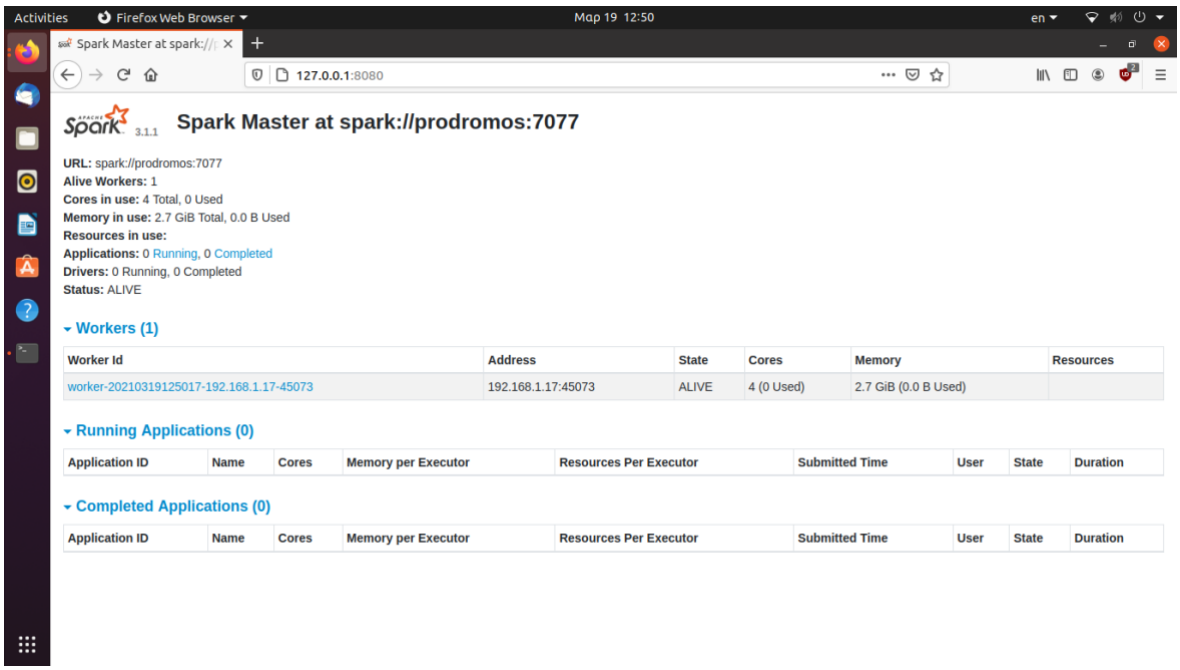
3.2.3 ΠΕΡΙΒΑΛΛΟΝ ΧΡΗΣΤΗ ΚΑΙ ΔΙΑΧΕΙΡΙΣΗΣ

Για την πρόσβαση στο περιβάλλον του χρήστη του Spark Web, πρέπει να ανοίξουμε τον φυλλομετρητή (browser) που χρησιμοποιούμε και να μεταβούμε στη διεύθυνση URL ή τη διεύθυνση IP του localhost στη θύρα 8080. Πληκτρολογώντας οπότε **http://127.0.0.1:8080/** μπαίνουμε στο περιβάλλον. Έχοντας ξεκινήσει τον master server αυτό που αντικρίζουμε στο περιβάλλον χρήστη και διαχείρισης είναι η παρακάτω εικόνα, η Εικόνα 3.2.3 1, όπου φαίνεται η έκδοση του Spark που χρησιμοποιούμε, το URL, πόσοι workers είναι ενεργοποιημένοι, πόσοι πυρήνες, μνήμη και πόροι χρησιμοποιούνται. Επιπρόσθετα απεικονίζονται οι εφαρμογές και οι drivers που τρέχουν και έχουν ολοκληρωθεί και η κατάσταση του Spark.



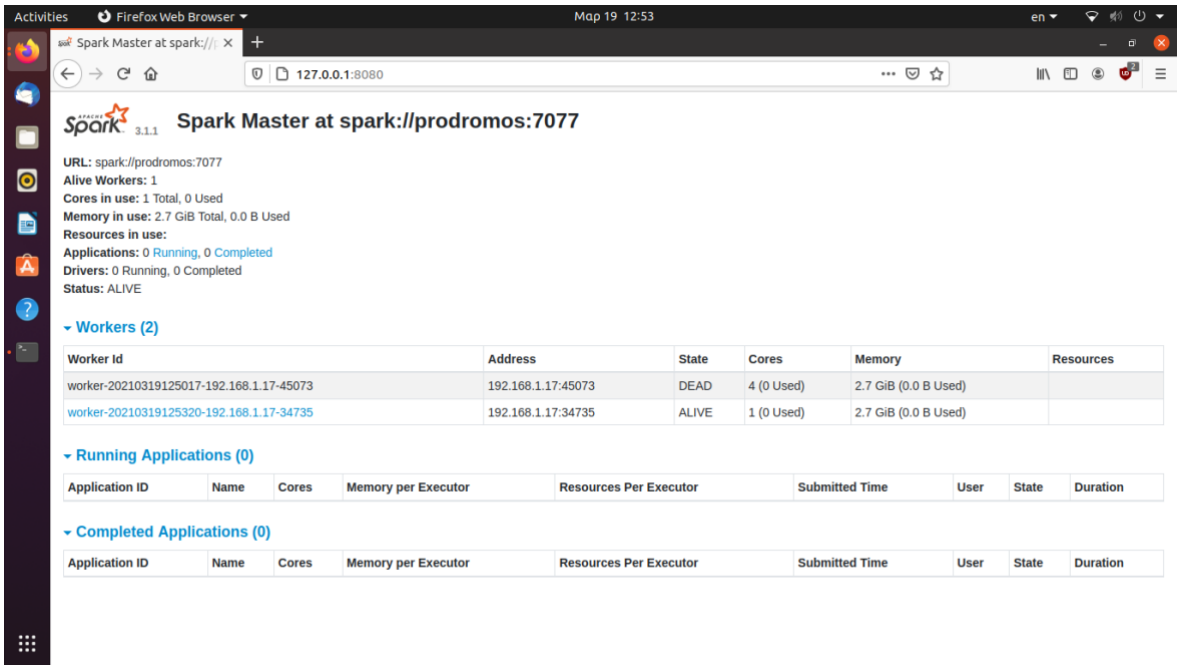
Εικόνα 3.2.3 1 Περιβάλλον χρήστη και διαχείρισης Spark (start-master.sh)

Ενεργοποιώντας και τον slave server (όπως φαίνεται στην Εικόνα 3.2.3 2) βλέπουμε στο περιβάλλον να έχει συμπληρωθεί ένας worker με το ID του, τη διεύθυνσή του, την κατάστασή του, τους πυρήνες και την μνήμη που χρησιμοποιεί.



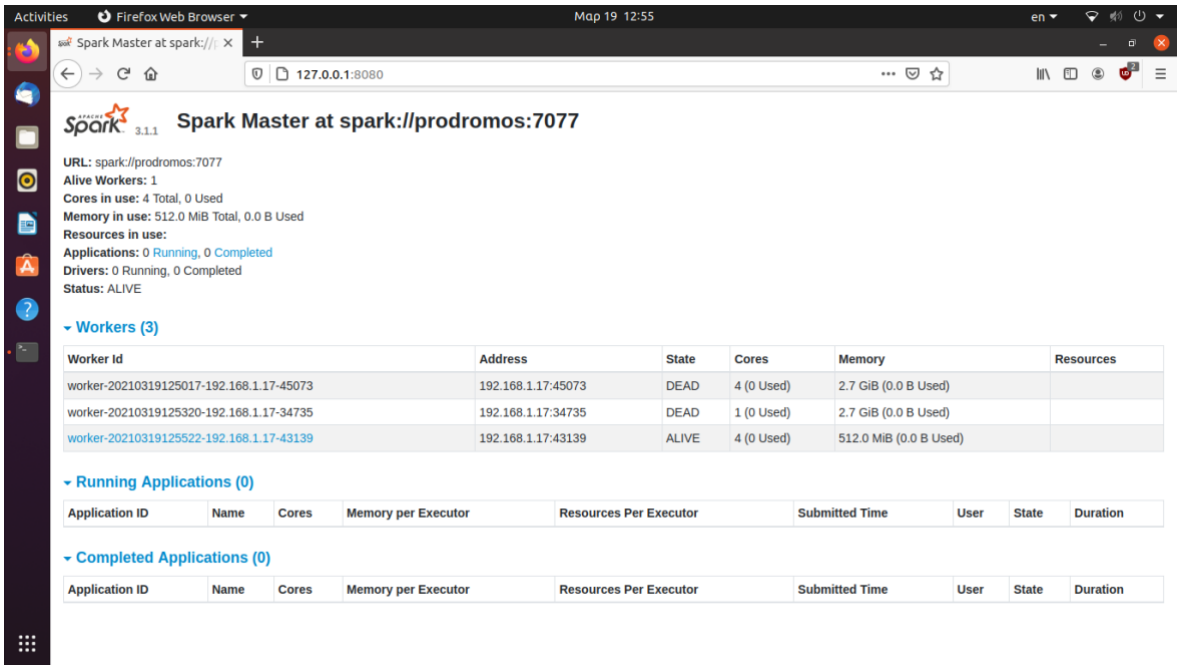
Εικόνα 3.2.3 2 Περιβάλλον χρήστης και διαχείρισης Spark (start-slave.sh)

Κατά την εκκίνηση η προεπιλεγμένη ρύθμιση του worker είναι η χρήση όλων των διαθέσιμων πυρήνων. Μία επιλογή ακόμα, είναι να καθορίσουμε οι ίδιοι την κατανομή πόρων για τους workers. Αυτό επιτυγχάνεται με την εντολή **“start-slave.sh -c 1 spark://prodromos:7077”**, όπου το -c 1 σημαίνει πόσοι πυρήνες θέλουμε να χρησιμοποιηθούν.



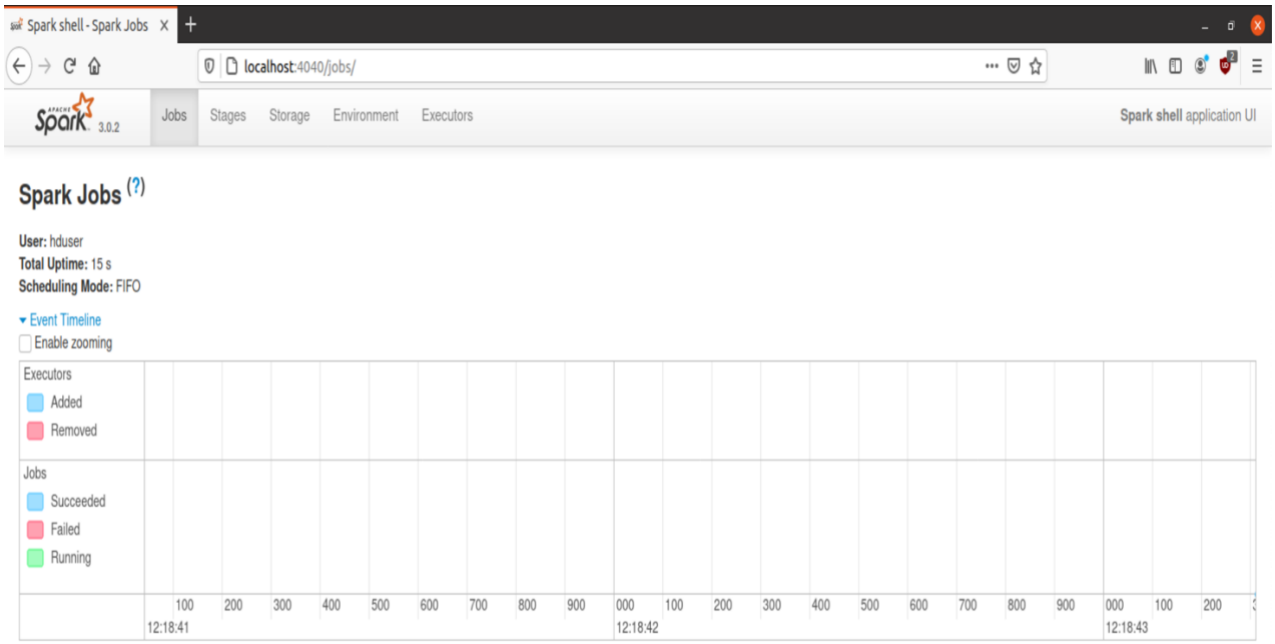
Εικόνα 3.2.3 3 Εκκίνηση worker χρησιμοποιώντας 1 πυρήνα

Με τον ίδιο τρόπο μπορούμε να ορίσουμε πόση μνήμη θέλουμε να διαθέσουμε. Αυτό γίνεται με την εντολή **“start-slave.sh -m 512M spark://prodromos:7077”**, όπου το -m 512M σημαίνει ότι θέλουμε να διαθέσουμε 512 MB. Με την προεπιλεγμένη ρύθμιση χρησιμοποιούμε όση μνήμη διαθέτει το μηχάνημα που έχουμε μείον 1 GB.

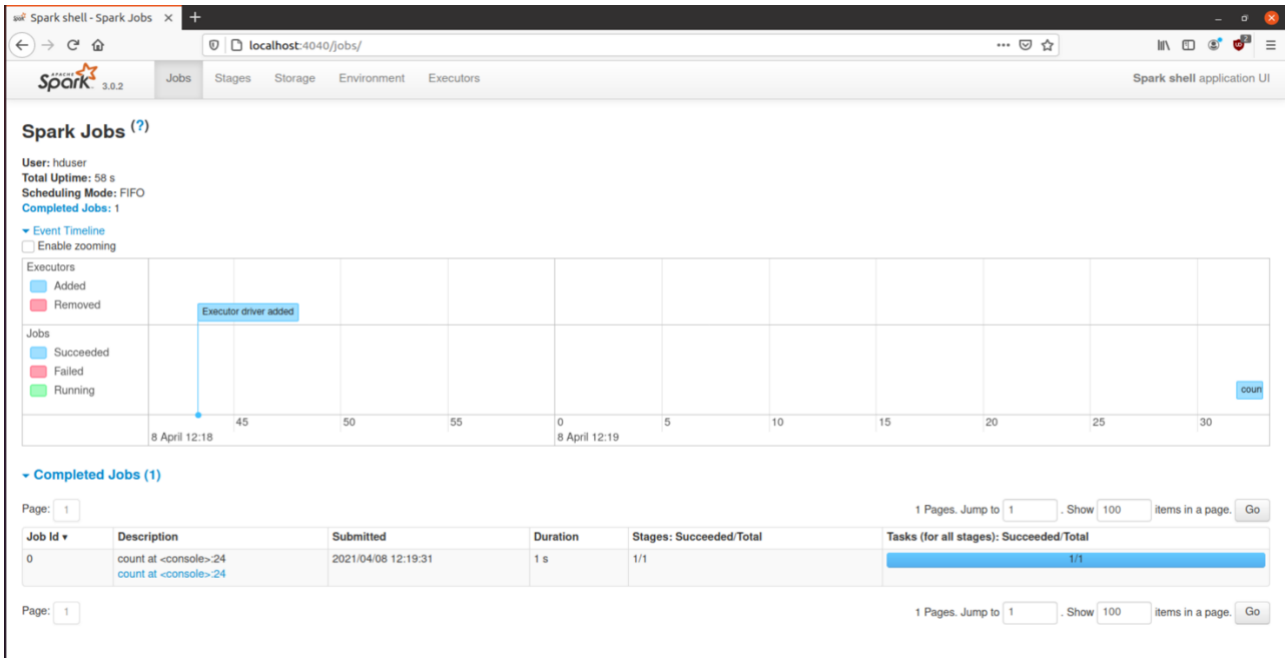


Εικόνα 3.2.3 4 Εκκίνηση worker χρησιμοποιώντας 512 MB

Εκτός από το παραπάνω περιβάλλον, υπάρχουν άλλα δύο, αυτά του spark-shell και του pyspark. Για να μεταβούμε σε αυτά πληκτρολογούμε στον φυλλομετρητή (browser) τη διεύθυνση **http://127.0.0.1:4040/**. Εκεί παρουσιάζονται αρχικά το όνομα του χρήστη, ο συνολικός χρόνος που είναι μέσα στο shell, η μέθοδος για την εξυπηρέτηση των εργασιών, η οποία είναι η FIFO (First In First Out) και οι ολοκληρωμένες εργασίες που έγιναν. Παρακάτω θα παρουσιαστούν κάποιες εικόνες (Εικόνα 3.2.3.5, Εικόνα 3.2.3.6, Εικόνα 3.2.3.7, Εικόνα 3.2.3.8) από το περιβάλλον πριν και αφότου εκτελέσουμε μία εργασία.

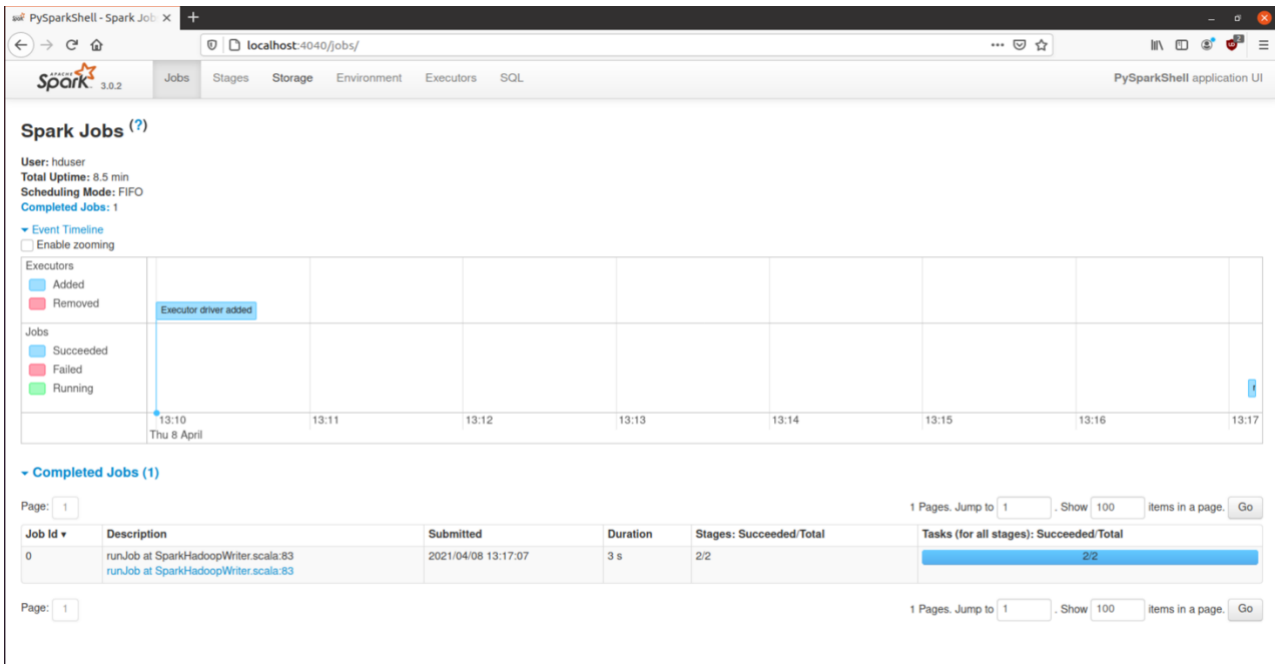


Εικόνα 3.2.3 5 Περιβάλλον χρήστη και διαχείρισης spark-shell (χωρίς να έχει εκτελεστεί κάποια εργασία)



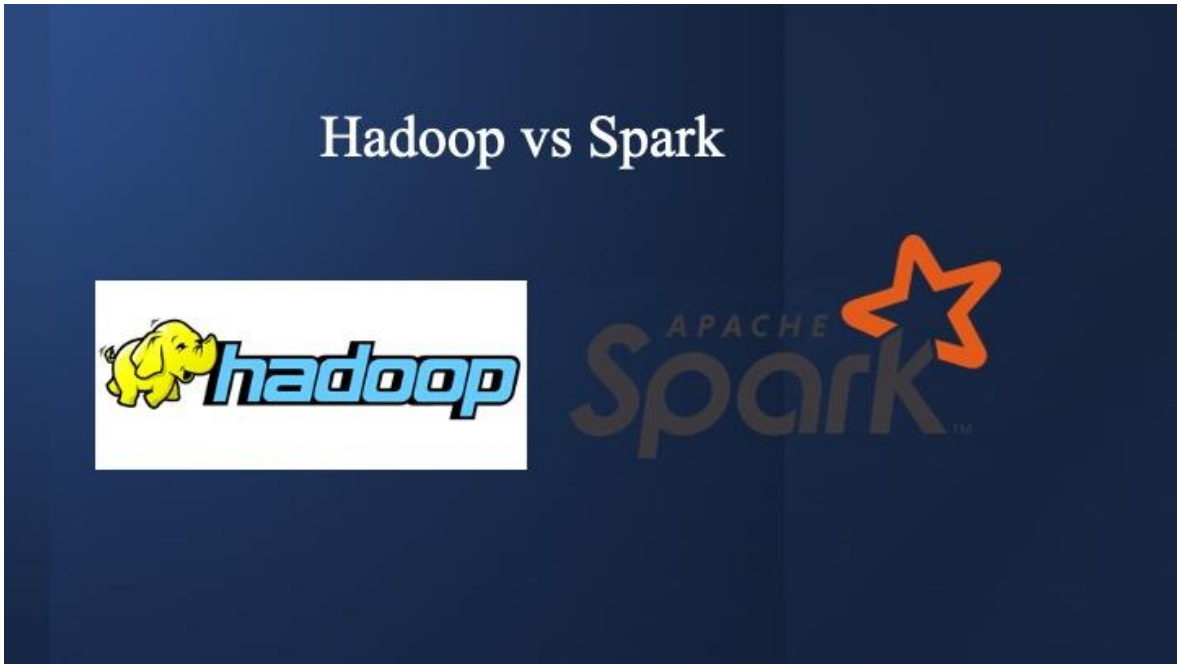
Εικόνα 3.2.3 6 Περιβάλλον χρήστη και διαχείρισης spark-shell (έχοντας εκτελεστεί μία εργασία)

Εικόνα 3.2.3 7 Περιβάλλον χρήστη και διαχείρισης pyspark (χωρίς να έχει εκτελεστεί κάποια εργασία)



Εικόνα 3.2.3 8 Περιβάλλον χρήστη και διαχείρισης pyspark (έχοντας εκτελεστεί μία εργασία)

4 ΣΥΓΚΡΙΣΗ HADOOP ΜΕ SPARK



Εικόνα 4 1 Hadoop VS Spark

Καθώς παραπάνω αναφέρθηκαν τα εργαλεία μοντελοποίησης δεδομένων Apache Hadoop και Apache Spark, τώρα ήρθε η ώρα να γίνει μία συγκριτική μελέτη μεταξύ τους. Η σύγκριση αυτή θα πραγματοποιηθεί με βάση τα κριτήρια της Εκτέλεσης, του Κόστους, της Ανοχής τους σε σφάλματα υλικού, της Επεξεργασίας Δεδομένων, της Ευκολίας στη Χρήση & Υποστήριξης Γλώσσας, της Επεκτασιμότητας, της Ασφάλειας, της Μηχανικής Εκμάθησης και τέλος του Προγραμματισμού και Διαχείρισης Πόρων [22].

- **Performance (Εκτέλεση):**

    - **Hadoop:** Το Hadoop παρόλη την πρόσβαση στα δεδομένα που είναι αποθηκευμένα τοπικά στο HDFS (αποτέλεσμα την ενίσχυση της συνολικής απόδοσης) είναι γενικά αργό, διότι εκτελεί εργασίες στο δίσκο. Συγκεκριμένα αποθηκεύει τα δεδομένα σε πολλές διαφορετικές τοποθεσίες και στη συνέχεια τα επεξεργάζεται σε παρτίδες (batches) χρησιμοποιώντας το MapReduce.

- **Spark:** Το Spark αποθηκεύει τα δεδομένα σε RAM και σύμφωνα με την Apache αυτό του προσφέρει τη δυνατότητα να είναι 100 φορές πιο γρήγορο από το Hadoop με το MapReduce. Σημαντική «νίκη» του Spark ήταν όταν κατάφερε να είναι 3 φορές πιο γρήγορο στην ταξινόμηση των δεδομένων σε δίσκους και χρειάστηκε 10 φορές λιγότερους κόμβους για την επεξεργασία 100 TB δεδομένων στο HDFS. Έτσι κατάφερε να δημιουργήσει παγκόσμιο ρεκόρ το 2014.

- **Cost (Κόστος):**

  - **Hadoop:** Το Hadoop, όπως και το Spark, είναι λογισμικά ανοιχτού κώδικα και είναι δωρεάν. Παράγοντας του κόστους οπότε είναι το κόστος υποδομής, συντήρησης και ανάπτυξης. Στο Hadoop λόγω της αποθήκευσης των δεδομένων στο δίσκο, το κόστος εκτέλεσης είναι σχετικά χαμηλό. Επιπρόσθετα, χρησιμοποιεί υπολογιστικά συστήματα χαμηλού κόστους

  - **Spark:** Το Spark όπως αναφέρθηκε και προηγουμένως είναι κι αυτό λογισμικό ανοιχτού κώδικα και να είναι δωρεάν. Βέβαια, εξαρτάται από υπολογισμούς στη μνήμη για την επεξεργασία δεδομένων σε πραγματικό χρόνο και έτσι η εκκίνηση κόμβων με μεγάλη ποσότητα μνήμης RAM αυξάνει σημαντικά το κόστος

- **Fault Tolerance (Ανοχή σε σφάλματα υλικού):**

  - **Hadoop:** Το Hadoop είναι ένα σύστημα όπου διαθέτει αρκετά μεγάλη ανοχή σε σφάλματα υλικού. Αναπαράγει τα δεδομένα στους κόμβους και σε περίπτωση κάποιου προβλήματος το σύστημα συνεχίζει την εργασία δημιουργώντας τα μπλοκ που λείπουν από άλλες τοποθεσίες. Ο κύριος κόμβος παρακολουθεί την κατάσταση των δευτερευόντων κόμβων και αν κάποιος δεν ανταποκρίνεται σε pings από έναν κύριο, ο κύριος κόμβος εκχωρεί τις εκκρεμείς εργασίες σε άλλον.

- **Spark:** Το Spark είναι κι αυτό ένα σύστημα όπου διαθέτει αρκετά μεγάλη ανοχή σε σφάλματα υλικού, αλλά ο τρόπος προσέγγισης σε αυτά είναι διαφορετικός. Συγκεκριμένα, χρησιμοποιεί μπλοκ RDD και παρακολουθεί τον τρόπο δημιουργίας του αμετάβλητου συνόλου δεδομένων. Έπειτα, επανεκκινεί τη διαδικασία εάν τυχόν υπάρξει κάποιο πρόβλημα. Επιπρόσθετα, μπορεί να αναδημιουργήσει δεδομένα σε ένα cluster χρησιμοποιώντας παρακολούθηση DAG (Directed Acyclic Graph) κατά την ροή εργασιών.

- **Data Processing (Επεξεργασία Δεδομένων):**

  - **Hadoop:** Το Hadoop για την επεξεργασία δεδομένων χρησιμοποιεί το MapReduce (το οποίο δεν απαιτεί μεγάλη ποσότητα μνήμης RAM για τη διαχείριση μεγάλου όγκου δεδομένων) με σκοπό να χωρίσει ένα μεγάλο σύνολο δεδομένων σε μία συστάδα για παράλληλη ανάλυση. Το Hadoop βασίζεται σε απλό υλικό για την αποθήκευση και κατά συνέπεια είναι πιο κατάλληλο για γραμμική επεξεργασία δεδομένων.

  - **Spark:** Το Spark χρησιμοποιεί τα RDDs, όπου το μέγεθός τους συνήθως είναι μεγάλο για να το χειριστεί ένας κόμβος και έτσι τα χωρίζει στους πλησιέστερους κόμβους ώστε να εκτελεί τις λειτουργίες παράλληλα. Επιπρόσθετα, παρέχει τη δυνατότητα της ανάλυσης δεδομένων σε πραγματικό χρόνο.

- **Ease of Use & Language Support (Ευκολία στη Χρήση & Υποστήριξη Γλώσσας):**

  - **Hadoop:** Το MapReduce του Hadoop δεν έχει διαδραστική λειτουργία και είναι περίπλοκο. Το framework του βασίζεται σε γλώσσα προγραμματισμού Java και η σύνταξη του κώδικα για MapReduce εργασίες πραγματοποιείται με Java ή Python.

- **Spark:** Το Spark είναι πιο φιλικό προς τον χρήστη διότι παρέχει υποστήριξη για API σε αρκετές γλώσσες προγραμματισμού. Η μητρική του γλώσσα είναι η Scala, αλλά υποστηρίζει και άλλες, όπως Java, Python, R και Spark SQL, με αποτέλεσμα ο χρήστης να διαλέγει όποια γλώσσα προγραμματισμού προτιμάει. Δίνει τη δυνατότητα των spark-shell και pyspark, τα λεγόμενα repl (Read Eval Print Loop), όπου εκεί μπορεί ο χρήστης να κάνει ανάλυση δεδομένων αλληλεπιδραστικά με Scala ή Python. Επιπρόσθετα, υπάρχει και το script, spark-submit όπου χρησιμοποιείται για την εκκίνηση εφαρμογών σε μία συστάδα υπολογιστών. Τέλος, δίνει τη δυνατότητα στον προγραμματιστή να επαναχρησιμοποιεί τον ήδη υπάρχοντα κώδικα και έτσι να κερδίζει σημαντικό χρόνο στην ανάπτυξη εφαρμογών.

- **Scalability (Επεκτασιμότητα):**

  - **Hadoop:** Το Hadoop λόγω του HDFS στην επεξεργασία μεγάλων δεδομένων, παρέχει μία εύκολη κλιμάκωση προσθέτοντας κόμβους και δίσκους για την αποθήκευση. Μπορεί και υποστηρίζει δεκάδες χιλιάδες κόμβους, χωρίς να είναι γνωστό κάποιο συγκεκριμένο όριο.

  - **Spark:** Επειδή το Spark βασίζεται στη μνήμη RAM για υπολογισμούς, είναι λιγότερο εύκολο να κλιμακωθεί. Μπορεί και υποστηρίζει χιλιάδες κόμβους σε ένα cluster και συγκεκριμένα μερικοί από τους επιβεβαιωμένους αριθμούς περιλαμβάνουν 8000 μηχανήματα με petabytes δεδομένων.

- **Security (Ασφάλεια):**

  - **Hadoop:** Το Hadoop είναι εξαιρετικά ασφαλές καθώς λειτουργεί με πολλαπλές μεθόδους ελέγχου ταυτότητας και ελέγχου πρόσβασης. Συγκεκριμένα, για τον έλεγχο ταυτότητας υποστηρίζει Kerberos και LDAP.

  - **Spark:** Το Spark δεν θεωρείται καθόλου ασφαλές καθώς η ασφάλεια είναι απενεργοποιημένη από προεπιλογή. Για να θεωρηθεί ασφαλές πρέπει να ενσωματώσει το Hadoop ώστε να μπορεί να χρησιμοποιήσει τις δικές του

μεθόδους.

- **Machine Learning (Μηχανική Εκμάθηση):**

    - **Hadoop:** Το Hadoop χρησιμοποιεί τη βιβλιοθήκη Mahout για επεξεργασία δεδομένων, ομαδοποίησης και ταξινόμησης. Επίσης, μία νεότερη είναι η Samsara, η οποία χρησιμοποιεί μία γλώσσα DSL με υποστήριξη Scala και μπορεί να χρησιμοποιηθεί για αλγεβρικές λειτουργίες στη μνήμη και επιτρέπει στους χρήστες να γράφουν δικό τους αλγόριθμο.

    - **Spark:** Το Spark χρησιμοποιεί μία ενσωματωμένη βιβλιοθήκη την MLlib, όπου του προσφέρει επαναληπτικούς υπολογισμούς στη μνήμη. Περιλαμβάνει εργαλεία για την εκτέλεση παλινδρόμησης, ταξινόμησης, αξιολόγησης και πολλά άλλα ακόμη. Το Spark με τη βιβλιοθήκη MLlib αποδείχτηκε αρκετά πιο γρήγορο από το Hadoop με τη βιβλιοθήκη Mahout.

- **Scheduling & Resource Management (Προγραμματισμός & Διαχείριση Πόρων):**

    - **Hadoop:** Το Hadoop δεν διαθέτει ενσωματωμένο προγραμματιστή, με αποτέλεσμα να χρησιμοποιεί εξωτερικές λύσεις για τον προγραμματισμό και τη διαχείριση πόρων. Το Oozie είναι ένα από τα διαθέσιμα εργαλεία για τον προγραμματισμό ροών εργασιών και το YARN για τη διαχείριση πόρων.

    - **Spark:** Το Spark διαθέτει ενσωματωμένα εργαλεία για κατανομή πόρων, προγραμματισμό και παρακολούθηση. Κύριο εργαλείο είναι το DAG, το οποίο είναι υπεύθυνο για τον διαχωρισμό των λειτουργιών σε στάδια.

5 ΕΦΑΡΜΟΓΕΣ

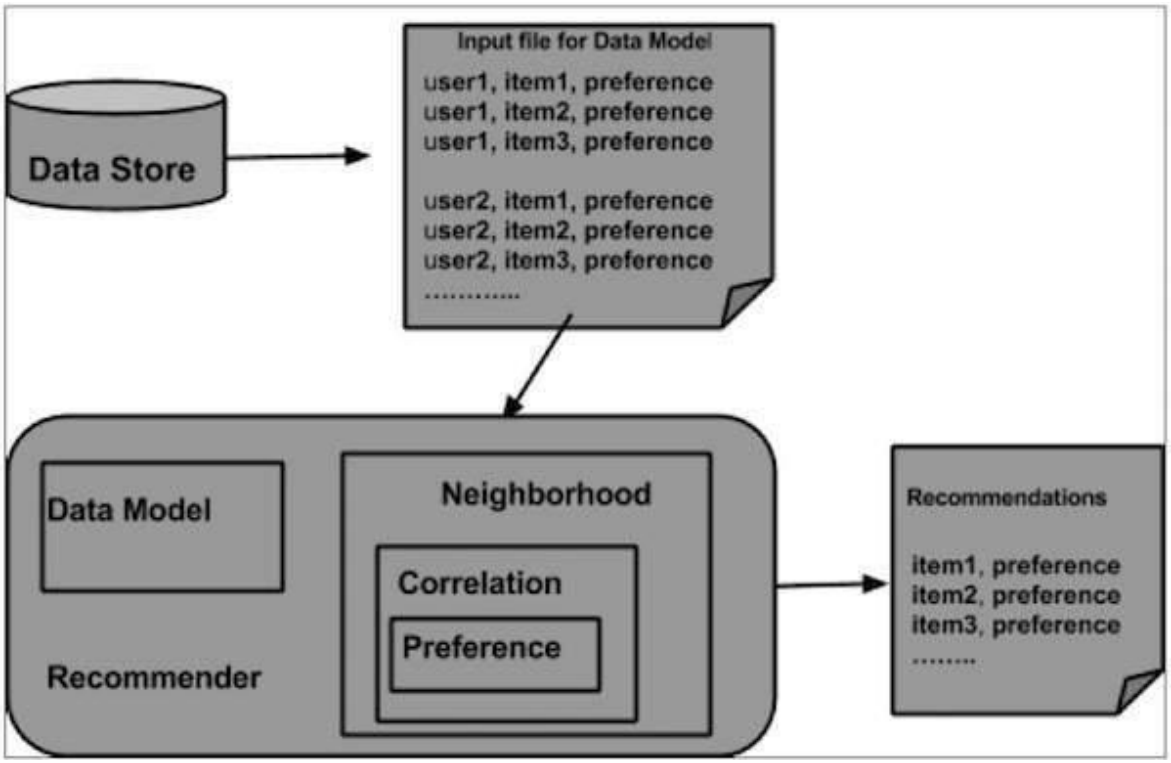
5.1 ΕΦΑΡΜΟΓΕΣ ΜΕ ΤΗ ΧΡΗΣΗ APACHE HADOOP

Κατά τη διάρκεια της πτυχιακής εργασίας, δημιουργήθηκαν για το Apache Hadoop δύο εφαρμογές. Ένα για συστάσεις ταινιών με τη χρήση του Apache Mahout και ένα για ανάλυση βαθμολογίας με τη χρήση Hadoop Streaming (Οι κώδικες των εφαρμογών υπάρχουν στο ΠΑΡΑΡΤΗΜΑ μετά τη Βιβλιογραφία).

5.1.1 ΕΦΑΡΜΟΓΗ ΜΕ ΤΗ ΧΡΗΣΗ APACHE MAHOUT

Το Apache Mahout είναι ένα εκ των υποσυστημάτων του οικοσυστήματος του Hadoop και είναι ένα έργο ανοιχτού κώδικα, όπου χρησιμοποιείται κυρίως για τη δημιουργία κλιμακούμενων αλγορίθμων μηχανικής εκμάθησης. Δημοφιλείς τεχνικές είναι η ομαδοποίηση, η ταξινόμηση και η σύσταση. Μπορεί και λειτουργεί σε κατανεμημένο περιβάλλον, διότι οι αλγόριθμοί του γράφονται στην κορυφή του Hadoop και προσφέρει στον χρήστη ένα έτοιμο πλαίσιο για την εκτέλεση εργασιών εξόρυξης δεδομένων σε δεδομένα με μεγάλο όγκο. Επιπρόσθετα, επιτρέπει στις εφαρμογές να πραγματοποιούν ανάλυση μεγάλων συνόλων δεδομένων αρκετά γρήγορα και αποτελεσματικά. Το Mahout χρησιμοποιείται από αρκετά γνωστές εταιρείες, όπως Adobe, Facebook, LinkedIn, Twitter και Yahoo. Συγκεκριμένα το Twitter χρησιμοποιεί το Mahout για μοντελοποίηση των ενδιαφερόντων του χρήστη και το Yahoo για την εξόρυξη προτύπων [23]. Ως προς την τεχνική των συστάσεων, έχουμε διάφορους μηχανισμούς, συστάσεις με βάσει τον χρήστη, το αντικείμενο και πολλά άλλα ακόμη. Ο τρόπος με τον οποίο λειτουργεί είναι σχετικά απλός, δίνεις ένα αρχείο κειμένου με προτιμήσεις διάφορων χρηστών για κάποια αντικείμενα και η έξοδος αυτού είναι οι εκτιμώμενες προτιμήσεις ενός συγκεκριμένου χρήστη για άλλα αντικείμενα. Για παράδειγμα, έχουμε μία πλατφόρμα με ταινίες και θέλουμε να εφαρμόσουμε τις δυνατότητες του Mahout, ώστε να δημιουργήσουμε μια μηχανή συστάσεων. Η μηχανή αυτή αναλύει προηγούμενες ταινίες που έχουν παρακολουθήσει οι χρήστες και τους προτείνει νέες βάσει με αυτές που έχουν δει. Η μηχανή αυτή αποτελείται από το Μοντέλο Δεδομένων (DataModel), την Ομοιότητα των χρηστών

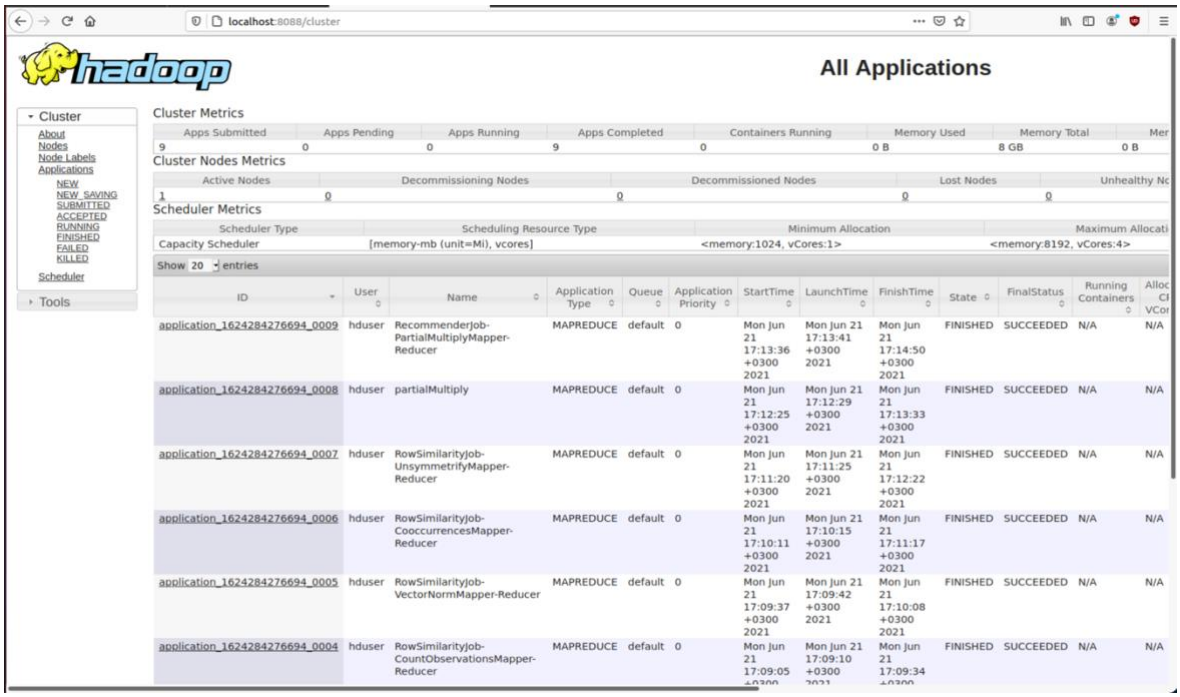
(UserSimilarity), την ομοιότητα των αντικειμένων (ItemSimilarity), τον γειτονικό χρήστη (UserNeighborhood) και τις συστάσεις (Recommender). Στην Εικόνα 5.1.1 1 φαίνεται η αρχιτεκτονική της μηχανής των συστάσεων [24].



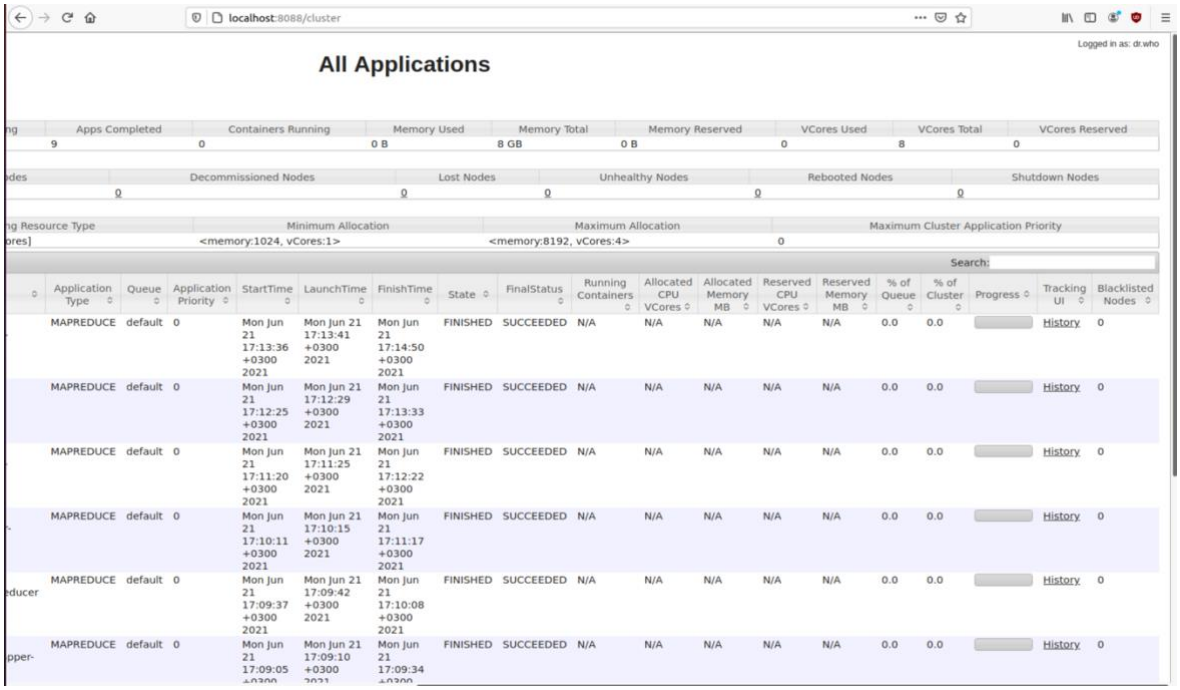
Εικόνα 5.1.1 1 Αρχιτεκτονική Μηχανής Συστάσεων (Πηγή: https://www.tutorialspoint.com/mahout/mahout_recommendation.htm)

Τώρα ως προς την τεχνική της ομαδοποίησης. Με την ομαδοποίηση πραγματοποιείται η οργάνωση των στοιχείων ή των αντικειμένων μιας συλλογής σε ομάδες, βάσει την ομοιότητα μεταξύ των αντικειμένων. Η ομαδοποίηση χρησιμοποιείται σε αρκετές εφαρμογές, όπως έρευνα αγοράς, αναγνώριση προτύπων και ανάλυση δεδομένων. Μπορεί να βοηθήσει τους εμπόρους να ανακαλύψουν ξεχωριστές ομάδες στη βάση των πελατών τους. Επιπρόσθετα, χρησιμοποιείται και σε εφαρμογές ανίχνευσης εξωτερικών παραμέτρων, όπως για παράδειγμα η ανίχνευση απάτης με πιστωτικές κάρτες. Οι αλγόριθμοι ομαδοποίησης που υποστηρίζει το Mahout είναι οι εξής, Canopy και K-means. Η ομαδοποίηση Canopy είναι μία γρήγορη τεχνική, όπου τα αντικείμενα αντιμετωπίζονται ως σημεία. Επίσης, χρησιμοποιείται και ως αρχικό βήμα σε άλλους αλγορίθμους όπως ο K-means. Ο K-means είναι ένας αλγόριθμος συστάδων και το K καθορίζει τον αριθμό των συστάδων όπου πρόκειται να διαιρεθούν τα δεδομένα [25]. Τέλος, η τεχνική της ταξινόμησης χρησιμοποιεί γνωστά δεδομένα για να καθορίσει πώς τα νέα δεδομένα πρέπει να ταξινομηθούν σε ένα σύνολο ήδη υπαρχουσών κατηγοριών. Για παράδειγμα, το Spotify

χρησιμοποιεί την ταξινόμηση για την προετοιμασία των λιστών αναπαραγωγής. Επίσης, πάροχοι υπηρεσιών αλληλογραφίας χρησιμοποιούν την τεχνική αυτή ώστε να αποφασίσουν εάν θα ταξινομηθεί η αλληλογραφία στα ανεπιθύμητα. Συγκεκριμένα, ο αλγόριθμος αναλύει τις συνήθειες των χρηστών στην επισήμανση των μηνυμάτων ως ανεπιθύμητα κι έτσι μετά μπορεί και αποφασίζει το που θα τα τοποθετήσει στο μέλλον αυτόματα, στα εισερχόμενα ή στα ανεπιθύμητα [26]. Τώρα ήρθε η ώρα να δείξουμε και την εφαρμογή που δημιουργήσαμε. Για την εφαρμογή μας έχουμε εγκαταστήσει το mahout στο Hadoop, έχουμε κατεβάσει τα δεδομένα που χρειαζόμαστε από το MovieLens και συγκεκριμένα το ml-100k.zip [27], όπου από το οποίο θα χρησιμοποιήσουμε το u.data, το u.user και το u.item. Το u.data αποτελείται από τις πλειάδες, user_id, movie_id, rating και timestamp, το u.user από τις πλειάδες user_id, age, gender, occupation και zip_code και το u.item από τις πλειάδες movie_id, title, release_date, video_release_data, imdb_url, cat_unknown, cat_action, cat_adventure, cat_animation, cat_children, cat_comedy, cat_crime, cat_documentary, cat_drama, cat_fantasy, cat_film_noir, cat_horror, cat_musical, cat_mystery, cat_romance, cat_sci_fi, cat_thriller, cat_war, cat_western. Για αρχή θα χρειαστεί με την εντολή **hadoop fs -put u.data u.data** να φορτώσουμε τα δεδομένα στο hdfs και ύστερα με την εντολή **hadoop jar mahout-mr-0.13.0-job.jar org.apache.mahout.cf.taste.hadoop.item.RecommenderJob -s SIMILARITY_COOCCURRENCE -input u.data –output output** να εκτελέσουμε το πρόγραμμα, όπου βρίσκεται μέσα στον φάκελο του mahout και συγκεκριμένα στο path org/apache/mahout/cf/taste/hadoop/item. Το όρισμα **-s SIMILARITY_COOCURRENCE** συμβολίζει τον τύπο προσομοίωσης των αντικειμένων που θα χρησιμοποιηθεί. Επιπρόσθετα, με το SIMILARITY COOCURRENCE δύο ταινίες θα είναι παρόμοιες εάν εμφανίζονται συχνά μαζί στις αξιολογήσεις του χρήστη. Για να βρούμε οπότε τις ταινίες όπου το σύστημα θα προτείνει στον χρήστη, πρέπει πρώτα να βρούμε 10 ταινίες παρόμοιες με αυτές που έχει αξιολογήσει. Το Mahout έπειτα θα υπολογίσει τις προτάσεις εκτελώντας μερικές MapReduce εργασίες, όπως φαίνεται στις Εικόνες 5.1.1 2 και 5.1.1 3.



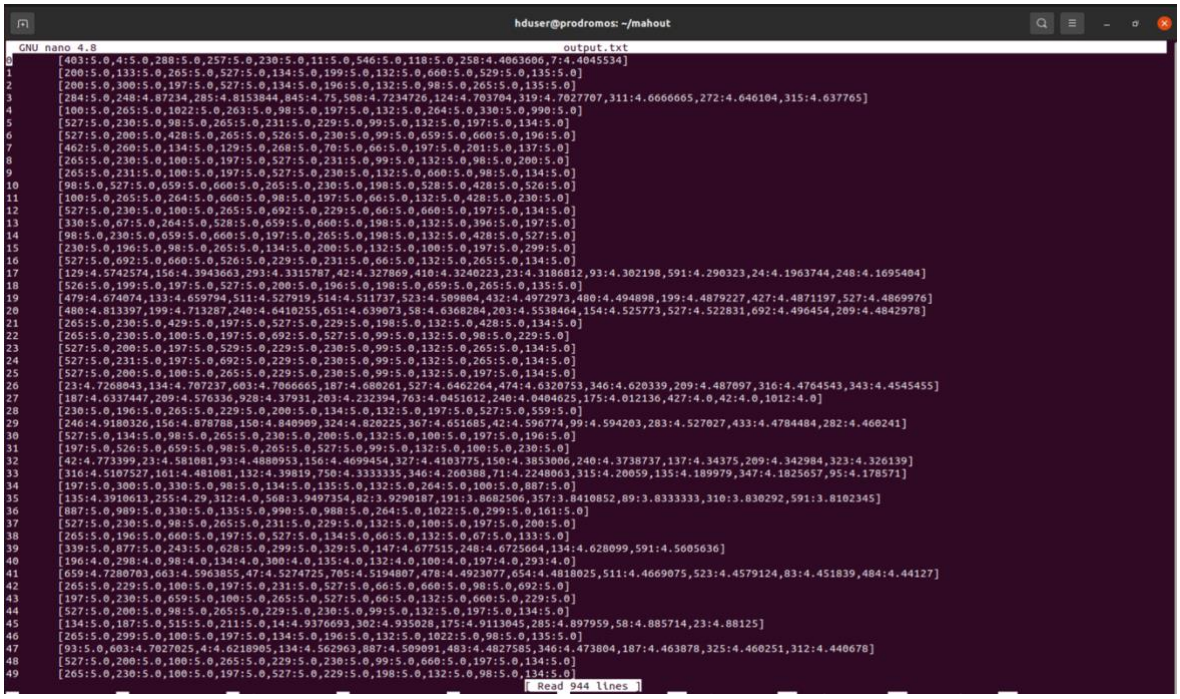
Εικόνα 5.1.1 2 Hadoop Jobs (1)



Εικόνα 5.1.1 3 Hadoop Jobs (2)

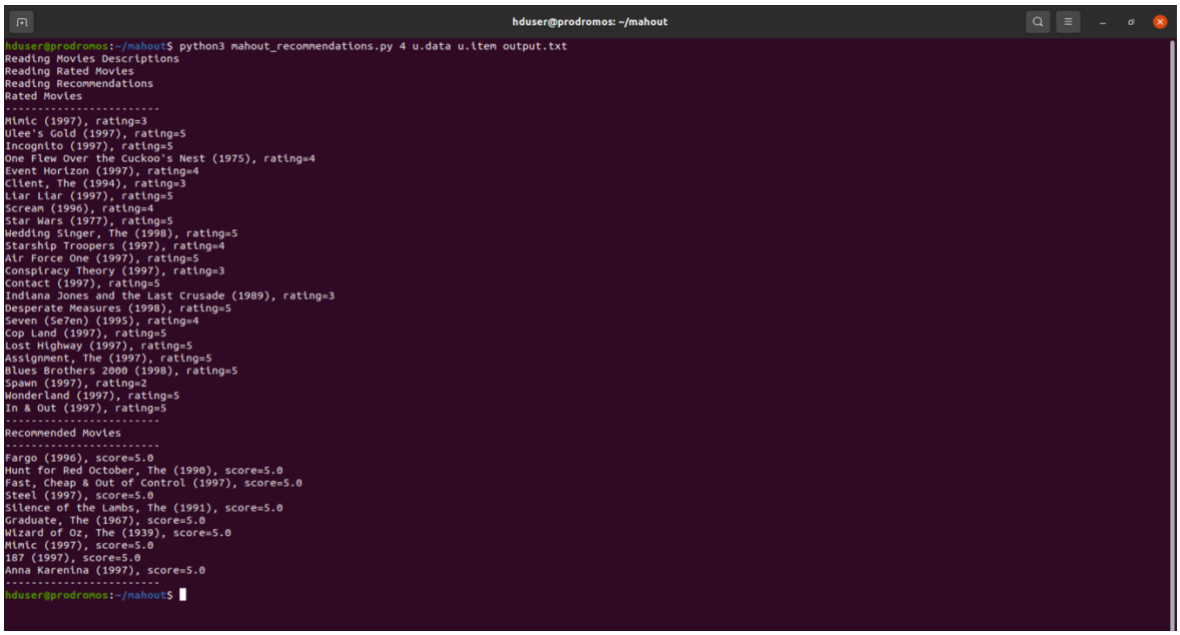
Μόλις ολοκληρωθούν αυτές οι εργασίες, κάθε χρήστης θα έχει τις 10 ταινίες που είναι πιθανό να του αρέσουν. Επόμενο βήμα είναι να αντιγράψουμε και να συγχωνεύσουμε τα αρχεία από το hdfs στο τοπικό σύστημα αρχείων. Αυτό επιτυγχάνεται με την εντολή **hadoop fs -getmerge output output.txt**. Το txt αρχείο θα είναι όπως στην Εικόνα 5.1.1 4. Κάθε γραμμή συμβολίζει τη σύσταση για τον κάθε χρήστη. Συγκεκριμένα, ο πρώτος αριθμός είναι

το user id και τα 10 ζεύγη αριθμών αντιπροσωπεύουν ένα movie id και ένα score. Για παράδειγμα, για τον χρήστη με id=4 οι 10 προτάσεις ταινιών είναι αυτές με id=100, 265, 1022, 263, 98, 197, 132, 264, 330, 990.



Εικόνα 5.1.1 4 output.txt

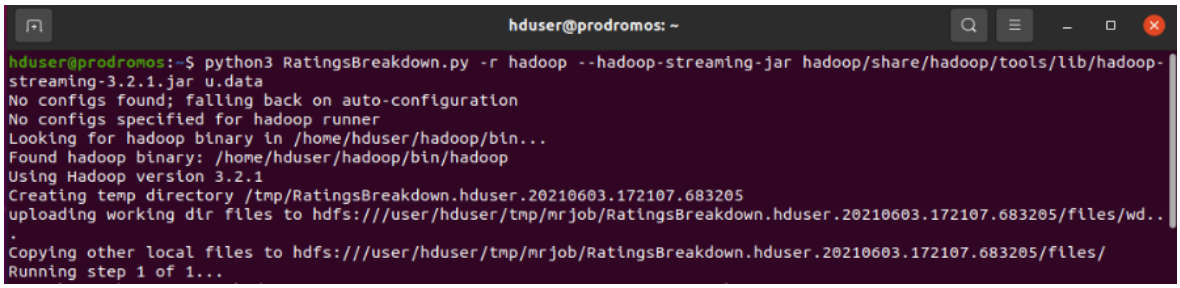
Για να δούμε όμως και ποιες είναι ακριβώς αυτές οι ταινίες, γράφτηκε και ένα πρόγραμμα σε γλώσσα προγραμματισμού python. Για το πρόγραμμα χρησιμοποιήθηκε το αρχείο u.data για τη λίστα των ταινιών, το αρχείο u.item για τη λίστα των τίτλων των ταινιών και το output.txt για τη λήψη της λίστας των προτεινόμενων ταινιών για τον χρήστη. Για την εκτέλεση του προγράμματος για τις συστάσεις στον χρήστη με id=4, δώσαμε την εντολή **python3 mahout_recommendations.py 4 u.data u.item output.txt**. Το αποτέλεσμα φαίνεται στην Εικόνα 5.1.1 5.



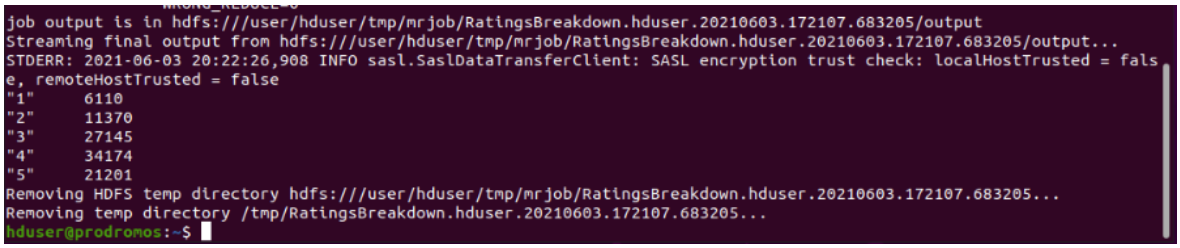
Εικόνα 5.1.1 5 Αποτέλεσμα συστάσεων για τον χρήστη με id=4

5.1.2 ΕΦΑΡΜΟΓΗ ΜΕ ΤΗ ΧΡΗΣΗ HADOOP STREAMING

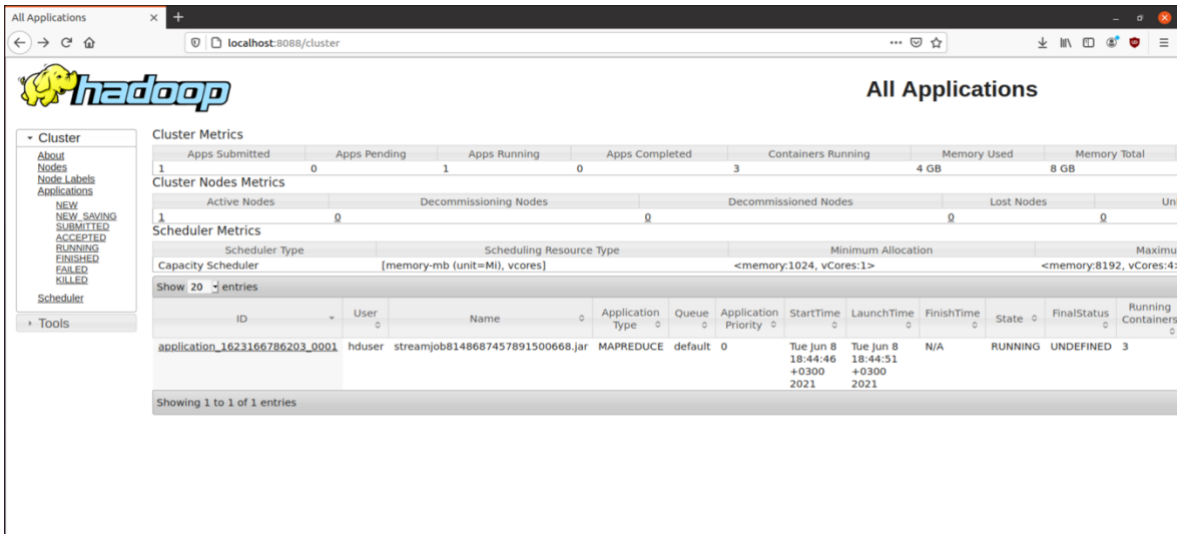
Το Hadoop Streaming είναι ένα βοηθητικό πρόγραμμα όπου υπάρχει μέσα στο Hadoop και επιτρέπει τη δημιουργία και την εκτέλεση Map Reduce εργασιών με οποιαδήποτε εκτελέσιμο ή script ως mapper ή και reducer. Συγκεκριμένα βρίσκεται μέσα στον φάκελο hadoop/share/hadoop/tools/lib. Στο παράδειγμά μας χρησιμοποιήσαμε 1 σύνολο δεδομένων από το MovieLens (ml-100k.zip) [27], το u.data, όπου το αποθηκεύσαμε μέσα στο hdfs και αποτελείται από τέσσερις πλειάδες, USER ID, MOVIE ID, RATING και TIMESTAMP. Ο κώδικας γράφτηκε σε γλώσσα προγραμματισμού python και επιστρέφει τις βαθμολογίες των χρηστών για τις ταινίες και τον αριθμό όπου πραγματοποιήθηκαν οι αξιολογήσεις. Επιπρόσθετα χρησιμοποιήθηκε και το mrjob, το οποίο είναι ένα πακέτο της python και βοηθά να γράφονται και να εκτελούνται εργασίες χρησιμοποιώντας Hadoop Streaming. Παρακάτω στις Εικόνες 5.1.2 1 και 5.1.2 2 φαίνεται η εντολή που δόθηκε για να εκτελεστεί το πρόγραμμα και το αποτέλεσμά του.



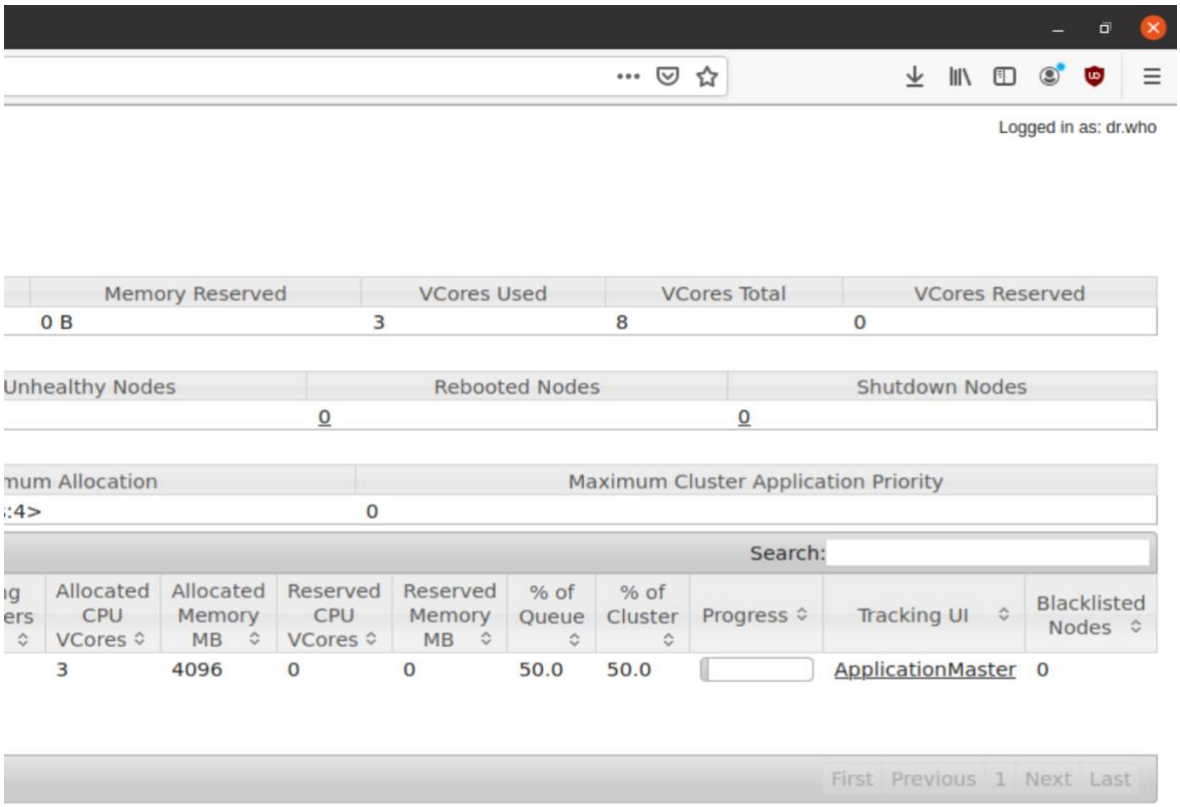
Εικόνα 5.1.2 1 Εκτέλεση RatingsBreakdown.py



Εικόνα 5.1.2 2 Βαθμολογίες & οι φορές όπου πραγματοποιήθηκαν οι αξιολογήσεις



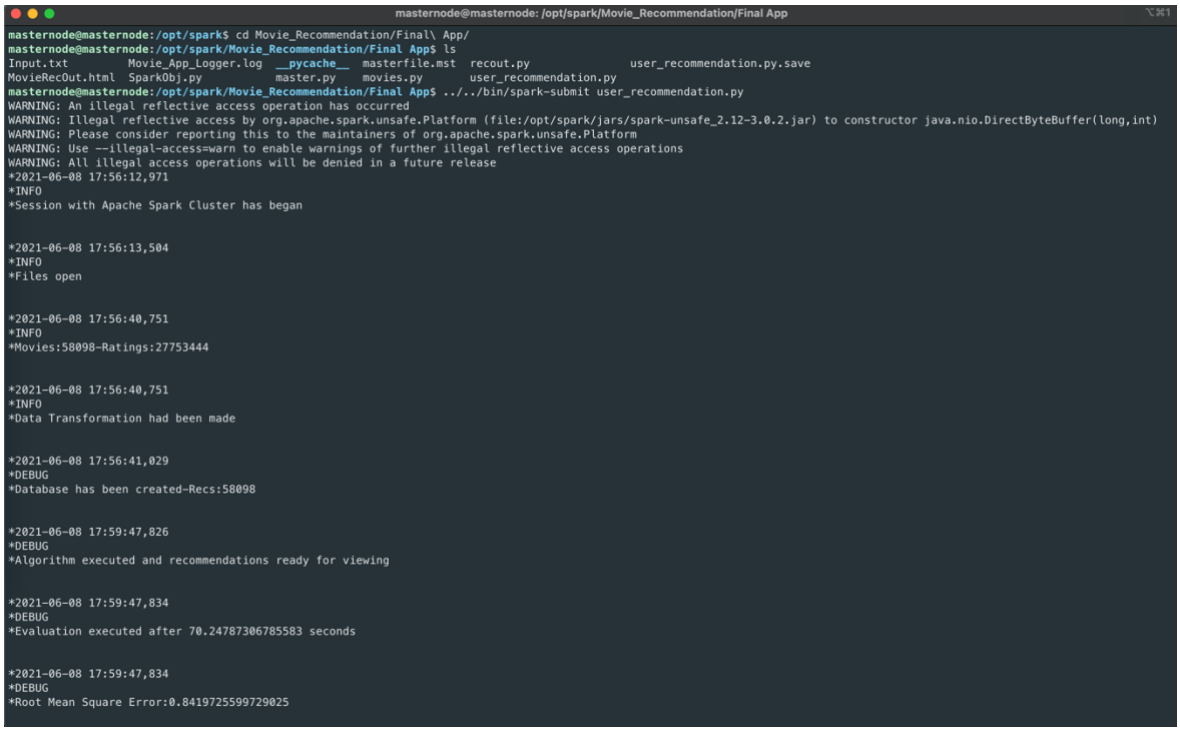
Εικόνα 5.1.2 3 Hadoop All Applications UI (έχοντας εκτελεστεί το παράδειγμα) [1η]



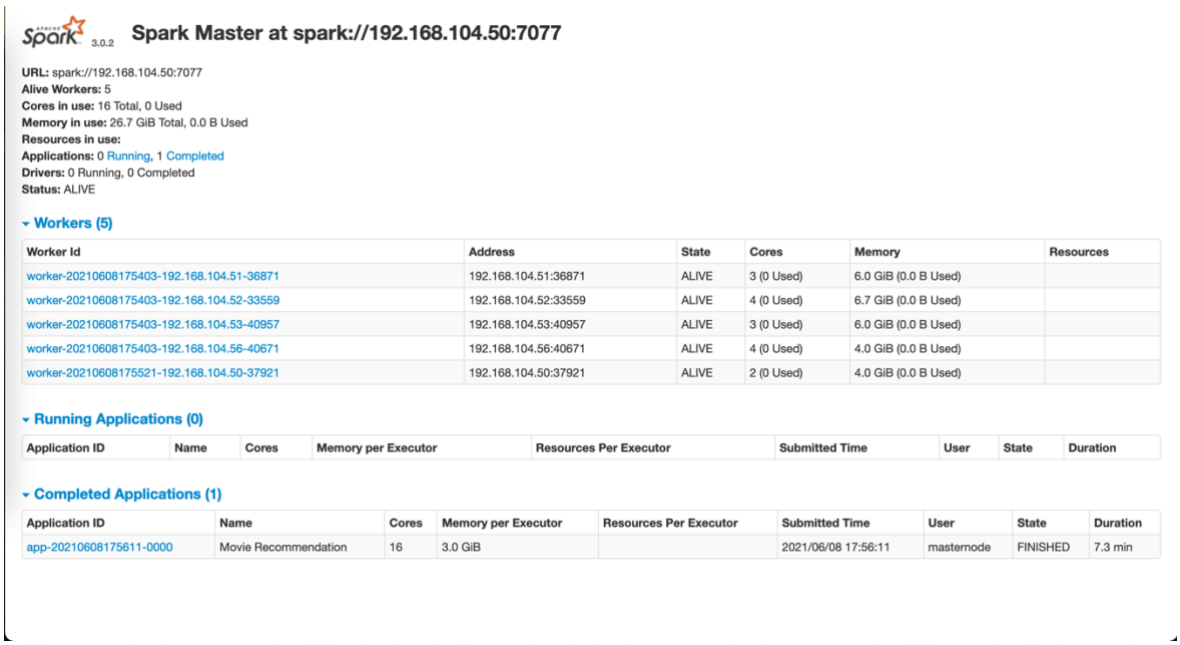
Εικόνα 5.1.2 4 Hadoop All Applications UI (έχοντας εκτελεστεί το παράδειγμα) [2η]

5.2 ΕΦΑΡΜΟΓΗ ΜΕ ΤΗ ΧΡΗΣΗ APACHE SPARK

Κατά τη διάρκεια της πτυχιακής εργασίας, δημιουργήθηκε για το Apache Spark μία εφαρμογή σύστασης ταινιών με τη χρήση της μεθόδου του Collaborative Filtering. Το Collaborative Filtering βασίζεται στη βιβλιοθήκη ALS (Alternating Least Squares) και είναι ένας αλγόριθμος που χρησιμοποιείται για τη δημιουργία συστάσεων. Συγκεκριμένα, συγκεντρώνει προβλέψεις που σχετίζονται με τα ενδιαφέροντα ενός χρήστη, συλλέγοντας παράλληλα προτιμήσεις από πολλούς άλλους χρήστες. Έχει εφαρμοστεί επίσης στο Spark MLlib, ώστε να εκτελεί μεγάλα σύνολα δεδομένων γρήγορα [28]. Στη δικιά μας περίπτωση, χρησιμοποιήσαμε 2 σύνολα δεδομένων από το MovieLens (ml-latest.zip) [29], το movies.csv και το ratings.csv, όπου μέσα έχουν διάφορες ταινίες και κριτικές και διαλέξαμε έναν χρήστη, όπου με βάσει κάποιες ταινίες που έχει δει και κριτικές που έχει κάνει, το σύστημά μας να μπορεί να του προτείνει 10 νέες ταινίες. Η υλοποίηση έγινε με το pyspark και δημιουργήθηκε στατικά με html και javascript μία σελίδα για την εμφάνιση των αποτελεσμάτων. Όπως φαίνεται και στην Εικόνα 5.2 1, για την εκτέλεση του προγράμματος χρειάστηκε να μεταβούμε στον φάκελο όπου αποθηκεύσαμε το πρόγραμμα μέσα στο spark και αυτό έγινε με την εντολή **“ cd ../../opt/spark/Movie_Recommendation/’Final App’/ ”** και έπειτα δώσαμε την εντολή **“ ../../bin/spark-submit user_recommendation.py ”** ώστε να υποβάλλουμε το πρόγραμμα. Όπως φαίνεται από την εντολή εκτέλεσης του προγράμματος, η υποβολή στο cluster έγινε με το script spark-submit.



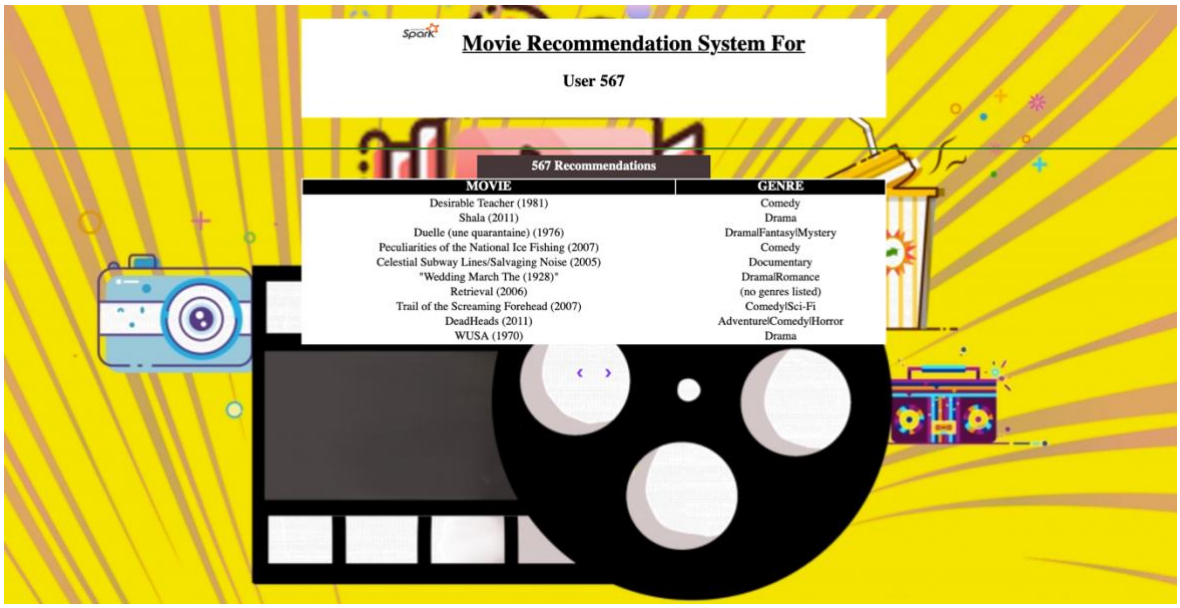
Εικόνα 5.2 1 Εκτέλεση προγράμματος



Εικόνα 5.2 2 Spark UI (έχοντας εκτελεστεί η εφαρμογή)

Εικόνα 5.2 3 Ταινίες που έχει δει ο χρήστης με ID=567 (i)

Εικόνα 5.2 4 Ταινίες που έχει δει ο χρήστης με ID=567 (ii)



Εικόνα 5.2 5 Ταινίες που προτείνει το σύστημα στον χρήστη με ID=567

6 ΣΥΜΠΕΡΑΣΜΑΤΑ

Τα συμπεράσματα που προέκυψαν κατά την εκπόνηση της εργασίας είναι τα εξής. Αρχικά, ως προς τις εγκαταστάσεις των δύο αυτών εργαλείων παρατηρούμε πως η εγκατάσταση του Apache Hadoop παρουσιάζει μεγαλύτερο βαθμό πολυπλοκότητας από την αντίστοιχη εγκατάσταση του Apache Spark. Το Hadoop έχει αρκετές παραμετροποιήσεις όπου πρέπει να δημιουργηθούν, έχει το core-site.xml, το hdfs-site.xml, το mapred-site.xml και το yarn-site.xml. Επίσης, χρειάζεται να δημιουργηθούν και κάποιοι φάκελοι (tmpdata, dfsdata/namenode, dfsdata/datanode), αλλιώς θα υπάρξει σφάλμα κατά την εκκίνηση των κόμβων. Αντίθετα στο Spark, μετά τους ορισμούς όπου πρέπει να γίνουν στο profile, ο χρήστης είναι έτοιμος να ξεκινήσει τους κόμβους χωρίς να χρειάζεται να κάνει κάτι άλλο. Παρόλα αυτά, υπάρχει και η λύση από τις εταιρείες MapR και Cloudera σε συνεργασία με τη Hortonworks, δίνοντας έτοιμο το περιβάλλον των εργαλείων σε virtual machines. Και τα δύο εργαλεία είναι πολύ χρήσιμα ως προς την επεξεργασία Μεγάλων Δεδομένων. Συνδυάζοντάς τα ειδικά, πιστεύω είναι ο καλύτερος τρόπος διαχείρισής τους. Όπου υστερεί το ένα το άλλο εργαλείο έρχεται και το συμπληρώνει. Το Hadoop εκτελεί τις εργασίες στον δίσκο με αποτέλεσμα να είναι σχετικά αργό, σε αντίθεση με το Spark όπου αποθηκεύει τα δεδομένα σε RAM. Το Hadoop χρησιμοποιεί υπολογιστικά συστήματα χαμηλού κόστους, ενώ το Spark εξαρτάται από υπολογισμούς στη μνήμη για την επεξεργασία δεδομένων σε πραγματικό χρόνο, με αποτέλεσμα η εκκίνηση των κόμβων με μεγάλη ποσότητα μνήμης RAM να αυξάνει σημαντικά το κόστος. Ως προς την ανοχή σε σφάλματα υλικού, το Hadoop αναπαράγει τα δεδομένα σε κόμβους και σε περίπτωση κάποιου προβλήματος το σύστημα συνεχίζει την εργασία δημιουργώντας τα μπλοκ που λείπουν από άλλες τοποθεσίες. Το Spark αντίθετα μπορεί να αναδημιουργήσει δεδομένα σε ένα cluster χρησιμοποιώντας παρακολούθηση DAG. Το Hadoop είναι αρκετά ασφαλές, σε αντίθεση με το Spark που έχει από προεπιλογή την ασφάλεια κλειστή. Τέλος, το Hadoop παρέχει μία εύκολη κλιμάκωση προσθέτοντας κόμβους και δίσκους για την αποθήκευση, ενώ το Spark είναι λιγότερο εύκολο να κλιμακωθεί, αφού βασίζεται στη μνήμη RAM για υπολογισμούς. Επιπρόσθετα, ένα από τα πλεονεκτήματα τους είναι ότι μπορούν να εκτελεστούν από υπολογιστές με hardware το οποίο δεν πρέπει να είναι απαραίτητα υψηλών προδιαγραφών. Μπορούν εύκολα να εκτελεστούν και μέσα από virtual machines. Στην περίπτωση μας πραγματοποιήθηκαν αρκετά πειράματα τοπικά σε VM. Εν κατακλείδι, μπορεί κάποιος να κάνει αρκετά πράγματα με αυτά τα εργαλεία, όπως η δημιουργία εφαρμογών συστάσεων

ταινιών (όπως δείξαμε κι εμείς), εφαρμογές ανίχνευσης κάποιας ηλεκτρονικής απάτης και γενικά οτιδήποτε έχει να κάνει με Μεγάλα Δεδομένα, δεν είναι τυχαίο εξάλλου που χρησιμοποιούνται και από τόσο μεγάλες εταιρείες. Όποιος θέλει να ασχοληθεί με τον κλάδο των Big Data, θα ήταν πολύ ενδιαφέρον και αρκετά χρήσιμο για εκείνον να ασχοληθεί και με το Hadoop και το Spark.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] D. Gewirtz, “Volume, velocity, and variety: Understanding the three V’s of big data.” 2018, [Online]. Available: https://www.zdnet.com/article/volume-velocity-and-variety-understanding-the-three-vs-of-big-data/.

[2] C. P. Kasula, “Netflix Recommender System — A Big Data Case Study.” 2020, [Online]. Available: https://towardsdatascience.com/netflix-recommender-system-a-big-data-case-study-19cfa6d56ff5.

[3] S. Gutta, “Data Science: The 5 V’s of Big Data.” 2020, [Online]. Available: https://suryagutta.medium.com/the-5-vs-of-big-data-2758bfcc51d.

[4] R. Kiran, “Top Big Data Technologies that you Need to know.” 2020, [Online]. Available: https://www.edureka.co/blog/top-big-data-technologies/.

[5] Apache Software Foundation, “Apache Hadoop.” 2021, [Online]. Available: http://hadoop.apache.org/.

[6] H. V. Karambelkar, *Apache Hadoop 3 Quick Start Guide*. Packt Publishing, 2018.

[7] Apache Software Foundation, “HDFS Architecture.” 2021, [Online]. Available: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.

[8] Apache Software Foundation, “Apache Hadoop YARN.” 2021, [Online]. Available: https://hadoop.apache.org/docs/r2.7.1/hadoop-yarn/hadoop-yarn-site/YARN.html.

[9] Apache Software Foundation, “MapReduce Tutorial.” 2021, [Online]. Available: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

[10] D. Kvasnicka and G. Roda, “BIG DATA ON VSC,” p. 63, 2021.

[11] R. Kiran, “MapReduce Tutorial – Fundamentals of MapReduce with MapReduce Example.” 2020, [Online]. Available: https://www.edureka.co/blog/mapreduce-tutorial/.

[12] S. Sinha, “Hadoop Ecosystem: Hadoop Tools for Crunching Big Data.” 2020, [Online]. Available: https://www.edureka.co/blog/hadoop-ecosystem.

[13] Databricks, “Apache Spark.” 2021, [Online]. Available: https://databricks.com/glossary/what-is-apache-spark.

[14] S. Penchikala, *BIG-DATA PROCESSING WITH APACHE SPARK*. C4Media, 2018.

[15] DataFlair, “Apache Spark SQL Tutorial – Quick Introduction Guide.” 2021, [Online]. Available: https://data-flair.training/blogs/apache-spark-sql/.

[16] DataFlair, “Spark Streaming Tutorial for Beginners.” 2021, [Online]. Available: https://data-flair.training/blogs/apache-spark-streaming-tutorial/.

[17] Apache Spark, “Machine Learning Library (MLlib) Guide.” 2021, [Online]. Available: https://spark.apache.org/docs/latest/ml-guide.html.

[18] DataFlair, “Spark MLlib Data Types | Apache Spark Machine Learning.” 2021, [Online]. Available: https://data-flair.training/blogs/spark-mllib-data-types/.

[19] Databricks, “Apache Spark Ecosystem.” 2021, [Online]. Available: https://databricks.com/spark/about.

[20] DataFlair, “GraphX API in Apache Spark: An Introductory Guide.” 2021, [Online]. Available: https://data-flair.training/blogs/graphx-api-spark/.

[21] Apache Spark, “Graphx Programming Guide.” 2021, [Online]. Available: https://spark.apache.org/docs/latest/graphx-programming-guide.html.

[22] phoenixNAP, “Hadoop vs Spark - Detailed Comparison.” 2020, [Online]. Available: https://phoenixnap.com/kb/hadoop-vs-spark.

[23] tutorialspoint, “Mahout - Quick Guide.” 2021, [Online]. Available:

https://www.tutorialspoint.com/mahout/mahout_quick_guide.htm.
[24] tutorialspoint, “Mahout - Recommendation.” 2021, [Online]. Available: https://www.tutorialspoint.com/mahout/mahout_recommendation.htm.
[25] tutorialspoint, “Mahout - Clustering.” 2021, [Online]. Available: https://www.tutorialspoint.com/mahout/mahout_clustering.htm.
[26] tutorialspoint, “Mahout - Classification.” 2021, [Online]. Available: https://www.tutorialspoint.com/mahout/mahout_classification.htm.
[27] grouplens, “MovieLens 100K Dataset.” 2016, [Online]. Available: https://grouplens.org/datasets/movielens/100k/.
[28] S. Rosaria, “Movie Recommendations with Spark Collaborative Filtering.” 2018, [Online]. Available: https://www.datanami.com/2018/11/02/movie-recommendations-with-spark-collaborative-filtering/.
[29] grouplens, “MovieLens Latest Datasets.” 2018, [Online]. Available: https://grouplens.org/datasets/movielens/latest/.

ΠΑΡΑΡΤΗΜΑ

Α. ΚΩΔΙΚΑΣ APACHE HADOOP

Α.Α ΚΩΔΙΚΑΣ ΕΦΑΡΜΟΓΗΣ ΣΥΣΤΑΣΕΩΝ ΜΕ ΤΗ ΧΡΗΣΗ APACHE MAHOUT

- Mahout_recommendations.py

```
import sys

# Έλεγχος του αριθμού των ορισμάτων τα οποία έδωσε ο χρήστης κατά την εκτέλεση της εντολής καταχώρησης της εργασίας στο cluster
if len(sys.argv) != 5:
        print("Arguments: userId userDataFilename movieFilename recommendationFilename")
        sys.exit(1)

 # Εκχώρηση δεδομένων σε 4 μεταβλητές
userId, userDataFilename, movieFilename, recommendationFilename = sys.argv[1:]

print("Reading Movies Descriptions")
# Άνοιγμα αρχείου
movieFile = open(movieFilename)
# Δημιουργία dictionary
movieById = {}
# Προσπέλαση του movieFile
for line in movieFile:
        # Αποθήκευση πληροφοριών ανά ταινία
        tokens = line.split("|")
        movieById[tokens[0]] = tokens[1:]
movieFile.close()

print("Reading Rated Movies")
# Άνοιγμα αρχείου
userDataFile = open(userDataFilename)
# Δημιουργία λίστας
ratedMovieIds = []
for line in userDataFile:
```

```
        # Split τα δεδομένα με βάσει το κενό
        tokens = line.split("\t")
        if tokens[0] == userId:
                ratedMovieIds.append((tokens[1],tokens[2]))
userDataFile.close()

print("Reading Recommendations")
# Βρίσκει τις αξιολογήσεις ανά ταινία
recommendationFile = open(recommendationFilename)
recommendations = []
for line in recommendationFile:
        tokens = line.split("\t")
        if tokens[0] == userId:
                movieIdAndScores = tokens[1].strip("[]\n").split(",")
                recommendations = [ movieIdAndScore.split(":") for movieIdAndScore in movieIdAndScores ]
                break
recommendationFile.close()

print("Rated Movies")
print("------------------------")
for movieId, rating in ratedMovieIds:
        print("%s, rating=%s" % (movieById[movieId][0], rating))
print("------------------------")
 # Έναρξη προτάσεων ανά ταινία
print("Recommended Movies")
print("------------------------")
for movieId, score in recommendations:
        print("%s, score=%s" % (movieById[movieId][0], score))
print("------------------------")
```

Α.Β ΚΩΔΙΚΑΣ ΠΑΡΑΔΕΙΓΜΑΤΟΣ ΑΝΑΛΥΣΗΣ ΒΑΘΜΟΛΟΓΙΑΣ ΜΕ ΤΗ ΧΡΗΣΗ HADOOP STREAMING

- RatingsBreakdown.py

```
from mrjob.job import MRJob
from mrjob.step import MRStep

class RatingsBreakdown(MRJob):
   # Define single mapreduce phase
   def steps(self):
```

```
        return [
            MRStep(mapper=self.mapper_get_ratings,
                   reducer=self.reducer_count_ratings)
        ]

    # Define mapper to break up lines of data on tab, return key value pair of rating and 1
    def mapper_get_ratings(self, _, line):
        (userID, movieID, rating, timestamp) = line.split('\t')
        yield rating, 1

    # Reduce key values, return rating and rating count
    def reducer_count_ratings(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    RatingsBreakdown.run()
```

Β. ΚΩΔΙΚΑΣ APACHE SPARK

Β.Α ΚΩΔΙΚΑΣ ΕΦΑΡΜΟΓΗΣ ΣΥΣΤΑΣΕΩΝ ΤΑΙΝΙΩΝ ΜΕ ΤΗ ΧΡΗΣΗ COLLABORATIVE FILTERING, ALS ΚΑΙ MLLIB

- user_recommendation.py (Movie Recommendation System)

```
import pyspark as ps
from pyspark.sql import SparkSession,DataFrame
import pyspark.sql as sql
from pyspark import SparkConf,SparkContext,RDD
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import RegressionEvaluator
import master as ms
import os
import sys
from time import time
import webbrowser
import logging as log
import SparkObj as so
import recout as rc

#Create a logger
logger=log.getLogger('Movie Recommendation Logger')
logger.setLevel(log.DEBUG)
fh=log.FileHandler('Movie_App_Logger.log','w')
sh=log.StreamHandler()
formatter=log.Formatter('*%(asctime)s\n*%(levelname)s\n*%(message)s\n\n')
fh.setFormatter(formatter)
sh.setFormatter(formatter)
logger.addHandler(fh)
logger.addHandler(sh)

#Handler functions Συναρτήσεις για επεξεργασία δεδομένων
# Επιστρέφει λίστα από τις λέξεις από τις οποίες έχουν προκύψει
# έπειτα από διαχωρισμό ενός αλφαριθμητικού με βάσει το κόμμα (,)
def SplitCsv(s):
    return [x for x in s.split(',')]

def CreateTuple(m): # Μετατρτοπή της λίστας σε πλειάδα
```

```
        return (m[0],''.join(m[1:len(m)-1]),m[len(m)-1])

def RatingTuple(s): # Κατασκευή υποπλειάδας με βάσει μια πλειάδα
    return (s[0],s[1],s[2])
    #Throw timestamp

def DbModel(elem): # Μετατροπή πλειάδας σε αντίστοιχη πλειάδα διαφορετικού μήκους
    return (elem[0],[elem[1],elem[2]])
def RemoveFiles():
    pages=os.path.join('','Static_Pages')
    #os.removedirs(pages)
    for x in os.listdir(pages):
        os.remove(os.path.join(pages,x))

#-------------------------------------------------------------------------------------

#Spark Session Initiate and Configure Spark Parameters
# Παραμετροποίηση πόρων εφαρμογής
conf=SparkConf().setAppName('Movie
Recommendation').setMaster(ms.master().Master()).set('spark.executor.memory','3G').set('spark.ui.showCons
oleProcess',True)
# Συνεδρία με cluster.
# Δημιουργεί ή αν υπάρχει συνεδρία με το cluster εκχωρεί την εφαρμογή στην υπάρχουσα συνεδρία
session=SparkSession.builder.config(conf=conf).getOrCreate()
sc=session.sparkContext
logger.info('Session with Apache Spark Cluster has began')

#Open Datasets and transform them
moviespath=os.path.join('..','ml-latest','movies.csv')
ratingspath=os.path.join('..','ml-latest','ratings.csv')

#In order to create an instict DataFrame
#movieFrame=sc.read.csv(moviespath)
#EratingsFrame=sc.read.csv(ratingspath)
#Alternative-->txt,json

#Open File Data
movies_raw_data=sc.textFile(moviespath) # Κατασκευή RDD
ratings_raw_data=sc.textFile(ratingspath) # Κατασκευή RDD
logger.info('Files open')
```

```
mheaderstemplate=movies_raw_data.take(1)[0] # Παίρνει το 1ο στοιχείο του RDD (επικεφαλίδες)
rheaderstemplate=ratings_raw_data.take(1)[0] # Παίρνει το 1ο στοιχείο του RDD (επικεφαλίδες)
# filtering των δεδομένων. Απαλοιφή πρώτης στήλης
movies=movies_raw_data.filter(lambda x:x!=mheaderstemplate).map(lambda x:SplitCsv(x)).map(lambda
y:CreateTuple(y))
# filtering των δεδομένων. Απαλοιφή πρώτης στήλης
ratings=ratings_raw_data.filter(lambda x:x!=rheaderstemplate).map(lambda x:SplitCsv(x)).map(lambda
y:RatingTuple(y))
mheaders=mheaderstemplate.split(',') # # Βρίσκουμε τις επακεφαλίδες του αρχείου movies.csv (movieId, name,
genre)
rheaders=rheaderstemplate.split(',') # Βρίσκουμε τις επακεφαλίδες του αρχείου ratings.csv (userId, movieId,
rating, timestamp)
rheaders.remove('timestamp') # Αφαίρεση στήλης timespamp (userId, movieId, rating)
logger.info(f'Movies:{movies.count()}-Ratings:{ratings.count()}')
logger.info('Data Transformation had been made')
#create a PseudoDatabase for finding a movie name
moviedb=movies.map(lambda k:(k[0],[k[1],k[2]])).collectAsMap() #moviedb['500'][0]-->NM [1]-->genre
logger.debug(f'Database has been created-Recs:{len(moviedb)}')

#create the apropriate DataFrames
movieFrame=movies.toDF(schema=mheaders) # Μετατρέπουμε το rdd σε dataframe
# Επεξεργασία των υπάρχοντων δεδομένων του dataset χωρίς να αλλάζει τον συνολικό αριθμό των δεδομένων
ratingsFrame=ratings.map(lambda elem:(int(elem[0]),int(elem[1]),float(elem[2]))).toDF(schema=rheaders)

#Split Data into Training and Test Set
train,test=ratingsFrame.randomSplit([0.7,0.3])

#Start working with the Data and execute alternating least square algorithm
als=ALS() # Κατασκευή αντικειμένου τύπου als
# Ορισμός παραμέτρων
als.setMaxIter(7).setRegParam(0.01).setUserCol(rheaders[0]).setItemCol(rheaders[1]).setRatingCol(rheaders[
2]).setColdStartStrategy('drop').setImplicitPrefs(False)
model=als.fit(train) # Εκπαίδευση αλγορίθμου
prediction=model.transform(test) # Παραγωγή προβλέψεων
# Αξιολόγηση το μοντέλου πρόβλεψης που φτιάξαμε με χρήση του μέτρου Root Mean Square Error
evaluator=RegressionEvaluator(metricName='rmse',labelCol='rating',predictionCol='prediction')
start=time()
rmse=evaluator.evaluate(prediction) # Αξιολόγηση με βάσει τις προβλέψεις
evaluationtime=time()-start
logger.debug('Algorithm executed and recommendations ready for viewing')
```

```
logger.debug(f'Evaluation executed after {evaluationtime} seconds')
logger.debug(f'Root Mean Square Error:{rmse}')

#------------------------ App Starting -------------------------------
userId=567 # User Id
n=10 #Number of Recommendations
ratingsFrame.createOrReplaceTempView('ratings')
# Εκτέλεση sql ερωτήματος
userDf=session.sql(f'SELECT userId FROM ratings WHERE userId={userId}').distinct()
userr=model.recommendForUserSubset(userDf,n)
userrecommendations=userr.rdd.collectAsMap() #DF-->RDD-->Dictionary
logger.info(f'Recommendations for {len(userrecommendations)} users made')

#################################### Recommendations
##########################################
logger.info(f'User sign in:{userId}')
table='<tr><th>Movie</th><th>Genre</th></tr>'
# Εκτέλεση sql ερωτήματος για εύρεση αξιολογήσεων του χρήστη που επιλέξαμε
userinfo=session.sql(f'SELECT movieid,rating FROM ratings WHERE userId={int(userId)}')
# Πρόσθεση των αξιολογήσεων στον html κώδικα
userrates='<tr><th>MOVIE</th><th>RATING</th></tr>'
message=''
serialout=''
for y in userinfo.collect():
        userrates+=f'<tr><td>{moviedb[str(y[0])][0]}</td><td>{moviedb[str(y[0])][1]}</td></tr>'
        message+=f"<li>{y[0]},{y[1]}</li>"
userrates+=f'<tr style=\"color:white; font-size:22px; background-color:#a410c9;\"><td>TOTAL
RATINGS</td><td>{userinfo.count()}</td></tr>'
logger.info(f'Total user Ratings:{userinfo.count()}')

userRecs=''
print('\t\tPosibilities')
for x in userrecommendations[int(userId)]:
    userRecs+=f'<tr><td>{moviedb[str(x[0])][0]}</td><td>{moviedb[str(x[0])][1]}</td></tr>'
    print(x)
    serialout+=f'<li>{x[0]}-->{x[1]}</li>'
########################### Output Production
#########################################################
code=rc.htmloutput()
logger.debug(str(code))
```

```
codeout=code.SliderShow(userId,userrates,userRecs)
logger.info(codeout)
obj=so.SparkObject(ps.__version__,sc.appName,sc.master,serialout,message,sc.getConf().getAll())
logger.info('Static Page for User {userid} made')
obj.Show(userId)
######################### App Stop ##############################################
RemoveFiles()
logger.info('Application Exceeded')
session.stop()
```

- recout.py (Αρχείο για τη δημιουργία στατικής ιστοσελίδας με σκοπό την αναπαράσταση των αποτελεσμάτων)

```
from datetime import datetime
import webbrowser as wb
import os


class htmloutput:
    def __init__(self):
        self.time=datetime.now().strftime('%Y/%M/%D  %H:%M;%S')
        self.htmlcode=''


    def SliderShow(self,userid,message,recommendations):
        backpath=os.path.join('..','Images','usersrec.jpg')
        self.htmlcode=f"""
        <html>
        <head>
        <title>{userid} recommendations</title>
        <style>
        """+"""
        * {box-sizing:border-box}
        .slideshow-container {
        max-width: 1000px;
        position: relative;
        margin: auto;
        }
        .prev, .next {
        cursor: pointer;
        position: absolute;
        top: 50%;
        width: auto;
```

```
margin-top: -22px;
padding: 16px;
color: white;
font-weight: bold;
font-size: 18px;
transition: 0.6s ease;
border-radius: 0 3px 3px 0;
user-select: none;
}
.next {
right: 0;
border-radius: 3px 0 0 3px;
}

.prev:hover, .next:hover {
background-color: rgba(0,0,0,0.8);
}
.text {
color: ;
font-size: 15px;
padding: 8px 12px;
position: absolute;
bottom: 8px;
font-size:21px;
width: 100%;
text-align: center;
}
.dot {
cursor: pointer;
height: 15px;
width: 15px;
margin: 0 2px;
background-color: #bbb;
border-radius: 50%;
display: inline-block;
transition: background-color 0.6s ease;
}

.active, .dot:hover {
background-color: #717171;
```

```
}
.fade {
-webkit-animation-name: fade;
-webkit-animation-duration: 1.5s;
animation-name: fade;
animation-duration: 1.5s;
}

@-webkit-keyframes fade {
from {opacity: .4}
to {opacity: 1}
}

@keyframes fade {
from {opacity: .4}
to {opacity: 1}
}
"""+"""
body{
font-family:calibri;
font-size:18px;
"""+f"""
background-image:url(\'{backpath}\');
"""+"""
background-repeat:no-repeat;
background-size:cover;
background-attachment:fixed;
color:white;
}
table{
background-color:white;
color:black;
font-size:19px;
width:70%;
text-align:center;
}
th{
    background-color:black;
    color:white;
    font-size:21px;
```

```
        }
        hr{
        border-top:2px solid red;
        }
        </style>
        </head>
        <body>
        """+f"""
        <center>
        <h1 style=\"margin-top:100px; width:80% font-weight:bold; text-decoration:underline; color:black;\"><img
src=\"https://beta.cloudnesil.com/wp-content/uploads/2018/12/apa>
        <hr>
        <h3 style=\"width:80%; font-weight:bold;\">Movies Evaluated by user {userid}</h3>
        <hr style=\"boredr:top 2px solid green;\">
        <div class="slideshow-container">
        <div class=\"mySlides Fade\">
        <table>
        {message}
        </table>
        <br>
        <div class=\"text\">{userid} Ratings</div>
        </div>
        <div class="mySlides Fade">
        <table>
        {recommendations}
        </table>
        <br>
        <div class="text">{userid} Recommendations</div>
        </div>
        </div>
        </center>
        <script>
        """
        self.htmlcode+="""{
        var slideIndex = 1;
        showSlides(slideIndex);
        // Next/previous controls
        function plusSlides(n)
        {
            showSlides(slideIndex += n);
```

```
    }
    function currentSlide(n) {
      showSlides(slideIndex = n);
    }
    function showSlides(n) {
    var i;
    var slides = document.getElementsByClassName("mySlides");
    var dots = document.getElementsByClassName("dot");
    if (n > slides.length) {slideIndex = 1}
    if (n < 1) {slideIndex = slides.length}
    for (i = 0; i < slides.length; i++) {
      slides[i].style.display = "none";
    }
    for (i = 0; i < dots.length; i++) {
      dots[i].className = dots[i].className.replace(" active", "");
    }
    slides[slideIndex-1].style.display = "block";
    dots[slideIndex-1].className += " active";
    }
    </script>
    </body>
    </html>
    """
    outpath=os.path.join('..','Static Pages',str(userid)+'.html')
    if  not os.path.exists(os.path.join('..','Static Pages')):
      os.mkdir(os.path.join('..','Static Pages'))
    with open(outpath,'w') as F:
      F.write(self.htmlcode)
    return f'File Output Created At {outpath}'

def __str__(self):
    return str(self.time)+'-->Html Slider Object Has been Created!!!!!'
```