



ΒΟΗΘΗΤΙΚΟ ΕΡΓΑΣΤΗΡΙΑΚΟ ΥΛΙΚΟ

ΡΟΗ Α – ΕΞΑΜΗΝΟ 9^ο

ΑΚ. ΕΤΟΣ 2020 - 2021

ΠΡΟΧΩΡΗΜΕΝΑ ΘΕΜΑΤΑ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

Apache Spark over Hadoop – Εγκατάσταση και Παραδείγματα

Το παρόν έγγραφο αποτελεί ένα αναλυτικό εργαστηριακό οδηγό σχετικά με το Apache Spark. Αρχικά περιγράφεται η εγκατάσταση του σε μία συστοιχία εικονικών μηχανημάτων στην υποδομή του Okeanos, ενώ στη συνέχεια δίνονται κάποια παραδείγματα σχετικά με την συγγραφή κώδικα με χρήση δύο βασικών προγραμματιστικών διεπαφών που παρέχει, το RDD API και το Dataframe API / SparkSQL. Για να ακολουθήσετε τον συγκεκριμένο οδηγό είναι απαραίτητο να έχετε ολοκληρώσει επιτυχώς την προπαρασκευή του εργαστηρίου και να έχετε αποκτήσει τα εικονικά μηχανήματα της υπηρεσίας Okeanos.

Το κομμάτι της εγκατάστασης αφορά μία πιο αναλυτική περιγραφή της επίδειξης που έγινε στο εργαστηριακό μάθημα.

1 Αναλυτικός οδηγός εγκατάστασης του Apache Spark

Η εγκατάσταση περιλαμβάνει τα εξής συστήματα:

- hdfs: Κατανεμημένο filesystem, από το οποίο διαβάζουν και γράφουν τα Spark jobs.
- Spark: Open source framework γενικού σκοπού για επεξεργασία δεδομένων, που περιλαμβάνει υλοποίηση του προγραμματιστικού μοντέλου MapReduce.

Σε μία συστοιχία μηχανημάτων με τα παραπάνω συστήματα, οι κόμβοι που συμμετέχουν μπορούν να έχουν τους ακόλουθους ρόλους:

Πίνακας 1: Ρόλοι κόμβων σε μία συστοιχία υπολογιστών

hdfs namenode	Πρόκειται για τον κεντρικό κόμβο του hdfs. Ο NameNode γνωρίζει ποιος datanode διατηρεί ποιο block ενός αρχείου.
hdfs datanode	Οι DataNodes περιέχουν κομμάτια (blocks) από τα αρχεία του hdfs. Αναλαμβάνουν να «σερβίρουν» δεδομένα σε κλήσεις εξωτερικών πελατών.
Spark master	Ο master είναι ο κεντρικός κόμβος του Spark που ελέγχει τους διαθέσιμους πόρους και με βάση αυτούς δρομολογεί και διαχειρίζεται τις κατανεμημένες εφαρμογές που τρέχουν στο

	cluster.
Spark worker	Ο worker είναι μια διεργασία του Spark, η οποία τρέχει σε κάθε κόμβο του cluster και διαχειρίζεται τις προς εκτέλεση διεργασίες των κατανεμημένων εφαρμογών στον κόμβο αυτό.

1.1 Γενικές ρυθμίσεις

Βρίσκουμε από το UI του Okeanos την public IP του master vm που κατασκευάστηκε, η οποία θα είναι της μορφής 83.212.XX.YY, καθώς και τις private IP του master και του slave vm, οι οποίες είναι της μορφής 192.168.ZZ.WW η καθεμία. Βεβαιωνόμαστε ότι έχουμε τους κωδικούς σύνδεσης στα μηχανήματα master και slave, οι οποίοι μας δόθηκαν κατά την κατασκευή των μηχανημάτων από την υπηρεσία του Okeanos.

Με χρήση του ssh, συνδεόμαστε στο μηχάνημα master. Για παράδειγμα μέσω linux/mac terminal συνδεόμαστε ως `ssh user@83.212.XX.XX`. Στην συνέχεια θα μας ζητηθεί να εισάγουμε κωδικό πρόσβασης, ο οποίος είναι ο κωδικός που έχει παραχθεί από το σύστημα του Okeanos κατά τη δημιουργία του μηχανήματος.

Το πρώτο βήμα αφορά στην ρύθμιση κατάλληλου ονόματος για το μηχάνημα αντί του snf-KLMN που έχει. Ο ορισμός του ονόματος γίνεται με χρήση της ακόλουθης εντολής

```
sudo hostname master
```

Εάν μας ζητηθεί εισάγουμε και πάλι των κωδικό πρόσβασης. Στην συνέχεια τροποποιούμε το αρχείο hosts, το οποίο χρησιμεύει για την εύκολη εύρεση μηχανημάτων προς σύνδεση μέσω ονομάτων αντί για διευθύνσεις IP. Αυτό είναι απαραίτητο καθώς στην συνέχεια θα πρέπει να ορίσουμε στο Spark ποιο μηχάνημα έχει πιο ρόλο και κατά την εκτέλεση εφαρμογών θα πρέπει το ένα μηχάνημα να μπορεί να επικοινωνήσει με το άλλο. Για να τροποποιήσουμε το αρχείο τρέχουμε στο terminal την εντολή

```
sudo vim /etc/hosts
```

ώστε να ανοίξει το αρχείο στο Vim. Για να το επεξεργαστούμε πατάμε το πλήκτρο <Insert> ή το πλήκτρο <i>. Μεταβαίνουμε στο τέλος του αρχείου και εισάγουμε Και στο τέλος το αρχείου εισάγουμε τις δύο ακόλουθες γραμμές.

```
<private_ip_master> master
<private_ip_slave> slave
```

Για να αποθηκευτούν οι αλλαγές στο αρχείο μέσω Vim πατάμε αρχικά <Esc> για να βγούμε από τη λειτουργία εισαγωγής κειμένου και στην συνέχεια πληκτρολογούμε «:wq» και πατάμε Enter. Βασικές λειτουργίες του Vim μπορείτε να βρείτε [εδώ](#).

!Προσοχή! Εάν δεν βάλετε σωστά τις private IP δεν θα λειτουργήσει το Spark και το Hadoop στην συνέχεια.

Στην συνέχεια επαναλαμβάνουμε τα ίδια βήματα και στον slave. Συνδεόμαστε με πληκτρολογώντας στο terminal

```
ssh slave
```

και εισάγουμε τον κωδικό του μηχανήματος. Έχοντας συνδεθεί στο μηχάνημα, τροποποιούμε αρχικά το hostname του με

```
sudo hostname slave
```

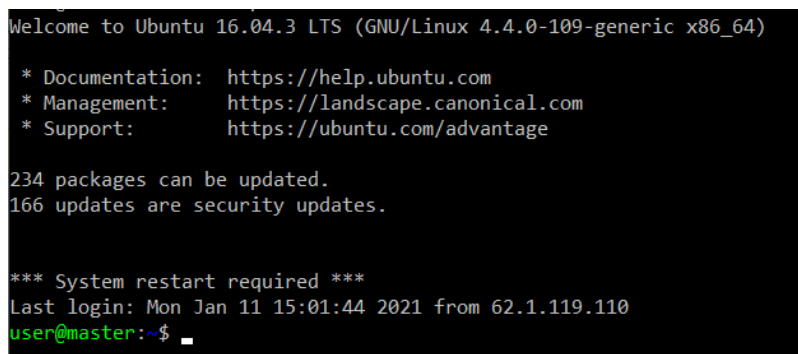
Επεξεργαζόμαστε και πάλι το αρχείο hosts με

```
sudo vim /etc/hosts
```

και στο τέλος του προσθέτουμε και πάλι τις δυο γραμμές

```
<private_ip_master> master  
<private_ip_slave> slave
```

Σώζουμε το αρχείο και στην συνέχεια αποσυνδεόμαστε από τον slave προς τον master και από τον master προς τον υπολογιστή μας. Η αποσύνδεση γίνεται πατώντας Ctrl+D. Στη συνέχεια ξανασυνδεόμαστε πίσω στον master και παρατηρούμε ότι το hostname έχει αλλάξει. Θα πρέπει να δείτε μία εικόνα της μορφής



Εικόνα 1: Παράδειγμα από σύνδεση στο master έχοντας ρυθμίσει το hostname

1.2 Λήψη αρχείων εγκατάστασης

Έχοντας συνδεθεί στον master, εκτελέστε στο home directory την ακόλουθη εντολή ώστε να λάβετε τα αρχεία που είναι απαραίτητα για την επιτυχή εγκατάσταση του Spark στη συστοιχία.

```
wget --no-check-certificate \  
'https://docs.google.com/uc?export=download&id=1f4p9uMzvzv2oq2t1FkvlhLpHWDtj-d01' \  
-O installers.tar.gz
```

Το tar που θα λάβετε περιέχει μέσα scripts για ρυθμίσεις σύνδεσης και δικτυακές ρυθμίσεις των εικονικών μηχανημάτων, καθώς και script για εγκατάσταση της java του Hadoop, που περιέχει και το hdfs και του ίδιου του spark. Αποσυμπίεστε τα αρχεία εγκατάστασης με χρήση της εντολής

```
tar -xzf installers.tar.gz
```

1.3 Ρύθμιση για passwordless ssh μεταξύ master – slave

Το Spark αλλά και το hdfs για να λειτουργήσουν θα πρέπει οι υπολογιστές που αποτελούν το Spark ή το hdfs cluster να μπορούν να συνδέονται μεταξύ τους με την χρήση ssh χωρίς κωδικό. Αυτό επιτυγχάνεται με την χρήση ζεύγους ιδιωτικού/δημόσιου κλειδιού1 . Η βασική ιδέα της τεχνικής αυτής είναι ότι ο κάθε χρήστης μπορεί να έχει ένα ιδιωτικό κλειδί (αρχείο) το οποίο έρχεται ζεύγος με ένα δημόσιο κλειδί. Ο χρήστης τοποθετεί το δημόσιο κλειδί στους υπολογιστές τους οποίους θέλει να έχει πρόσβαση, και με την χρήση του ιδιωτικού μπορεί να συνδέεται σε αυτούς χωρίς κωδικό. Η πιστοποίηση του χρήστη γίνεται καθώς μόνο ο χρήστης έχει στην κατοχή του το ιδιωτικό κλειδί. Για να υλοποιήσετε την συγκεκριμένη διαδικασία εκτελέστε στον master:

```
~/scripts/set_passwordless_ssh.sh
```

Εισάγετε τον κωδικό του slave όταν ζητηθεί ώστε να ολοκληρωθεί η διαδικασία.

Αντιγράψουμε συνολικά τα αρχεία εγκατάστασης στο slave

```
scp -r ~/scripts user@slave:.
```

1.4 Δημιουργία NAT

Από τα δύο VMs που φτιάξαμε και τα δύο έχουν private IPv4, αλλά μόνο ο master έχει public IPv4. Αυτό σημαίνει ότι στο ισχύον setup μόνο ο master μπορεί να βλέπει σε όλο το δίκτυο του «έξω κόσμου». Επειδή θα χρειαστεί να εγκαταστήσουμε διάφορα πακέτα και στον slave, θέλουμε να δώσουμε και σε αυτόν πλήρη πρόσβαση στο εξωτερικό δίκτυο. Για να το πετύχουμε αυτό δημιουργούμε και τρέχουμε στον master το παρακάτω script:

```
sudo ~/scripts/nat_master.sh
```

Στη συνέχεια, συνδεόμαστε στο slave μέσω ssh

```
ssh slave
```

και από τον slave πλέον εκτελούμε

```
sudo ~/scripts/nat_slave.sh
```

Αφου ολοκληρωθούν τα παραπάνω, αποσυνδεόμαστε από τον slave πίσω στον master.

1.5 Εγκατάσταση Java

Τόσο Hadoop όσο και το Spark χρειάζονται ένα JVM για να τρέξουν. Επομένως, είναι απαραίτητο να εγκαταστήσουμε την java τόσο στον master όσο και στον slave. Για την εγκατάσταση στον master εκτελούμε:

```
sudo ~/scripts/install_java.sh
```

Στη συνέχεια, συνδεόμαστε στο slave μέσω ssh

```
ssh slave
```

και από τον slave πλέον εκτελούμε

```
sudo ~/scripts/install_java.sh
```

Αφου ολοκληρωθούν τα παραπάνω, αποσυνδεόμαστε από τον slave πίσω στον master.

1.6 Εγκατάσταση Hadoop

Για την εγκατάσταση του Hadoop δίνεται αυτοματοποιημένο script που εγκαθιστά και ρυθμίζει το hadoop και στον master και στον slave. Συνεπώς, εκτελούμε το script μόνο από

```
τον  
~/scripts/install_hadoop.sh
```

master.

1.7 Εγκατάσταση Spark

Για την εγκατάσταση του Spark δίνεται αυτοματοποιημένο script που εγκαθιστά και ρυθμίζει το spark και στον master και στον slave. Συνεπώς, εκτελούμε μόνο από τον master.

```
~/scripts/install_spark.sh
```

Στη συνέχεια, μεταβείτε στον φάκελο /home/user/spark-2.4.4-bin-hadoop2.7/conf με

```
cd /home/user/spark-2.4.4-bin-hadoop2.7/conf
```

και επεξεργαστείτε το αρχείο spark-defaults.conf με

```
vim spark-defaults.conf
```

και αλλάξτε τα \t\t με κανονικά tabs για να λειτουργήσει μετέπειτα σωστά το spark.

Τα αυτοματοποιημένα script των προηγούμενων βημάτων σετάρουν κάποιες μεταβλητές περιβάλλοντος στο αρχείο ~/.bashrc, το οποίο διαβάζεται κάθε φορά κατά την εκκίνηση ενός interactive ssh session. Για να ενημερωθεί το session που βρισκόμαστε τώρα, εκτελούμε

```
source ~/.bashrc
```

1.8 Format filesystem και εκκίνηση Hdfs/Spark

Πριν ξεκινήσουμε το hadoop και το spark πρέπει να κάνουμε format το κατανεμημένο file system. Γι' αυτό εκτελούμε:

```
hdfs namenode -format
```

Σημειώνεται ότι αυτή η διαδικασία γίνεται μόνο την πρώτη φορά αμέσως μετά την εγκατάσταση του Hadoop. Στη συνέχεια εκκινούμε το hdfs εκτελώντας

```
start-dfs.sh
```

και στη συνέχεια εκκινούμε το Spark εκτελώντας

```
start-all.sh
```

Βεβαιωνόμαστε ότι όλα τα Services τρέχουν, εκτελώντας

```
jps && ssh slave jps
```

Από όπου πρέπει να λάβουμε αντίστοιχο output:

```
18592 DataNode
18774 SecondaryNameNode
18903 Master
18443 NameNode
19003 Worker
26495 Jps
18358 Worker
18233 DataNode
19978 Jps
```

1.9 Εκτέλεση παράδειγματος σε Spark

Τρέχουμε και έναν έτοιμο κώδικα spark για περαιτέρω επιβεβαίωση, ο οποίος υπολογίζει τον αριθμό π. Η εκτέλεση γίνεται χρησιμοποιώντας το spark-submit ως εξής

```
spark-submit --class org.apache.spark.examples.JavaSparkPi /home/user/spark-2.4.4-bin-hadoop2.7/examples/jars/spark-examples_2.11-2.4.4.jar
```

και θα τυπωθεί ανάμεσα στα logs στο ακόλουθο μήνυμα.

```
Pi is roughly 3.1449
```

2 Οδηγίες χρήσης hdfs

Το hdfs αποτελεί ένα κατανεμημένο σύστημα αρχείων, το οποίο θα χρησιμοποιηθεί στα πλαίσια του μαθήματος ώστε να διαβάζει το Spark από εκεί τα δεδομένα που θα χρησιμοποιεί στο κάθε job. Όπως θα φανεί στη συνέχεια, όλες σχεδόν οι εντολές του hdfs θυμίζουν εντολές του Unix για τη διαχείριση αρχείων. Όλα τα παραδείγματα εντολών διαχείρισης αρχείων δίνονται στα πλαίσια του hdfs που έχει στηθεί στην υποδομή του Okeanos και ακούει στη διεύθυνση master:9000.

2.1 Δημιουργία φακέλου

Για τη δημιουργία φακέλου στο hdfs εκτελούμε την εντολή

```
hadoop fs -mkdir hdfs://master:9000/<rest_of_path_to_check_contents>
```

Για παράδειγμα, για τη δημιουργία του φακέλου myfolder στο home directory του hdfs θα εκτελούσαμε την εντολή:

```
hadoop fs -mkdir hdfs://master:9000/myfolder
```

2.2 Ανέβασμα αρχείου σε κάποιο φάκελο του hdfs

Για το ανέβασμα ενός αρχείου από το τοπικό file system στο hdfs χρησιμοποιούμε την εντολή

```
hadoop fs -put <file_name> hdfs://master:9000/<rest_path_to_upload>
```

Για παράδειγμα, για το ανέβασμα του τοπικού αρχείου afile.txt στον φάκελο myfolder του hdfs εκτελούμε

```
hadoop fs -put afile.txt hdfs://master:9000/myfolder/.
```

2.3 Προβολή αρχείων εντός φακέλου του hdfs

Για να δούμε τα αρχεία που υπάρχουν σε ένα φάκελο του hdfs χρησιμοποιούμε την εντολή

```
hadoop fs -ls hdfs://master:9000/<folder_path>
```

Ένα παράδειγμα για την προβολή των αρχείων και των φακέλων που υπάρχουν στο root directory του hdfs είναι

```
hadoop fs -ls hdfs://master:9000/.
```

ενώ το αντίστοιχο παράδειγμα για προβολή των αρχείων / φακέλων στο φάκελο myfolder είναι

```
hadoop fs -ls hdfs://master:9000/myfolder/.
```

Περισσότερες πληροφορίες από αυτές που παρατίθενται για την χρήση του hdfs μπορείτε να βρείτε [εδώ](#), όπου το hdfs dfs είναι alias για το hadoop fs που δίνεται στα παραδείγματα παραπάνω.

3 Χρήση του Apache Spark

3.1 Εκτέλεση ενός προγράμματος στο Spark

Για την εκτέλεση ενός προγράμματος στο Apache Spark χρησιμοποιούμε το spark-submit. Εάν έχουμε κώδικα python εκτελούμε απλά το python script με spark-submit, ενώ εάν έχουμε java ή scala κώδικα κατασκευάζουμε το αντίστοιχο jar το οποίο εκτελούμε με spark-submit. Στη συνέχεια, δίνονται ενδεικτικά παραδείγματα για την εκτέλεση ενός python script myscript.py και για την εκτέλεση ενός jar myjar.jar

```
spark-submit myscript.py <script_arg1> <script_arg2> ... <script_argN>
```

```
spark-submit --class path.in.jar.to.to.class.with.Main \
```

```
myjar.jar <script_arg1> <script_arg2> ... <script_argN>
```

3.2 Διαφορές Spark Session και Spark Context

Για να μπορέσουμε να δώσουμε των κώδικα μας στο Spark να εκτελεστεί χρειαζόμαστε ένα σημείο εισόδου. Τόσο το spark session όσο και το spark context μπορούν να παίξουν το ρόλο του σημείου εισόδου. Το spark context είναι σημείο εισόδου για να χρησιμοποιηθεί κώδικας από το RDD API του Spark, ενώ το spark session είναι ένα γενικό σημείο εισόδου για να χρησιμοποιηθεί κώδικας από οποιοδήποτε API του Spark.

3.3 Προετοιμασία για την εκτέλεση παραδειγμάτων

Για την εκτέλεση των παραδειγμάτων που δίνονται στις ακόλουθες ενότητες κατεβάστε αρχικά τα αρχεία που θα χρησιμοποιηθούν για αυτά με χρήση της εντολής

```
wget --no-check-certificate \
    'https://docs.google.com/uc?export=download&id=1k8-U4XXXODGnSZqGwp8y2vwAGXeurNab' \
    -O samples_atds.tar.gz
```

και αποσυμπίεστε το tar με χρήση της εντολής

```
tar -zxf samples_atds.tar.gz
```

Στη συνέχεια φορτώστε τα αρχεία text.txt, employees.csv και departments.csv στο hdfs σε έναν φάκελο examples που θα κατασκευάσετε με χρήση της αλληλουχίας εντολών

```
hadoop fs -mkdir hdfs://master:9000/examples
```

```
hadoop fs -put departments.csv hdfs://master:9000/examples/.
```

```
hadoop fs -put employees.csv hdfs://master:9000/examples/.
```

```
hadoop fs -put text.txt hdfs://master:9000/examples/.
```

3.4 Βασικό Παράδειγμα: WordCount

Το βασικότερο παράδειγμα σχετικά με την συγγραφή Map Reduce κώδικα είναι η μέτρηση λέξεων ενός κειμένου. Ένας βασικός ψευδοκώδικας για το wordcount είναι ο ακόλουθος:

```
map(_, value):
    tokens = split(value, " ")
    for token in tokens:
        emit(token, 1)

reduce(token, list_of_aces):
    count = 0
    for ace in list_of_aces:
```

```

        count += ace
    emit(token, count)

```

Το συγκεκριμένο παράδειγμα δίνεται στη συνέχεια ως MapReduce κώδικας στο RDD API του Spark και γλώσσα python, λίγο τροποποιημένο ώστε να βρίσκει τις 10 δημοφιλέστερες λέξεις.

```

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("WordCountExample").getOrCreate()

sc = spark.sparkContext

popular_words = sc.textFile("hdfs://master:9000/examples/text.txt"). \
    flatMap(lambda x : x.split(" ")). \
    map(lambda x : (x, 1)). \
    reduceByKey(lambda x, y : x + y). \
    sortBy(lambda x: x[1], ascending=False)

for i in popular_words.take(10):
    print(i)

```

Αν ο κώδικας βρίσκεται στο αρχείο wordcount.py εκτελούμε ως
spark-submit wordcount.py

Επεξήγηση του κώδικα:

Αρχικά παρατηρούμε ότι στην περίπτωση του Ψευδοκώδικα θέλουμε να γίνει emit κάθε μία λέξη της γραμμής που λαμβάνει ως είσοδο ο mapper. Έτσι χωρίζουμε την γραμμή στον κενό χαρακτήρα σε ένα μετασχηματισμό flatMap. Ο μετασχηματισμός flatMap χρησιμοποιείται όταν θέλουμε να προσομοιώσουμε τα πολλαπλά emits σε ένα mapper, όπως στο for του ψευδοκώδικα. Ο μετασχηματισμός κάνει emit ξεχωριστά κάθε ένα στοιχείο της λίστας που εμπεριέχεται στο αποτέλεσμα που θα παίρνει η lambda συνάρτηση που του δίνεται ως όρισμα. Στη συνέχεια για να μετασχηματίσουμε κάθε μία λέξη ώστε να έχει τη μοναδιαία εμφάνιση της που είναι γνωστή, χρησιμοποιούμε έναν απλό μετασχηματισμό map ο οποίος για κάθε λέξη που x που μπαίνει ως είσοδο στο mapper επιστρέφει την τούπλα (x, 1) δηλώνοντας ότι η λέξη x εμφανίζεται 1 φορά. Αυτοί οι άσσοι θα χρησιμοποιηθούν στο reduce stage ακολούθως και αν αθροιστούν για μια δεδομένη λέξη θα δώσουν την συχνότητα εμφάνισης της. Παρατηρούμε ότι στο reduceByKey, για να υλοποιηθεί αυτή η διαδικασία δίνεται η ακόλουθη συνάρτηση ως είσοδος

```

lambda x, y : x + y

```

Η συνάρτηση αυτή προσομοιώνει το count += ace του ψευδοκώδικα του wordcount θεωρώντας ως count το x και ως ace το y. Πρακτικά η ανώνυμη αυτή συνάρτηση δύο ορισμάτων στο στάδιο reduceByKey πάντα θεωρεί ως πρώτο όρισμα τον accumulator που θέλουμε να επιστρέψουμε (π.χ. μέγιστο / ελάχιστο στοιχείο, άθροισμα, γινόμενο, κλπ) και ως δεύτερο το τρέχον στοιχείο που θα συμπεριληφθεί. Τελικά χρησιμοποιούμε την sortBy, η οποία παίρνει 2 ορίσματα, για να ταξινομήσουμε τις λέξεις ως προς τη συχνότητα εμφάνισης τους. Το πρώτο δείχνει ποιο στοιχείο της τούπλας είναι το στοιχείο ως προς το οποίο θα γίνει η ταξινόμηση, και το δεύτερο εάν επιθυμούμε αύξουσα ή φθίνουσα ταξινόμηση. Τελικά μέσω του take(10) παίρνουμε τις 10 λέξεις με τη μεγαλύτερη συχνότητα εμφάνισης, δηλαδή τις πιο δημοφιλείς. Εναλλακτικά εάν θέλαμε όλες τις λέξεις με τη συχνότητα εμφάνισης τους θα χρησιμοποιούσαμε τον μετασχηματισμό collect.

Εκτελώντας το αρχείο αυτό του κώδικα, λαμβάνουμε την ακόλουθη έξοδο.

```
('the', 42)
('of', 27)
('I', 24)
('and', 18)
('my', 17)
('a', 9)
('in', 7)
('with', 7)
('that', 7)
('feel', 5)
```

3.5 SQL ερωτήματα στο Apache Spark

Μεταξύ των αρχείων που σας δόθηκαν υπάρχει το employees.csv και το departments.csv τα οποία αποτελούν ένα τμήμα μιας βάσης δεδομένων. Ο πρώτος πίνακας περιγράφει τους υπαλλήλους της εταιρίας και ο δεύτερος τα τμήματα αυτής.

Ο πίνακας employees.csv περιέχει την ακόλουθη πληροφορία

Θέση	0	1	2	3
Πληροφορία	id υπαλλήλου	Όνομα Επώνυμο	Ετήσιος Μισθός	id τμήματος εργασίας

Ο πίνακας departments.csv περιέχει την ακόλουθη πληροφορία

Θέση	0	1
Πληροφορία	id τμήματος εργασίας	Όνομα Τμήματος

Στόχος είναι να υλοποιήσουμε τα δύο επόμενα queries:

Query 1	Να βρεθούν οι 5 υπάλληλοι με τον υψηλότερο μηνιαίο μισθό. Τα αποτελέσματα να δοθούν στην μορφή Επίθετο ΑρχικόΌνόματος, Μηνιαίος Μισθός
Query 2	Για κάθε τμήμα, να βρεθεί το πλήθος των υπαλλήλων που απασχολεί και οι συνολικές μισθολογικές δαπάνες του. Τα αποτελέσματα να είναι στη μορφή ΌνομαΤμήματος, #Υπαλλήλων, ΈξοδαΜισθών

Αρχικά, για τις υλοποιήσεις σημειώνεται ότι σε κάθε κώδικα φτιάχνουμε αρχικά τη μεταβλητή για το Entrypoint του spark session. Αν χρησιμοποιούμε το RDD API λαμβάνουμε από το spark session το spark context.

3.5.1 RDD API

Ερώτημα 1

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession.builder.appName("query1-rdd").getOrCreate()
```

```
sc = spark.sparkContext
```

```
def format_name(name):
    fName = name.split(" ")[0]
    lName = name.split(" ")[1]
    return lName + " " + fName[0] + "."
```

```
res = \
```

```

sc.textFile("hdfs://master:9000/examples/employees.csv"). \
map(lambda x : (int(x.split(",")[2])/12, \
    format_name(x.split(",")[1]))). \
sortByKey(ascending=False). \
map(lambda x : (x[1], x[0])). \
take(5)

for i in res:
    print(i)

```

Ο κώδικας διαβάζει το αρχείο employees και απομονώνει τις χρήσιμες για το ερώτημα 1 στήλες: το ονοματεπώνυμο του υπαλλήλου και τον ετήσιο μισθό (στήλες 1 και 2 αντίστοιχα) μέσω του πρώτου map. Εκεί μετασχηματίζονται κιόλας στην επιθυμητή μορφή τους (μηνιαίος μισθός και Επίθετο ΑρχικόΟνόματος.) Ο δεύτερος μετασχηματισμός γίνεται μέσω της συνάρτησης format_name που κατασκευάζεται παραπάνω. Στην συνέχεια, κάνουμε φθίνουσα ταξινόμηση για να βρούμε τους υπάλληλους με το μεγαλύτερο μισθό ενώ στο δεύτερο map αλλάζουμε τη σειρά εμφάνισης ώστε να είναι η επιθυμητή. Για να πάρουμε τους 5 πρώτους χρησιμοποιούμε το take(5). Η έξοδος του παραπάνω κώδικα είναι

```

('Dimitriou C.', 2919.0)
('Papachristou N.', 2889.0)
('Stamatiou N.', 2848.0)
('Papachristou D.', 2817.0)
('Papageorgiou D.', 2787.0)

```

Ερώτημα 2

```

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("query1-rdd").getOrCreate()

sc = spark.sparkContext

employees = \
    sc.textFile("hdfs://master:9000/examples/employees.csv"). \
    map(lambda x : (x.split(",")[3], (int(x.split(",")[2]), 1))). \
    reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))

departments = \
    sc.textFile("hdfs://master:9000/examples/departments.csv"). \
    map(lambda x : (x.split(",")[0], x.split(",")[1]))

res = employees.join(departments). \
    map(lambda x : (x[1][1], x[1][0][1], x[1][0][0]))

for i in res.collect():
    print(i)

```

Για το ερώτημα 2 αξίζει να σχολιασθεί η χρήση του join. Για να εφαρμόσουμε το join για τη συνένωση 2 πινάκων πρέπει να τα δεδομένα κάθε πίνακα που συμμετέχει να είναι στην μορφή

(key, (value1, value2, ..., valueN))

όπου key θα είναι το πεδίο ή μία τούπλα πεδίων πάνω στην οποία είναι επιθυμητό να γίνει το join. Οι υπόλοιπες τιμές θα είναι εντός μίας άλλης τουπλας όπως φαίνεται παραπάνω. Το αποτέλεσμα της συνένωσης των πινάκων θα έχει τη μορφή

(key, ((table1_value1, ..., table1_valueN), (table2_value1, ..., table2_valueM)))

δηλαδή θα είναι ένα key-value αποτέλεσμα με key αυτό της συνένωσης και value που περιέχει τις τούπλες με τα values του κάθε πίνακα. Σημειώνεται ότι εάν από έναν πίνακα έχουμε μοναδικό value αυτό δεν χρειάζεται να είναι εντός τούπλας. Η έξοδος αυτού του κώδικα είναι

```
('Data Science', 8, 213300)
('Software Engineering', 9, 241848)
('Research', 8, 224844)
```

3.5.2 SparkSQL

Χρησιμοποιώντας sparkSQL μπορούμε να γράψουμε κανονική sql για να απαντήσουμε τέτοιου τύπου ερωτήματα. Φορτώνουμε αρχικά όποιον πίνακα-csv θέλουμε μέσω του

```
spark.read.format(<format_αρχείου>).options(...).load(<hdfs_path>)
```

Συνήθως σαν option χρησιμοποιούμε το inferSchema στην τιμή true ώστε να καταλάβει το Spark τον τύπο της κάθε στήλης στο dataframe που θα διαβάσει. Σημειώστε ότι το SparkSQL θα λειτουργήσει καλύτερα με αυτήν την πληροφορία καθώς περιέχει βελτιστοποιητή ερωτημάτων, όπως οι κλασσικές βάσεις δεδομένων. Για να μπορούμε να χρησιμοποιήσουμε έναν πίνακα εντός sparkSQL πρέπει πρώτα να δηλώσουμε το αντίστοιχο dataframe. Αυτό γίνεται με την registerTempTable ενός dataframe που θα το δηλώσει στα ερωτήματα με όνομα αυτό που θα δοθεί ως όρισμα. Επιπλέον, εάν θέλουμε να χρησιμοποιήσουμε κάποια συνάρτηση που δεν υπάρχει έτοιμη στο SparkSQL όπως πχ η format_name του Q1, μπορούμε αρκεί να τη δηλώσουμε στο SparkSQL. Αυτό θεωρείται ένα user defined function (UDF) και δηλώνεται ως εξής

```
spark.udf.register(<όνομαΕντόςSQL>, όνομαΣυνάρτησης)
```

Στην συνέχεια δίνονται ως παράδειγμα οι υλοποιήσεις των 2 query.

Ερώτημα 1

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("query1-rdd").getOrCreate()

def format_name(name):
    fName = name.split(" ")[0]
    lName = name.split(" ")[1]
    return lName + " " + fName[0] + "."

employees = spark.read.format('csv'). \
    options(header='false', \
            inferSchema='true'). \
    load("hdfs://master:9000/employees.csv")

employees.registerTempTable("employees")
spark.udf.register("formatter", format_name)

sqlString = \
    "select formatter(_c1) as Name, _c2/12.0 as MonthSalary " + \
    "from employees " + \
    "order by _c2 desc " + \
    "limit 5"

res = spark.sql(sqlString)
```

```
res.show()
```

Ερώτημα 2

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("query1-rdd").getOrCreate()

employees = spark.read.format("csv").options(header='false',
inferSchema='true').load("hdfs://master:9000/employees.csv")
departments = spark.read.format("csv").options(header='false',
inferSchema='true').load("hdfs://master:9000/departments.csv")

employees.registerTempTable("employees")
departments.registerTempTable("departments")

sqlString = \
    "select " + \
        "first(d._c1) as DeptName, " + \
        "count(*) as NumEmployees, " + \
        "sum(e._c2) as SalariesInTotal " + \
    "from employees as e, departments as d " + \
    "where e._c3 == d._c0 " + \
    "group by e._c3"

res = spark.sql(sqlString)

res.show()
```