ΤΕΙ ΗΠΕΙΡΟΥ

ΣΧΟΛΗ: ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ: ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ Τ.Ε

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

USING APACHE HADOOP WITH MAPREDUCE, SPARK, PIG AND HIVE TO PERFORM MODELING AND GRAPH ANALYTICS TO MODEL PROBLEMS

Κωνσταντίνος Καραθύμιος

Επιβλέπουσα: Σπυριδούλα Μαργαρίτη

Άρτα, Ιούνιος, 2018

ΤΕΙ ΗΠΕΙΡΟΥ

ΣΧΟΛΗ: ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ: ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ Τ.Ε

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Κωνσταντίνος Καραθύμιος

Επιβλέπουσα: Σπυριδούλα Μαργαρίτη

Άρτα, Ιούνιος, 2018

USING APACHE HADOOP WITH MAPREDUCE, SPARK, PIG AND HIVE TO PERFORM MODELING AND GRAPH ANALYTICS TO MODEL PROBLEMS

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

Άρτα, 2018

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπουσα καθηγήτρια
   Σπυριδούλα Μαργαρίτη
   Εργαστηριακό Διδακτικό Προσωπικό

2. Χαριλόγης Βασίλειος
   Εργαστηριακό Διδακτικό Προσωπικό

3. Λιαροκάρπης Δημήτρης
   Καθηγητής Εφαρμογής

Ο/Η Προϊστάμενος/η του Τμήματος
Χρυσόστομος Στύλιος

Υπογραφή

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα πτυχιακή εργασία είναι εξολοκλήρου 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

Καραθύμιος, Κωνσταντίνος

Υπογραφή

ΕΥΧΑΡΙΣΤΙΕΣ

Ευχαριστώ πολύ τους γονείς μου για την υποστήριξη που έλαβα για να φτάσω στην ολοκλήρωση των σπουδών μου, που όσο δύσκολο και αν υπήρξε, ήταν συνεχώς στο πλευρό μου. Επίσης, ευχαριστώ την κ. Σπυριδούλα Μαργαρίτη για την καθοδήγηση που μου προσέφερε κατά την διάρκεια φοίτησης μου, αλλά και για την ανάληψη της πτυχιακής εργασίας και τη βοήθεια που προσέφερε για την ολοκλήρωσή της.

Περίληψη

Με την βοήθεια των εργαλείων Apache Spark, Hadoop και της γλώσσας Scala, γίνεται ανάλυση μεγάλων δεδομένων (Big Data). Τα δεδομένα, που είναι ανώνυμα, προέρχονται από μεγάλα και σύνθετα δίκτυα όπως αυτά των μέσων κοινωνικής δικτύωσης. Τα δίκτυα αυτά αναπαρίστανται ως γράφοι και με την ανάλυση εξάγονται συμπεράσματα, προκύπτουν νέοι γράφοι και γίνεται δυνατή η κατανόηση τους. Κατά την ανάλυση, εφαρμόζονται αλγόριθμοι όπως Google's PageRank, Community detection algorithms και άλλοι αναλυτικοί αλγόριθμοι. Εκτός από ανάλυση, γίνεται απεικόνιση των δεδομένων με τη χρήση του Gephi και συγκεκριμένων αλγορίθμων οπτικοποίησης τους. Παρουσιάζονται εφαρμογές ανάλυσης δεδομένων (graph analytics) σε δεδομένα που συλλέχθηκαν από κοινωνικά μέσα δικτύωσης και διατίθενται ελεύθερα.

Λέξεις-κλειδιά: ανάλυση, γράφοι, κοινότητες, αλγόριθμοι κατάταξης, ανάλυση γράφων

ABSTRACT

Using the tools Apache Spark, Hadoop with Scala programming language, I present graph analytics on Big Data in order to extract conclusions about data sourcing from Social Media in an anonymized form. During the analysis, these are divided into new subgraphs which include more accurate and clustered information depending on data variables. The application of Google's PageRank, Community detection algorithm and more analytic algorithms are being shown. The visualization of graphs is processed with Gephi which uses many visualization algorithms in order to show a graph structure. Some real-life applications of graph analytics are being shown and how they affecting our lives.

Keywords: graph analytics, PageRank, communities' algorithm, big data, apache spark

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΓΛΩΣΣΑΡΙΟ

Superstep: Το superstep αποτελεί μια μονάδα γενικού προγραμματισμού, το οποίο επικοινωνεί με τα επιμέρους προγράμματα μέσω ενός παγκοσμίου παράγοντα, κάνοντας χιλιάδες παράλληλες διεργασίες, συγχρονίζοντας όλα τα στοιχεία σε ένα σημείο που ονομάζεται φράγμα συγχρονισμού.

Υπεργράφημα: Από τα μαθηματικά εάν ο Α γράφος είναι υπο-γράφος του Β τότε το Β είναι υπεργράφημα του Α.

Framework: Συνήθως περιέχει το API και αποτελεί το θεμέλιο για τον προγραμματισμό μιας εφαρμογής και το API παρέχει πρόσβαση σε στοιχεία που υποστηρίζει το Framework. Επιπλέον περιέχει βιβλιοθήκες και άλλα στοιχεία που βοηθάνε στην ανάπτυξη λογισμικού.

Cross-platform: Είναι το λογισμικό που είναι υλοποιημένο με υποστήριξη πολλαπλών λειτουργικών συστημάτων.

JVM: Το JVM (Java Virtual Machine) είναι μία «εικονική μηχανή» που επιτρέπει στον υπολογιστή να εκτελεί προγράμματα σε Java.

IDE: Το IDE (Integrated Development Environment) είναι μία εφαρμογή η οποία παρέχει στους προγραμματιστές τα εργαλεία για την ανάπτυξη λογισμικού. Συνήθως περιέχει έναν επεξεργαστή πηγαίου κώδικα, εργαλεία σύνθεσης και απόσφαλματωσης του κώδικα.

Virtual Machine: Είναι μία προσομοίωση ενός υπολογιστικού συστήματος, βασίζεται σε αρχιτεκτονική υπολογιστών και λειτουργεί όπως τα φυσικά υπολογιστικά συστήματα.

ΕΙΣΑΓΩΓΗ

Οι γράφοι για την απεικόνιση δεδομένων υφίστανται εδώ και πολύ καιρό, ωστόσο σήμερα χρησιμοποιούνται ευρέως λόγω των νέων συνθηκών και αναγκών. Έχουμε μεταβεί σε τεράστιους όγκους δεδομένων της τάξης των TBs (Terabytes). Παραδείγματα αποτελούν δεδομένα καιρικών συνθηκών, δεδομένα κίνησης σε αυτοκινητοδρόμους, είτε πολύ δημοφιλή όπως είναι τα δεδομένα από τα μέσα κοινωνικής δικτύωσης. Έτσι, προέκυψε η ανάγκη της ανάλυσης αυτών των δεδομένων για την εξατομίκευση των προτιμήσεων για κάθε χρήστη στην περίπτωσή των μέσων κοινωνικής δικτύωσης ή την λήψη συμπερασμάτων που δεν είναι απαραίτητα φανερά άμεσα τα αποτέλεσμα τους.

Στην εργασία θα προσεγγίσουμε το θέμα της ανάλυσης γράφων με την χρήση διαφόρων εργαλείων και αλγορίθμων που θα μας βοηθήσουν να εξάγουμε κάποια συμπεράσματα αλλά και να οπτικοποιήσουμε τα δεδομένα μας σε διάφορες μορφές, αναλόγως της κατηγοριοποίησης τους. Συγκεκριμένα στο 2ο Κεφάλαιο παρουσιάζονται βασικοί αλγόριθμοι που χρησιμοποιούνται στην ανάλυση δεδομένων που αναπαρίστανται με γράφους. Στο 3ο Κεφάλαιο περιγράφονται βασικά εργαλεία λογισμικού που χρησιμοποιούνται ευρέως για την μοντελοποίηση και ανάλυση γράφων, ενώ στο επόμενο Κεφάλαιο περιγράφεται η διαδικασία εγκατάστασης τους. Στο 5ο Κεφάλαιο εφαρμόζονται οι παραπάνω αλγόριθμοι ανάλυσης με την βοήθεια του λογισμικού Apache Spark και άλλων εργαλείων λογισμικού σε συγκεκριμένες συλλογές δεδομένων που προέρχονται από το Facebook και το Twitter [17][18]. Τέλος κλείνουμε με τα συμπεράσματα της εργασίας.

1 Τι σημαίνει Graph Analytics?

Τα graph analytics, γνωστά και με τον όρο network analysis, είναι ένας καινούργιος τομέας που γνωρίζει πολύ μεγάλη ανάπτυξη και αφορά την ανάλυση μεγάλου όγκου δεδομένων. Αυτό που ώθησε την ανάπτυξη αυτή κατά ένα μεγάλο μέρος είναι η δυνατότητα της χρήσης των graph analytics για ανάλυση κοινωνικών δικτυών και την εύρεση παραγόντων επιρροής. Ιδιαίτερο ενδιαφέρον παρουσιάζουν οι εταιρείες μάρκετινγκ, όπου τους είναι χρήσιμο να αναγνωρίσουν ανθρώπους, οι οποίοι μπορούν να επηρεάσουν ομάδες ανθρώπων μέσα σε κοινότητες των κοινωνικών δικτύων, ώστε να χρησιμοποιηθούν για καμπάνιες προϊόντων ή υπηρεσιών και να αποτελέσουν μέρος τους. Φυσικά, η χρήση του Graph analytics δεν περιορίζεται μόνο εκεί, αλλά χρησιμοποιείται για τον εντοπισμό οικονομικών εγκλημάτων, βελτιστοποίηση διαδρομών των αεροπορικών γραμμών, αντιτρομοκρατία και άλλους τομείς που θα αναφερθούν σε παρακάτω κεφαλαίο. Οι κύριες προσεγγίσεις ανάλυσης γράφων, είναι η ανάλυση μονοπατιού (Path Analysis), η ανάλυση συνδεσιμότητας (Connectivity analysis), ανάλυση κοινοτήτων (Community analysis) και ανάλυση κεντρικότητας (Centrality analysis) [8] που θα δούμε στο Κεφάλαιο 2 αναλυτικότερα, για το πώς λειτουργούν και ποιος είναι ο σκοπός τους.

1.1 Graphs – γράφοι

Ο γράφος είναι γνωστός από τα μαθηματικά και είναι μια δισδιάστατη αναπαράσταση που δείχνει τη σχέση μεταξύ δύο στοιχείων ή σύνολο στοιχείων τα οποία είναι συνδεδεμένα μεταξύ τους με δεσμούς. Τα διασυνδεμένα στοιχεία καλούνται κορυφές (vertices) και οι δεσμοί που συνδέουν τις κορυφές ονομάζονται ακμές (edges). Οι ακμές των γράφων μπορούν να είναι κατευθυνόμενες αλλά και μη-κατευθυνόμενες αλλά και να παρουσιάζουν «βάρος». Το βάρος βρίσκει εφαρμογή σε διάφορους αλγορίθμους όπως για παράδειγμα ο αλγόριθμος PageRank για ανάλυση κεντρικότητας. Ιδιότητες όπως η συνεκτικότητα δεν λείπουν από τον κλάδο Graph analytics και μάλιστα είναι σημαντικές στον καθορισμό και προσδιορισμό αδύναμων κορυφών. Μπορούν, για παράδειγμα, να χρησιμοποιηθούν σε ένα δίκτυο ηλεκτροδότησης, για την εύρεση σταθμών ηλεκτροδότησης οι οποίοι δεν έχουν εναλλακτικό τρόπο διασύνδεσης με τον κεντρικό

σταθμό. Δύο επιπλέον χαρακτηριστικά που παίζουν σημαντικό ρόλο είναι ο εσωτερικός και ο εξωτερικός βαθμός ενός κόμβου, όπου ο εσωτερικός βαθμός είναι ο αριθμός των ακμών που καταλήγουν στον κόμβο, ενώ ο εξωτερικός βαθμός είναι ο αριθμός των ακμών που αναχωρούν από τον κόμβο.

1.2 Big Data

Η δημιουργία γράφων με αριθμό κορυφών κάτω από δέκα χιλιάδες κόμβους δεν αποτελεί ιδιαίτερη πρόκληση όσον αφορά στην επεξεργασία και την ανάλυσή τους αλλά και στην οπτικοποιήση τους. Εδώ είναι το σημείο όπου εισέρχεται ο τομέας Big Data ο οποίος μας δίνει μια «θάλασσα» δεδομένων, τα οποία προκύπτουν από μεγάλες εταιρείες αναλύσεων και όχι μόνο. Με τον όρο Big Data έχει καθοριστεί να ονομάζονται οι συστοιχίες δεδομένων που είναι τόσο μεγάλες αλλά και συνάμα πολύπλοκες, που αποθηκεύονται παγκοσμίως σε μεγάλα data centers. Σε μερικές περιπτώσεις ανήκουν σε μεγάλες τεχνολογικές εταιρείες (Google, Facebook, Amazon). Τα εν λόγω δεδομένα δεν είναι εύκολο να επεξεργαστούν από παραδοσιακά προγράμματα επεξεργασίας δεδομένων ή από ένα απλό υπολογιστικό σύστημα, διότι απαιτούν επεξεργαστική ισχύ που δεν είναι εύκολο να έχει ο καθένας σπίτι του. Μερικές προκλήσεις σχετικά με τα Big Data είναι η αποθήκευση τους, η οπτικοποιήση τους, η επεξεργασία τους αλλά και η ανάλυση τους [12][15].

Πλέον ο όρος Big Data χρησιμοποιείται για τον προσδιορισμό ενεργειών που γίνονται σχετικά με αυτά όπως είναι η ανάλυση συμπεριφοράς του χρήστη, ένας τομέας που συνήθως αναλύονται τα δεδομένα για την πρόβλεψη της ανθρώπινης συμπεριφοράς από τα Νευρωσικά δίκτυα (Neural networks) αλλά και για την βελτίωση της τεχνητής νοημοσύνης, όπως και σε predictive analytics που καταλήγουν να λειτουργούν σε graph analytics ή καλύτερα network analytics. Σε αυτές τις περιπτώσεις καταλήγουμε σε εκατομμύρια κόμβους (nodes), δισεκατομμύρια ακμές (edges) και εφόσον ξεφεύγουν τόσο πολύ οι αριθμοί, πλέον έχουμε clusters υπολογιστικών συστημάτων όπου επεξεργάζονται τα δεδομένα αναλαμβάνοντάς ένα μέρος τους το κάθε σύστημα.

2 Αλγόριθμοι Graph analytics

Στο κεφαλαίο αυτό θα δούμε βασικούς αλγορίθμους ανάλυσης γράφων, τον τρόπο λειτουργίας τους και τον λόγο που δημιουργήθηκαν. Ένας από τους πιο γνωστούς, είναι ο αλγόριθμος της Google, το PageRank και χρησιμοποιείται για τον καθορισμό της κατάταξης των αποτελεσμάτων στην μηχανή αναζήτησής της. Ο Breadth-first search (BFS) είναι και αυτός ένας αλγόριθμος αναζήτησης κυρίως μονοπατιών και δίνει τα πιο σύντομα μονοπάτια μεταξύ κόμβων.

2.1 Google's PageRank

Ο αλγόριθμος αυτός δημιουργήθηκε από την Google για να δημιουργεί την κατάταξη των ιστοσελίδων αξιολογώντας την ποιότητα κάθε ιστοσελίδας με κλίμακα από το μηδέν μέχρι το δέκα. Χρησιμοποιεί τις διασυνδέσεις της κάθε ιστοσελίδας, δηλαδή τους συνδέσμους που οδηγούν σε αυτές και αυτούς που οδηγούν σε εξωτερικές ιστοσελίδες, αλλά και όσες την αναφέρουν (backlinks). Όταν ένα backlink προέρχεται από σημαντική πηγή όπως το Facebook τότε η αξία/ποιότητα του αναφερθέντος ιστότοπου αυξάνεται περισσότερο από ότι θα αυξανόταν αν αναφερόταν από μία λιγότερο σημαντική πηγή. Ο αλγόριθμος PageRank χρησιμοποιείται για την αξιολόγηση της σημαντικότητας των κόμβων που μετέχουν στον γράφο. Εδώ συμμετέχει ο εξωτερικός και εσωτερικός βαθμός ενός κόμβου. Όταν δημιουργεί το γράφημα ο αλγόριθμός αντιλαμβάνεται τις ακμές ως κατευθυνόμενες, δηλαδή διακρίνει την πηγή και τον προορισμό. Στο παράδειγμα της Εικόνας 2.1 φαίνεται ο εσωτερικός βαθμός και εξωτερικός αλλά και το PageRank, σε περιβάλλον Apache Spark [13].

```
scala> pr_rt.inDegrees.limit(5).show()
+------+--------+
|    id|inDegree|
+------+--------+
|174116|       3|
|229650|       1|
|414827|       1|
|249908|       1|
| 39917|      11|
+------+--------+
```

Εικόνα 2.1 Εσωτερικός Βαθμός

```
scala> pr_rt.outDegrees.limit(5).show()
+------+---------+
|    id|outDegree|
+------+---------+
|  4821|      133|
| 44446|        3|
| 68285|      850|
| 58744|       61|
|104454|        1|
+------+---------+
```

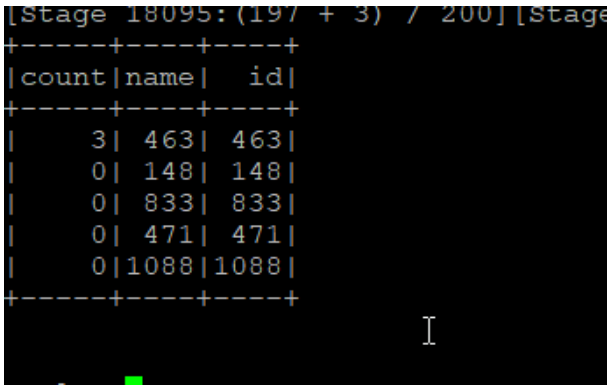
Εικόνα 2.2 Εξωτερικός Βαθμός

```
scala> result_pr_rt.limit(5).show()
+------+------+------------------+
|  name|    id|          pagerank|
+------+------+------------------+
|181190|181190| 49.05998138583171|
| 52204| 52204|19.813176265061852|
|254305|254305|17.947373640236698|
|342688|342688| 17.72456704950613|
|116484|116484|17.693497428877887|
+------+------+------------------+
```

Εικόνα 2.3 PageRank αποτελέσματα

2.2 Triangle Counting

Ο αλγόριθμος "triangle count" είναι ένας αλγόριθμος που εφαρμόζεται σε Big Data. Στον τομέα graph analytics υπολογίζει τον αριθμό των τριγώνων που περνούν από κάθε κόμβο ανεξάρτητα από την κατεύθυνση των ακμών. Αυτό είναι ιδιαίτερα χρήσιμο στην ανάλυση κοινωνικών δικτύων για δύο λόγους: α) καταδεικνύει την πιθανότητα αν δυο άτομα που έχουν κοινό φίλο, να είναι και μεταξύ τους φίλοι, β) δυνητικά πλεονεκτική θέση του ατόμου που συνδέεται με διαφορετικές κοινότητες και μπορεί να χρησιμοποιηθεί ως γέφυρα για την μεταφορά ιδεών από τη μία κοινότητα στην άλλη. Εδώ θα παρουσιαστεί μια υλοποίηση που χρησιμοποιείται με το Apache Spark και συμπεριλαμβάνεται μέσα σε ένα πρόσθετο το GraphFrame, που θα αναλυθεί σε επόμενα κεφάλαια [16].



Εικόνα 2.4 Στιγμιότυπο από τον αλγόριθμο Triangle count

Στην Εικόνα 2.4, βλέπουμε τον αριθμό των τριγώνων όπου διέρχονται από τους κόμβους “463”, “148”, “833”, “471” και “1088”. Παρατηρείται ότι στο συγκεκριμένο στιγμιότυπο μόνο από τον κόμβο “463” διέρχονται 3 τρίγωνα.

2.3 Shortest Paths

Ο αλγόριθμος "shortest paths" είναι ένας αλγόριθμος ανάλυσης μονοπατιού (Path analysis), και επομίζει τα συντομότερα μονοπάτια από έναν κόμβο προς όλους τους υπόλοιπους του γράφος ή από λίστα κόμβων προς όλους τους υπόλοιπους κόμβους του γραφήματος, λαμβάνοντας υπόψη την κατεύθυνση της κάθε ακμής [11], όπως παρουσιάζεται στην Εικόνα 2.5.

```
scala> short_p.sort(desc("pagerank")).limit(10).show()
+------+------+------------------+--------------------+
|  name|    id|          pagerank|           distances|
+------+------+------------------+--------------------+
|181190|181190| 49.05998138583169|       [181190 -> 0]|
| 52204| 52204| 19.81317626506184|[52204 -> 0, 1164...|
|254305|254305|17.947373640236687|[254305 -> 0, 260...|
|342688|342688| 17.72456704950613|       [342688 -> 0]|
|116484|116484| 17.69349742887788|       [116484 -> 0]|
| 10269| 10269| 16.15633878641779|[10269 -> 0, 2614...|
|260488|260488| 16.12134915286262|[260488 -> 0, 254...|
| 96067| 96067|15.520721277894047|[90979 -> 7, 3646...|
|261489|261489|14.596562887851745|[261489 -> 0, 102...|
|287256|287256| 14.52171767233725|       [287256 -> 0]|
+------+------+------------------+--------------------+
```

Εικόνα 2.5 Παράδειγμα εκτέλεσης αλγορίθμου Shortest Paths

Η στήλη "distances" δείχνει την απόσταση για κάθε κόμβο που δόθηκε ως είσοδος στον αλγόριθμο από τους υπόλοιπους κόμβους του γράφου. Το αποτέλεσμα είναι σε μορφή [node->distance, node1->distance1,…,noden->distancen]

2.4 Breadth-first Search (BFS)

Ο BFS [20] είναι αλγόριθμος αναζήτησης ενός δυαδικού δένδρου ή ενός γράφου και πραγματοποιεί αναζήτηση στο ίδιο επίπεδο κατά πλάτος, δηλαδή εξερευνά όλους τους γειτονικούς κόμβους στους οποίους μπορεί να μεταβεί απευθείας πριν προχωρήσει στο επόμενο. Η υλοποίηση του αλγορίθμου που εμφανίζεται στην εργασία βρίσκει το πιο κοντινό μονοπάτι από ένα κόμβο ή ζευγάρι κόμβων προς άλλο κόμβο ή ζευγάρι κόμβων. Το αποτέλεσμα που παρουσιάζεται στην Εικόνα 2.6, υλοποιείται σε περιβάλλον Apache Spark με την βοήθεια ενός πρόσθετου του GraphFrame και είναι αλγόριθμος κατηγορίας ανάλυσης μονοπατιού.

```
scala> pr_rt.bfs.fromExpr(col("name") ==="181190").toExpr(col("pagerank") >= 12.264750367638758).run()
res30: org.apache.spark.sql.DataFrame = [from: struct<name: string, id: int ... 1 more field>, to: struct<name: str
ng, id: int ... 1 more field>]

scala> res30.show()
+--------------------+--------------------+
|                from|                  to|
+--------------------+--------------------+
|[181190, 181190, ...|[181190, 181190, ...|
+--------------------+--------------------+
```

Εικόνα 2.6 Παράδειγμα εκτέλεσης Αλγορίθμους BFS σε γράφο

2.5 Label Propagation Algorithm (LPA)

Ο Label Propagation Algorithm [22] είναι αλγόριθμος εύρεσης κοινοτήτων σε κόμβους σε έναν γράφο. Η εύρεση κοινοτήτων σε ένα γράφο που αναπαριστά κάποιο από τα μέσα κοινωνικής δικτύωσης δείχνει ομάδες ανθρώπων που αλληλεπιδρούν μεταξύ τους και έχουν κοινά «ενδιαφέροντα». Ο αλγόριθμος αρχικά, τοποθετεί στην δική του κοινότητα (community) κάθε κόμβο και στη συνέχεια, σε κάθε superstep στέλνει την κοινοτική του υπαγωγή σε όλους τους γειτονικούς κόμβους. Στην συνέχεια, οι κόμβοι ανανεώνουν την κατάσταση τους σε λειτουργία κοινοτικής υπαγωγής εισερχόμενων μηνυμάτων από γειτονικούς κόμβους. Στην Εικόνα 2.7 φαίνεται το αποτέλεσμα της εκτέλεσης αλγόριθμου LPA σε περιβάλλον Apache Spark.

```
println("===Communities: Label Propagation Algorithm===")
val community_rt = pr_rt.labelPropagation.maxIter(10).run
community_rt.sort(desc("pagerank")).show()

// Exiting paste mode, now interpreting.

===Communities: Label Propagation Algorithm===
+------+------+------------------+------+
|  name|    id|          pagerank| label|
+------+------+------------------+------+
|181190|181190| 49.059981385831 71|181190|
| 52204| 52204|19.813176265061852| 52204|
|254305|254305|17.947373640236698|254305|
|342688|342688| 17.724567049506 13|342688|
|116484|116484|17.693497428877887| 32972|
| 10269| 10269| 16.156338786417 79| 10269|
|260488|260488|16.121349152862628|  6971|
| 96067| 96067| 15.520721277894 05| 96067|
|261489|261489|14.596562887851746|261489|
|287256|287256|14.521717672337253|287256|
| 33719| 33719|13.762435865272657| 32972|
| 66730| 66730|13.524923598353809|  1988|
|183312|183312|13.397650784803755|183312|
| 22759| 22759|13.326390327353224| 22759|
| 90979| 90979|13.128211344471886|266400|
|101502|101502|12.764014103875455|101502|
|364689|364689|12.510955442446281|364689|
| 81405| 81405|12.490069487220289| 81405|
| 45463| 45463|12.431400496976206|240677|
|229041|229041| 12.264750367638 76|229041|
+------+------+------------------+------+
```

Εικόνα 2.7 Παράδειγμα εκτέλεσης Αλγορίθμου LPA σε υπογράφο

2.6 Connected Components

Από την θεωρία των γράφων [14], μία συνδεδεμένη συνιστώσα σε ένα μη κατευθυνόμενο γράφο είναι ένας υπο-γράφος στον οποίο, ανεξαρτήτως που οι κορυφές του δεν συνδέονται με κανέναν άλλον επιπλέον κόμβο στο υπεργράφημα. Στη περίπτωση του Apache Spark με την χρήση του GraphFrame, ο αλγόριθμος μας επιστρέφει/υπολογίζει σε ποια συνιστώσα είναι μέλος ο κάθε κόμβος. Στην Εικόνα 2.5 φαίνονται τα αποτελέσματα μιας εκτέλεσης του αλγορίθμου, όπου ο κάθε κόμβος αντιστοιχίζεται σε ένα “component ID”. Αν ο γράφος είναι συνδεδεμένος (αν κάθε ζεύγος κορυφών, είναι μεταξύ τους συνδεδεμένοι) τότε όλοι οι κόμβοι αντιστοιχούν στο ίδιο “component ID”.

```
scala> val result = g1.connectedComponents.run()
result: org.apache.spark.sql.DataFrame = [name: int, id: int ...

scala> result.limit(10).show()
+----+----+---------+
|name|  id|component|
+----+----+---------+
| 148| 148|        0|
| 463| 463|        0|
| 471| 471|        0|
| 496| 496|        0|
|1088|1088|        0|
|1238|1238|        0|
|1342|1342|        0|
|1580|1580|        0|
|1591|1591|        0|
|1645|1645|        0|
+----+----+---------+
```

Εικόνα 2.8 Παράδειγμα εκτέλεσης αλγορίθμου Connected Components σε περιβάλλον Spark

Στο ανωτέρω παράδειγμα, ο αλγόριθμος καταχώρησε όλους τους κόμβους ότι ανήκουν στο "component 0 " και μπορεί να επιβεβαιωθεί με την εκτέλεση του κώδικα της Εικόνας 2.9.

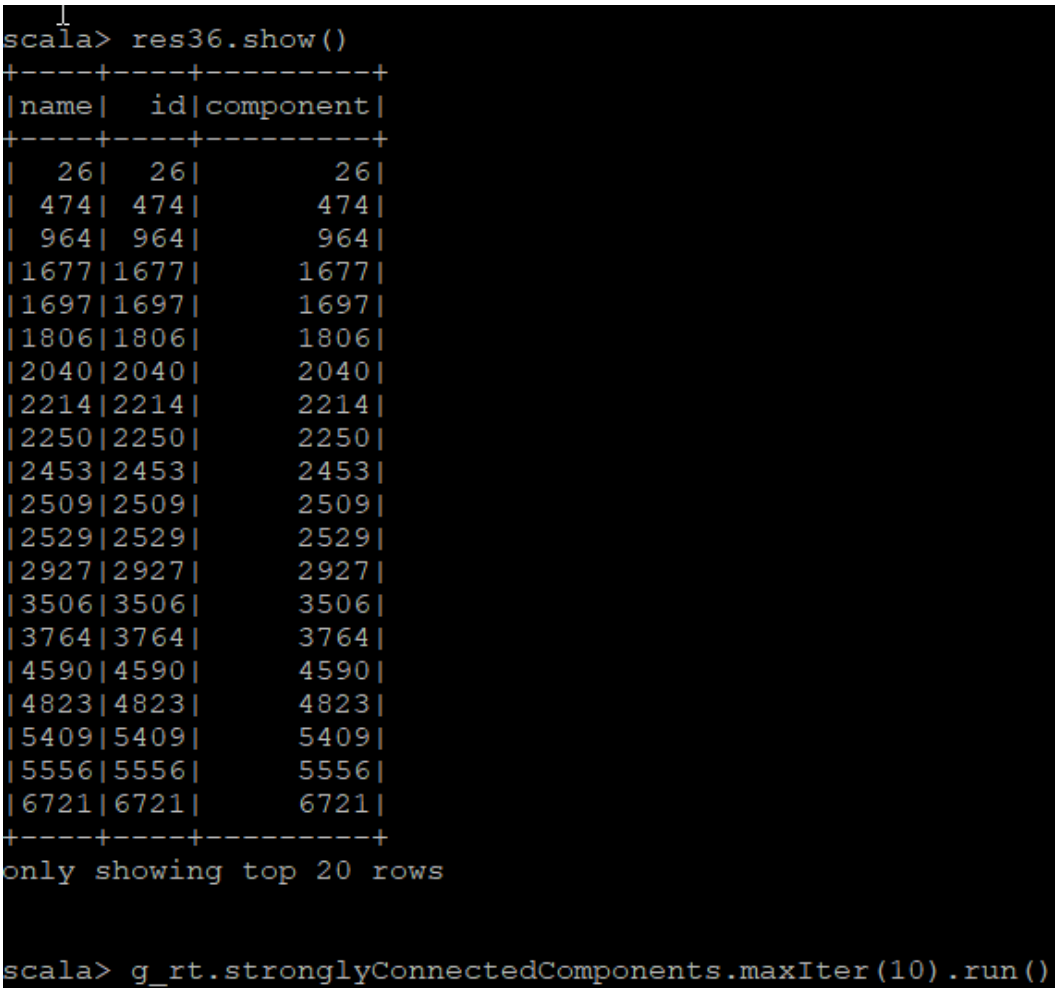
```
scala> result.limit(10).filter("component >0 ").show()
+----+---+---------+
|name| id|component|
+----+---+---------+
+----+---+---------+
```

Εικόνα 2.9 Εύρεση άλλου Component εκτός από 0

2.7 Strongly Connected Components

Από τη θεωρία των γράφων [14] είναι γνωστό ότι, ένας κατευθυνόμενος γράφος είναι ισχυρά συνδεδεμένος εάν για κάθε ζεύγος κορυφών υπάρχει διαδρομή και προς τις δύο κατευθύνσεις. Στο GraphFrame στο περιβάλλον του Apache Spark ο αλγόριθμος υπολογίζει τα ισχυρά

συνδεδεμένα στοιχεία του κάθε κόμβου και επιστρέφει ένα γράφο, με κάθε κόμβο αντιστοιχισμένο στο ισχυρά συνδεδεμένη συνιστώσα του, όπως παρουσιάζεται στην Εικόνα 2.10.



Εικόνα 2.10 Παράδειγμα εκτέλεσης αλγορίθμους Strongly Connected Component σε περιβάλλον Apache Spark.

3 Εργαλεία μοντελοποίησης και ανάλυσης γράφων

3.1 Apache Hadoop

Η βιβλιοθήκη λογισμικού του Apache Hadoop είναι ένα "framework" που επιτρέπει την διαμοιρασμένη επεξεργασία μεγάλων συνόλων δεδομένων σε ομάδες υπολογιστικών συστημάτων (clusters), χρησιμοποιώντας απλά προγραμματιστικά μοντέλα. Μπορεί να λειτουργήσει με ένα σύστημα μέχρι και με χιλιάδες ταυτόχρονα, προσφέροντας στο καθένα υπολογιστική ισχύ και αποθηκευτικό χώρο τοπικά. Δεν βασίζεται σε υλισμικό (hardware) για να προσφέρει υψηλής απόδοσης και διαθεσιμότητας υπηρεσία σε επίπεδο εφαρμογών, αλλά η βιβλιοθήκη από μόνη της εντοπίζει τυχόν αστάθειες και αποτυχίες και τις χειρίζεται αναλόγως. Το Apache Hadoop περιλαμβάνει το "Hadoop Common" που περιέχει εργαλεία, ώστε να υποστηρίζει τα υπόλοιπα Hadoop πρόσθετα (HDFS, Hadoop YARN και MapReduce). Το "Hadoop Distributed File System (HDFS)" είναι ένα σύστημα αρχείων υψηλής απόδοσης πρόσβασης σε δεδομένα εφαρμογών [3]. Το "Hadoop YARN " [4] είναι ένα framework για τον προγραμματισμό εργασιών και τη διαχείριση πόρων του cluster. Τέλος, διαθέτει ένα προγραμματιστικό μοντέλο το Hadoop MapReduce που χρησιμοποιεί το "YARN" για παράλληλη επεξεργασία μεγάλων συνόλων δεδομένων. Σε αυτή την εργασία χρησιμοποιούμε "Hadoop Common"και "HDFS".

3.1.1 Τεχνικά χαρακτηρίστηκα και Δομή

Το Apache Hadoop είναι κυρίως αναπτυγμένο σε γλώσσα προγραμματισμού Java, ενώ στα αρχεία παραμετροποίησης του χρησιμοποιεί XML. Είναι cross-platform λογισμικό, που σημαίνει ότι είναι δυνατή η εγκατάσταση του σε Windows, MacOS, Linux, BSD. Το Hadoop μπορεί να εγκατασταθεί και να χρησιμοποιηθεί από μια πλειάδα προγραμμάτων όπως είναι το Apache Spark, Apache Hive, Apache Pig, Cloudera Impala και άλλα, τα οποία χρησιμοποιούν κυρίως το σύστημα αρχείων του, το "HDFS". Για την εκτέλεση κώδικα ο χρήστης έχει τη δυνατότητα να χρησιμοποιήσει Java ή Python κυρίως για να εκτελέσει MapReduce

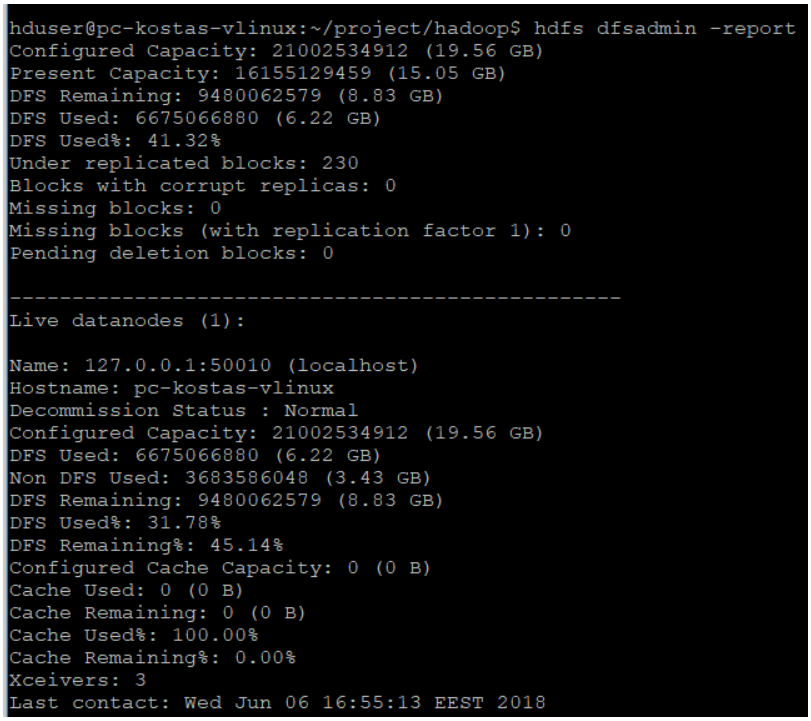
αλγορίθμους. Μερικά στοιχεία του Apache Hadoop εμπνεύστηκαν από εργασίες της Google πάνω στο δικό της MapReduce αλγόριθμο και στο δικό τους σύστημα αρχείων της που ονομάζεται "Google File System" από όπου προήλθε το "HDFS".

3.1.2 HDFS (High Demand File System)

Το HDFS [2] είναι το σύστημα αρχείων που χρησιμοποιείται για τη διαχείριση και αποθήκευση των δεδομένων που θέλουμε να επεξεργαστούμε σε παράλληλα συστήματα και είναι ανεξάρτητο από το λειτουργικό σύστημα, το οποίο είναι προσβάσιμο από κάθε σύστημα-μέλος του cluster. Το "HDFS" αποτελείται από το "NameNode" όπου διαχειρίζεται τα μεταδεδομένα του συστήματος αρχείων και από το "DataNode" που αποθηκεύει τα δεδομένα. Επιπλέον, υποστηρίζει εντολές που θυμίζουν εντολές unix συστημάτων που χρησιμοποιούνται για τη διαχείριση των "NameNode" και "DataNode", τα οποία έχουν ενσωματωμένους "web servers" έτσι ώστε να μπορεί να ελεγχθεί η κατάσταση του cluster οποιαδήποτε στιγμή. Το "HDFS" παρέχει διαχείριση δικαιωμάτων αρχείων και επαλήθευσης, λειτουργία ασφαλείας σε περίπτωση σφάλματος για να μπορέσει ο διαχειριστής να πραγματοποιήσει συντήρηση. Παρέχει επίσης το γνωστό εργαλείο από Linux το "fsck" που χρησιμοποιείται για την εξακρίβωση της υγείας του συστήματος αρχείων, λειτουργία "Balancer" για να εξισορροπεί τα δεδομένα όταν μοιράζονται μεταξύ δύο "DataNodes" ανόμοια και λειτουργίες διάσωσης και επαναφοράς σε περίπτωση αναβάθμισης που δημιούργησε πρόβλημα. Η βασική εντολή διαχειριστή που διαθέτει είναι το "bin/hdfs dfsadmin και δέχεται παραμέτρους "-report, -safemode, -finalizeUpgrade, -refreshNodes, -printTopology". Στις Εικόνες 3.1 και 3.2 εμφανίζονται τα αποτελέσματα από την εκτέλεση της εντολής με κάποιες παραμέτρους.

```
hduser@pc-kostas-vlinux:~/project/hadoop$ hdfs dfsadmin -printTopology
Rack: /default-rack
   127.0.0.1:50010 (localhost)
```

Εικόνα 3.1 Παράδειγμα εκτέλεσης dsfadmin με παράμετρο -printTopology



Εικόνα 3.2 Παράδειγμα εκτέλεσης εντολής dsfadmin με παράμετρο -report

Με την παράμετρο "-safemode", μπορούμε χειροκίνητα να μπούμε και να βγούμε από την λειτουργία «ασφαλής λειτουργία». Η παράμετρος "-finalizeUpgrade" διεκπεραιώνει την αναβάθμιση που έχουμε κάνει στο σύστημα και η "-refreshNodes" κάνει ανανέωση στα "DataNodes" και "NameNodes" σύμφωνα με το αρχείο παραμετροποίησης [2].

Σε αυτή την εργασία χρησιμοποιείται το HDFS για αποθήκευση δεδομένων που προκύπτουν από την ανάλυση που γίνεται πάνω στους γράφους αλλά και για αποθήκευση αρχείων CSV για μεταγενέστερη επεξεργασία και εισαγωγή στο λογισμικό οπτικοποιήση. Επιπλέον, χρησιμοποιείται για αποθήκευση αρχείων καταγραφής συμβάντων από το Apache Spark και για την δημιουργία προσωρινών αρχείων εκτέλεσης ορισμένων αλγορίθμων, όπου απαιτείται η δημιουργία ενός προσωρινής αποθήκευσης καταλόγου.

3.1.3 MapReduce

Το MapReduce [1] είναι ένα "framework", το οποίο επιτρέπει το να γράφονται εφαρμογές που επεξεργάζονται τεράστιες ποσότητες δεδομένων της τάξης των Terabytes, σε παράλληλα υπολογιστικά σύστημα (clusters) με αξιοπιστία και ανοχή σε σφάλματα. Πρόκειται για προγραμματιστικό μοντέλο, σύμφωνα με πολλούς, του χώρου του Big Data. Ο αλγόριθμος που χρησιμοποιεί έχει ως σκοπό να φιλτράρει τα δεδομένα, να τα ταξινομήσει (map) και να τα μειώσει (reduce) κατά σύνολο, όπως στην περίπτωση καταμέτρησης μαθητών σε ένα λύκειο ανά τάξη. Το μοντέλο αυτό, είναι εμπνευσμένο από τις συναρτήσεις "map" και "reduce", που χρησιμοποιούταν στο παρελθόν στο «συναρτησιακό προγραμματισμό» (functional programming) αν και δεν παραμένει ίδια η μορφή του από τότε. Όπως ήδη έχει αναφερθεί, το MapReduce ήταν ¨πατέντα¨ της Google.

Στο Apache Hadoop μια "MapReduce" εργασία συνήθως «σπάει» τα εισερχόμενα δεδομένα σε ανεξάρτητα σύνολα τα οποία επεξεργάζονται από τα "map tasks" με απολύτως παράλληλο τρόπο. Μετά την ταξινόμηση που έχει συμβεί στο "map task", τα εξαγόμενα δεδομένα, γίνονται είσοδος στα "reduce tasks". Το MapReduce χρησιμοποιεί κυρίως Java για την εκτέλεση προγραμμάτων "map-reduce", ωστόσο, μέσω της βιβλιοθήκης-εργαλείου "Hadoop Streaming " που διαθέτει, δίνεται η δυνατότητα να εκτελεστούν εργασίες mapper-reducer γραμμένες μέχρι και σε "shell scripts". Στην εργασία δεν γίνεται άμεση η χρήση του MapReduce από το Apache Hadoop αλλά εκτελείται από το Apache Spark διότι είναι μέρος της δομής του, όπως θα εξηγήσουμε στο Κεφάλαιο 4.3. Στην Εικόνα 3.3 βλέπουμε την μορφή μίας εργασίας MapReduce, η οποία δέχεται ως είσοδο ένα ζευγάρι τιμών της μορφής <κλειδί, τιμή> και η έξοδος της είναι <κλειδί3,τιμή3> όπου είναι αποτέλεσμα του συνδυασμού των εργασιών "Map" και "Reduce" [1].

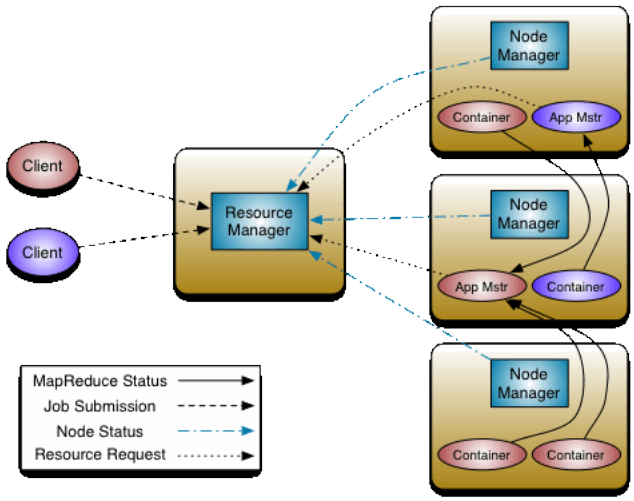
Input and Output types of a MapReduce job:

(input) <k1, v1> -> **map** -> <k2, v2> -> **combine** -> <k2, v2> -> **reduce** -> <k3, v3> (output)

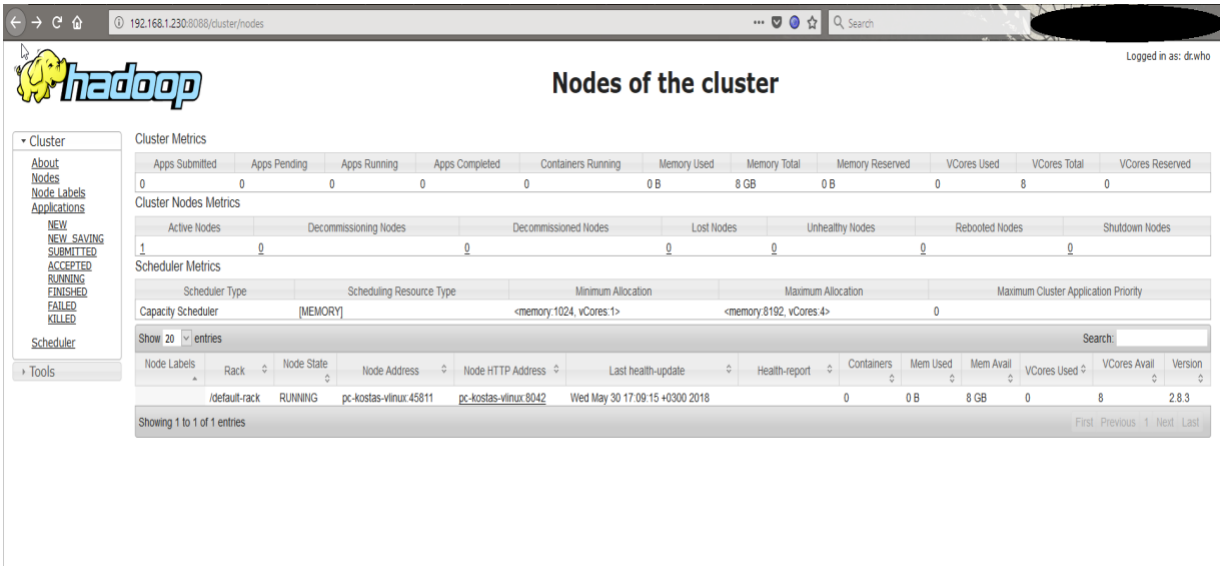
Εικόνα 3.3 Μορφή μιας εργασίας MapReduce. (Πηγή: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html)

3.1.4 Cluster – Hadoop YARN

Όπως ήδη έχει αναφερθεί το Hadoop YARN [4], είναι υπεύθυνο για την διαχείριση πόρων ενός cluster. αλλά είναι υπεύθυνο για τον προγραμματισμό και παρακολούθηση των εργασιών που αναθέτει σε κάθε σύστημα με παγκόσμιο σύστημα πόρων. Ουσιαστικά, το "YARN" λαμβάνει μια εργασία ή μια σειρά εργασιών και είναι υπεύθυνο να κάνει κατανομή πόρων και να δημιουργήσει την σειρά εκτέλεσης εργασιών με την βοήθεια του "ResourceManager" που είναι υπεύθυνο για την κατανομή των πόρων. Το "NodeManager" είναι υπεύθυνο ανά σύστημα για την παρακολούθηση των πόρων (cpu, ram, disk, network) και για την δημιουργία μίας αναφοράς που στέλνει στο "ResourceManager" για να κάνει σωστή κατανομή αυτών. Το "ResourceManager" αποτελείται από δυο βασικά μέρη, τον "Scheduler" και το "ApplicationManager". Ο "Scheduler" είναι υπεύθυνος για την κατανομή των πόρων σε διάφορες εφαρμογές που βρίσκονται σε εκτέλεση. Δεν είναι υπεύθυνος για την παρακολούθηση και τον εντοπισμό των εργασιών, ούτε προσφέρει εγγύηση ότι θα ανακινηθούν οι αποτυχημένες διεργασίες λόγω αστοχίας της εφαρμογής ή λόγω σφάλματος υλισμικού. Εκτελεί προγραμματισμένες εργασίες βάσει των απαιτήσεων σε πόρους της εφαρμογής, οπού υπαγορεύονται από το πλαίσιο της κατά την εκτέλεση της. Από την άλλη, το "ApplicationManager" είναι υπεύθυνο για την αποδοχή εργασίας που του υποβλήθηκε, την διαπραγμάτευση του πρώτου πλαισίου πόρων. Για κάθε εφαρμογή όπου εισέρχεται, το "ApplicationManager" είναι υπεύθυνο να διαπραγματευτεί τους κατάλληλους πόρους από το "Scheduler", ώστε να μπορεί να ανιχνεύσει την κατάστασή τους και να παρακολουθεί την πρόοδο της εκτέλεσης. Να σημειωθεί πως εδώ το "YARN" δεν χρησιμοποιήθηκε, διότι δεν είχαμε πολλά συστήματα, ώστε να υπάρχει cluster υπολογιστικών συστημάτων. Στο διάγραμμα που ακολουθεί βλέπουμε την αρχιτεκτονική του "YARN".



Εικόνα 3.4 Αρχιτεκτονική YARN (Πηγή:http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html)



Εικόνα 3.5 Περιβάλλον παρακολούθησης-επίβλεψης YARN

3.2 Apache Spark

Το Apache Spark [6] είναι γενικού σκοπού εργαλείο για υπολογιστικά συστήματα cluster. Παρέχει πρόσβαση σε υψηλού επιπέδου APIs (application programming interfaces) σε Java, Scala, Python και R και υποστηρίζει πληθώρα βιβλιοθηκών σε καθένα από αυτά. Επιπλέον, υποστηρίζει την Spark SQL για την δόμηση SQL δεδομένων και επεξεργασία αυτών με την βοήθεια των βιβλιοθηκών, "MLib" για "Machine Learning", "GraphX" για επεξεργασία γράφων. Όπως και το Apache Hadoop, μπορεί να λειτουργήσει, είτε σε cluster ,"Standalone setup" όπως αναφέρεται στις προδιαγραφές του. Συνεργάζεται με πληθώρα εργαλείων και διαθέτει πολλά πρόσθετα που η ενσωμάτωση τους είναι άμεση και εύκολη. Διαθέτει το "Spark YARN" που εποπτεύει τους πόρους του "cluster" και τους διαμοιράζει όπως ακριβώς λειτουργεί και στο Apache Hadoop. Στην περίπτωση του "Standalone setup" χρησιμοποιεί "FIFO" (First In-First Out) για την διεκπεραίωση των εργασιών. Το Apache Spark είναι σχετικά πρόσφατο λογισμικό, ξεκίνησε η ανάπτυξή του το 2009 από την Matei Zaharia στο UC Berkley's AMPLab και έγινε ανοιχτού κώδικα λογισμικό το 2010 υπό BSD άδεια χρήσης. Το 2013 έγινε δωρεά στο Apache Software Foundation και η άδεια χρήσης του άλλαξε το 2014 σε "Apache 2.0" και έγινε πρωτεύον έργο της Apache.

3.2.1 Τεχνικά χαρακτηριστικά και δομή

Το Spark έχει υλοποιηθεί σε τέσσερις βασικές γλώσσες προγραμματισμού τις Java, Scala, Python, R και χρησιμοποιεί και XML σε κάποια αρχεία παραμετροποίησης του. Είναι λογισμικό ανοιχτού κώδικα και είναι "cross-platform", δηλαδή η εγκατάσταση του είναι εφικτή σε λειτουργικό σύστημα Microsoft Windows, macOS, Linux. Υποστηρίζει πρόσθετα για την επέκταση των λειτουργιών του, τα οποία έχουν αναπτυχθεί από τρίτους προγραμματιστές όπως αυτά που έχουν χρησιμοποιηθεί στην παρούσα εργασία. Πρόκειται για το GraphFrame [9] και Spark-CSV, που θα τα δούμε πιο αναλυτικά στο Κεφάλαιο 5. Μπορεί να λειτουργήσει σε εγκατάσταση cluster αλλά και εγκατάσταση σε "Standalone setup" όπως χρησιμοποιείται στην παρούσα υλοποίηση.

Το κύριο στοιχείο που χρησιμοποιείται στο Apache Spark είναι το RDD (Resilient Distributed Dataset), το οποίο είναι ένα πολυδιάστατο σύνολο δεδομένων στο οποίο είναι επιτρέπονται μόνο η ανάγνωση και διατηρείται σε cluster για ανοχή στα σφάλματα. Στην έκδοση 1.x ήταν το κύριο API του προγράμματος. Αυτό άλλαξε με την έλευση της έκδοσης 2.x όπου χρησιμοποιείται ένα νέο API το "Dataset" για το οποίο μάλιστα υπάρχει ενθάρρυνση ως προς την χρήση του. Το "Dataset" είναι μια διανεμημένη συλλογή δεδομένων και έχει όλα τα οφέλη του RDD αλλά χρησιμοποίει την βελτιωμένη μηχανή εκτέλεσης που παρέχεται από το "Spark SQL". Με την έκδοση 2.x ήρθε και το DataFrame όπου είναι ένα οργανωμένο "Dataset" σε στήλες ονοματισμένες. Η λογική πίσω από αυτό είναι ίδια με έναν πίνακα σε μία βάση δεδομένων [7].

3.2.2 Πρόσθετα και επεκτάσεις

Το Spark δέχεται μεγάλο αριθμό επεκτάσεων τα οποία έχουν αναπτυχθεί από τρίτους προγραμματιστές και είναι ανοιχτού κώδικα. Είναι διαθέσιμα στον σύνδεσμο https://spark-packages.org/. Η προσθήκη τους στην εκτέλεση του προγράμματος γίνεται κατά την έναρξη της εκτέλεσης του περιβάλλοντος του Spark και η μορφή είναι όπως εμφανίζεται στην Εικόνα 3.6.

```
--packages graphframes:graphframes:0.5.0-spark2.1-s_2.11, com.databricks:spark-csv_2.11:1.5.0
```

Εικόνα 3.6 Προσθήκη επεκτάσεων, πρόσθετων στην εκτέλεση του Spark

Μέσα στον κώδικα της εφαρμογής θα πρέπει να κάνουμε εισαγωγή την αντίστοιχη βιβλιοθήκη της επέκτασης που μας ενδιαφέρει και στην περίπτωση του "GraphFrame" αυτό γίνεται όπως παρουσιάζεται στην Εικόνα 3.7.

```
1 import org.graphframes._
2 import org.apache.spark.sql._
3 import org.apache.spark.sql.functions._
4 import org.apache.spark.sql.types.{StructType, StructField, StringType, IntegerType};
5 import org.graphframes.GraphFrame
```

GraphFrame library

Εικόνα 3.7 Εισαγωγή βιβλιοθήκης πρόσθετου GraphFrame

3.2.3 MapReduce & HDFS

Αρχικά, το Spark δεν χρησιμοποιεί κάποιο συγκεκριμένο σύστημα αρχείων, μπορεί να γράψει και να διαβάσει δεδομένα από οποιοδήποτε σύστημα αρχείων συμπεριλαμβανομένου και του "HDFS" που παρουσιάστηκε στην Ενότητα 3.1.2. Είναι πολύ σύνηθες να χρησιμοποίει το "HDFS" για μόνιμη αποθήκευση αρχείων ύστερα από ανάλυση δεδομένων, για την εξαγωγή των τελικών αποτελεσμάτων. Η Εικόνα 3.8 δείχνει μια εξαγωγή δεδομένων σε αρχείο ".csv" το οποίο αποθηκεύεται στο "HDFS" του Hadoop.

```
pr_re.edges.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databricks.spark.csv").option("header","true").save("hdfs://localhost:54310/project/datacsv/graphs/
```

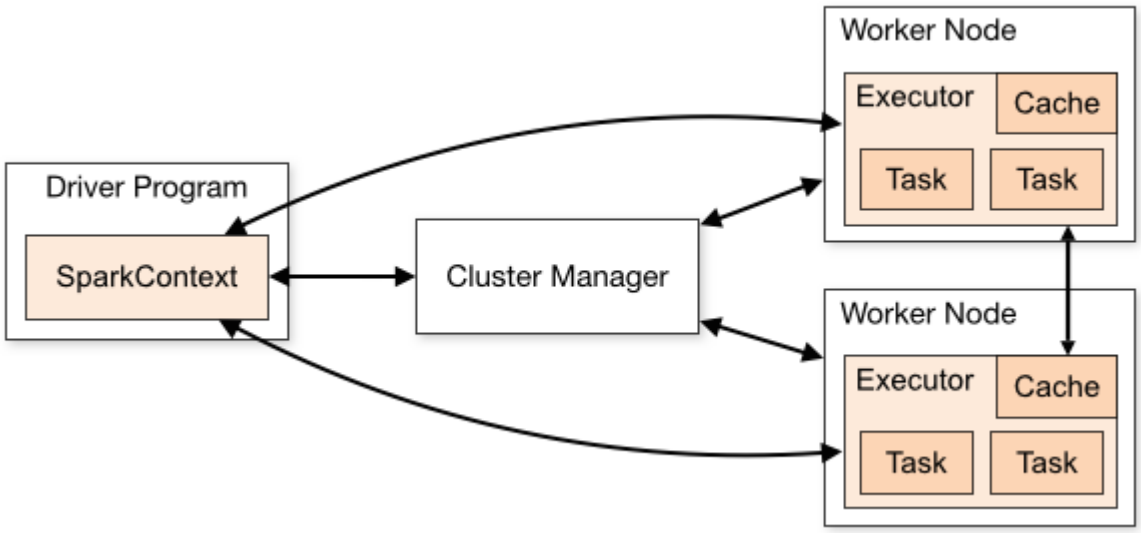
Εικόνα 3.8 Εγγραφή δεδομένων ανάλυσης σε αρχείο CSV

Το MapReduce στο Spark χρησιμοποιείται συνεχώς και πόσο μάλλον όταν έχουμε να κάνουμε με χρήση RDD, ωστόσο το συγκεκριμένο MapReduce δεν σχετίζεται με το Hadoop. Για την ακρίβεια το Spark ισχυρίζεται και μάλιστα ισχύει, βάσει διαφόρων δοκιμών που έχουν γίνει, ότι είναι μέχρι και εκατό φορές πιο γρήγορο από το Hadoop στον ίδιο όγκο δεδομένων, όσον αφορά την εκτέλεση της εφαρμογής σε μνήμη RAM και μέχρι και δέκα φορές πιο γρήγορο όσον αφορά την εκτέλεση σε σκληρό δίσκο [21].

3.2.4 Cluster – Spark Cluster mode

Στο Spark οι εφαρμογές εκτελούνται ως ανεξάρτητες συλλογές διεργασιών, συντονισμένες από το "SparkContext".Το SparkContext είναι ένα αντικείμενο που δημιουργείται από το κύριο πρόγραμμα και είναι «ο οδηγός του προγράμματος», δηλαδή καθοδηγεί το Spark για να προσπελάσει το cluster. Για να εκτελεστεί μια εφαρμογή σε cluster, το "SparkContext" μπορεί να συνδεθεί σε αρκετούς τύπους διαχειριστών clusters όπως είναι ο "Spark standalone cluster" , "Mesos" ή "Hadoop YARN", τα οποία διαχειρίζονται τους πόρους σε όλες τις εφαρμογές. Μόλις συνδεθεί ο "Spark Client ή Slave" στο διαχειριστή cluster, τότε λαμβάνει τους «εκτελεστές» (executors) που βρίσκονται στους κόμβους μέσα στο cluster, οι οποίοι

επεξεργάζονται τα δεδομένα της εφαρμογής στα οποία θα εκτελεστούν υπολογισμού και θα αποθηκευτούν τα δεδομένα της. Στην συνέχεια, στέλνεται ο κώδικας της εφαρμογής στους executors και τέλος το "SparkContext" στέλνει εργασίες (tasks) στους executors για την υλοποίηση τους [5]. Η διαδικασία εμφανίζεται στην Εικόνα 3.9.



Εικόνα 3.9 Διαδικασία εκτέλεσης εφαρμογής Apache Spark σε Cluster mode (Πηγή: https://spark.apache.org/docs/latest/cluster-overview.html)

3.3 Scala, Java, Python, R

Στο Spark υπάρχει η δυνατότητα ανάπτυξης εφαρμογών σε Scala, Java, Python, R, όπως ήδη έχει αναφερθεί σε προηγούμενο κεφαλαίο. Στην παρούσα εργασία έγινε ανάπτυξη κώδικα σε Scala διότι είναι η κύρια γλώσσα προγραμματισμού, για ανάπτυξη εφαρμογών σε Apache Spark. Τα οφέλη της επιλογής της Scala είναι αρκετά, αν και η Python είναι μια γλώσσα που κατά πολλούς που υπερτερεί στο τομέα των "Big Data". Στην περίπτωση του Apache Spark δεν ισχύει απόλυτα κάτι τέτοιο. Η Python αποτελεί σίγουρα μια σταθερή επιλογή, είναι αξιόπιστη, διότι είναι μία ώριμη γλώσσα και έχει πλούσια έγγραφη υποστήριξη για τις βιβλιοθήκες της, είναι δημοφιλής με αυξανόμενη τάση και περιλαμβάνει βιβλιοθήκες για "Deep Learning" και κυρίως έχει εύκολο κύκλο εκμάθησης. Από την άλλη πλευρά δεν είναι τόσο γρήγορη και κάποιες από τις βιβλιοθήκες της μπορεί να δημιουργήσουν πρόβλημα στην ενσωμάτωση τους.

Η Scala χρησιμοποιεί JVM όπως και βιβλιοθήκες από την Java, οπότε αν και θα φανεί περίεργη η μετάβαση σε αυτή, είναι λίγο πιο εύκολη, για κάποιον χρήστη. Διατίθενται προηγμένων δυνατοτήτων IDE και περιβάλλον δοκιμών (test units) για τον κώδικα, βελτιστοποιημένες φόρμουλες παραλληλισμού δεδομένων, προηγμένες δυνατότητες λήψης δεδομένων και επεξεργασίας σε πραγματικό χρόνο (Streaming Capabilities, Spark Streaming Library) και κυρίως είναι πολύ πιο γρήγορη από κάθε άλλη λόγω της γηγενής υποστήριξης από το Apache Spark [19]. Η εκτέλεση κώδικα σε Python πραγματοποιείται μέσω "pyspark" ενώ η Scala εκτελείται μέσω του "spark-shell". Στην Εικόνα 3.10 παρουσιάζεται ο κώδικα για την δημιουργία ενός "DataFrame" και στις τέσσερις προαναφερθέντες γλώσσες προγραμματισμού.

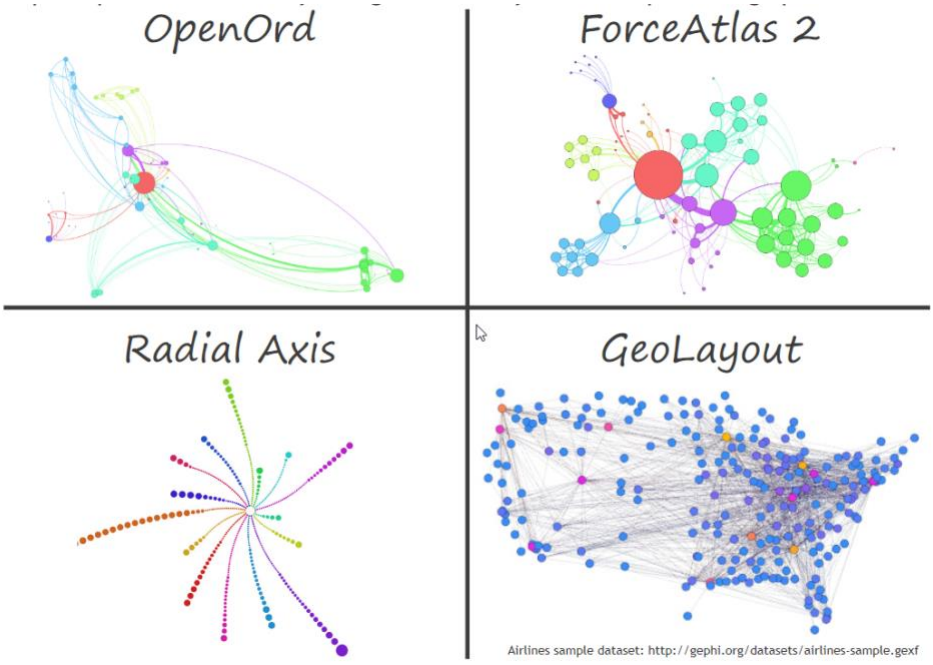
Εικόνα 3.10 Δημιουργία DataFrame σε Scala, Java Python, R από αρχείο JSON

Η υλοποίηση της Python, όπου φαίνεται στην Εικόνα 3.10, είναι σχεδόν ίδια θα έλεγε κανείς, η διαφορά είναι πως δεν χρειάζεται να δηλώσουμε στην Python την "df" ως μεταβλητή όπως κάνουμε στην Scala. Στη Scala απαιτείται να δηλώσουμε "val" αν είναι μεταβλητή που θέλουμε το περιεχόμενο της να είναι σταθερό, και "var" αν θέλουμε το περιεχόμενο της να αλλάζει τιμή, από εκχωρήσεις τιμών σε παρακάτω κομμάτια του κώδικα.

3.4 Gephi

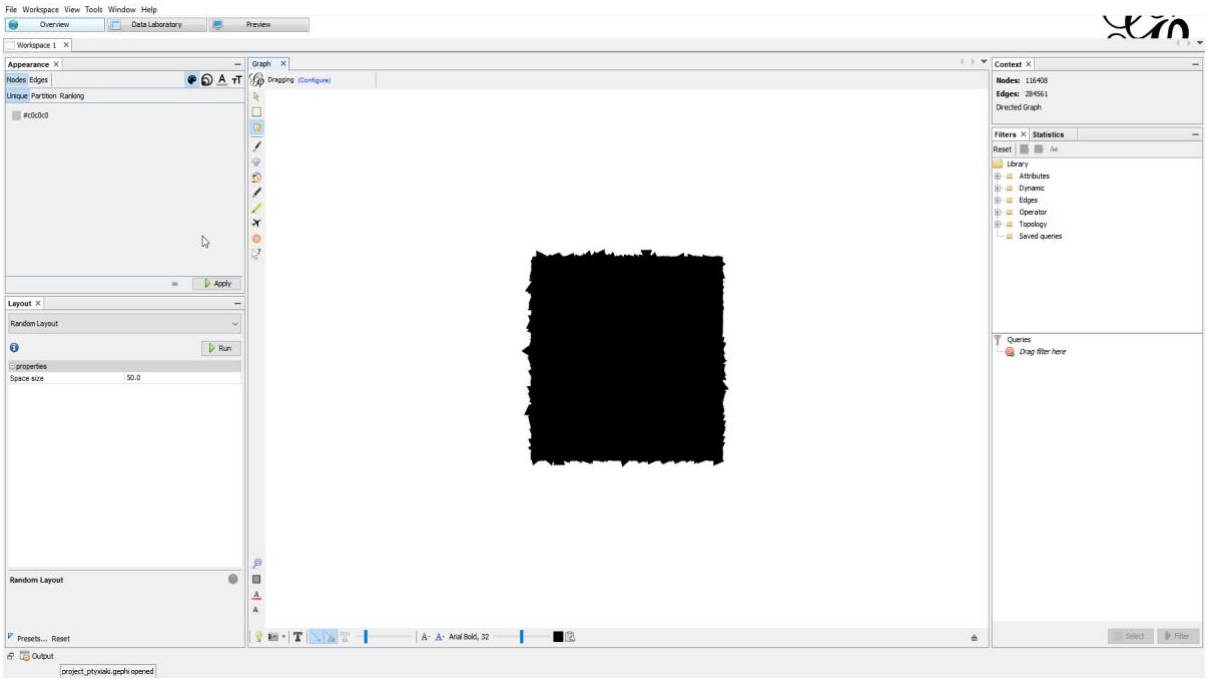
Το Gephi είναι λογισμικό ανοιχτού κώδικα που ειδικεύεται στην οπτικοποιήση και ανάλυση γράφων. Διαθέτει δυνατότητες οπτικοποίησης σε 3D μοντέλα και επιτρέπει την αλληλεπίδραση με τον χρήστη ώστε να επιτύχει, εξατομικευμένα αποτελέσματα. Για την εγκατάσταση του απαιτείται πρόσφατη έκδοση Java JRE και λειτουργικό σύστημα Microsoft Windows, macOS, Linux. Η πρώτη κυκλοφορία του προγράμματος έλαβε χώρα στις 31 Ιουλίου του 2008 και είναι αναπτύχθηκε με την χρήση Java και OpenGL. Διαθέτει ακόμη και σήμερα την αρχική του τοποθεσία στο "GitHub" https://github.com/gephi/gephi, όπου διατίθεται ελεύθερα ο κώδικας του προγράμματος στο ευρύ κοινό. Το Gephi μπορεί να εγκατασταθεί με την χρήση του πακέτου εγκατάστασης που είναι διαθέσιμό στο https://gephi.org/users/download/ .

Στην παρούσα εργασία χρησιμοποιήθηκε η έκδοση 0.9.2 που είναι και η πιο προσφατη και εγκαταστάθηκε σε λειτουργικό σύστημα Windows 10 64-bit. Το Gephi χρησιμοποιεί γνωστούς αλγορίθμους οπτικοποιήσης όπως είναι το ForceAtlas 2, OpenOrd, YifanHu Multilevel, Radial Axis Layout, GeoLayout και άλλοι. Στην Εικόνα 3.11 παρουσιάζονται παραδείγματα μερικών αλγορίθμων οπτικοποιήσης δεδομένων.



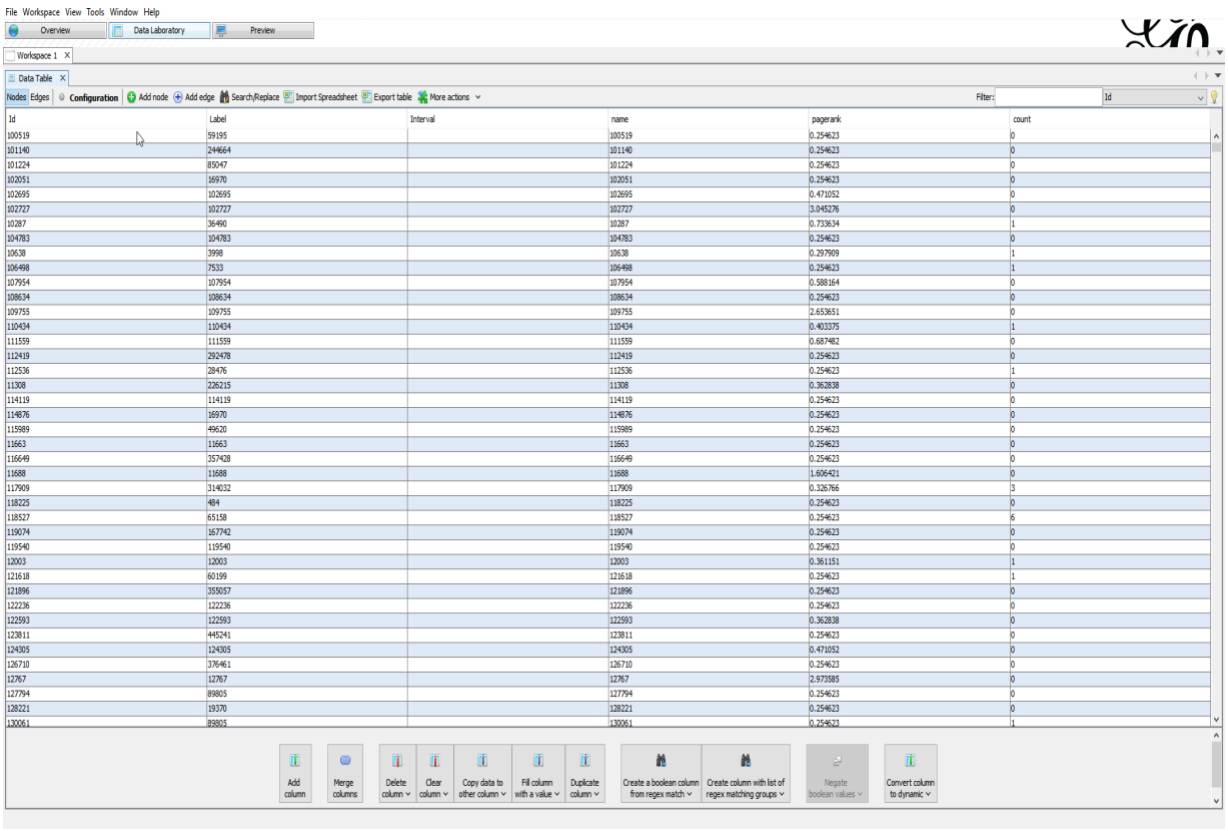
Εικόνα 3.11 Gephi παραδείγματα αλγορίθμων

Οι βασικές καρτέλες του γραφικού περιβάλλοντος του Gephi είναι το Overview όπου μπορούμε να εκτελέσουμε διάφορους στατιστικούς αλγορίθμους, να επιλέξουμε το αλγόριθμο που θα απεικονίσει τον γράφο και να χρωματίσουμε τους κόμβους. Στην Εικόνα 3.12 βλέπουμε έναν γράφο προς επεξεργασία στον οποίο δεν έχουν καθοριστεί παράμετροι διαχωρισμού των κόμβων, με αποτέλεσμα να μην διαχωρίζεται η μοναδικότητα των κόμβων.



Εικόνα 3.12 Καρτέλα "Overview" στο Gephi με γράφο προς επεξεργασία

Στην καρτέλα "Data Laboratory" μπορούμε να επεξεργαστούμε τα δεδομένα μας που βρίσκονται σε μορφή στηλών, σειρών. Μπορούμε να εισάγουμε δεδομένα από αρχείο csv που μπορεί να περιέχει κόμβους ή ακμές ακόμη και γενικού τύπου δεδομένα. Στην Εικόνα 3.13 απεικονίζεται η καρτέλα "Data Laboratory" με δεδομένα που χρησιμοποιήθηκαν στην παρούσα εργασία.



Id	Label	Interval	name	pagerank	count
100519	59195		100519	0.254623	0
101140	244664		101140	0.254623	0
101224	85047		101224	0.254623	0
102051	16970		102051	0.254623	0
102695	102695		102695	0.471052	0
102727	102727		102727	3.045276	0
10287	36490		10287	0.733634	1
104783	104783		104783	0.254623	0
10638	3998		10638	0.297909	1
106498	7533		106498	0.254623	1
107954	107954		107954	0.588164	0
108634	108634		108634	0.254623	0
109755	109755		109755	2.653651	0
110434	110434		110434	0.403375	1
111559	111559		111559	0.687482	0
112419	292478		112419	0.254623	0
112536	28476		112536	0.254623	1
11308	226215		11308	0.362838	0
114119	114119		114119	0.254623	0
114876	16970		114876	0.254623	0
115989	49620		115989	0.254623	0
11663	11663		11663	0.254623	0
116649	357428		116649	0.254623	0
11688	11688		11688	1.606421	0
117909	314032		117909	0.326766	3
118225	484		118225	0.254623	0
118527	65158		118527	0.254623	6
119074	167742		119074	0.254623	0
119540	119540		119540	0.254623	0
12003	12003		12003	0.361151	1
121618	60199		121618	0.254623	1
121896	355057		121896	0.254623	0
122236	122236		122236	0.254623	0
122593	122593		122593	0.362838	0
123811	445241		123811	0.254623	0
124305	124305		124305	0.471052	0
126710	376461		126710	0.254623	0
12767	12767		12767	2.973585	0
127794	89805		127794	0.254623	0
128221	19370		128221	0.254623	0
130061	89805		130061	0.254623	1

Εικόνα 3.13 Καρτέλα "Data Laboratory" στο Gephi και δεδομένα εργασίας

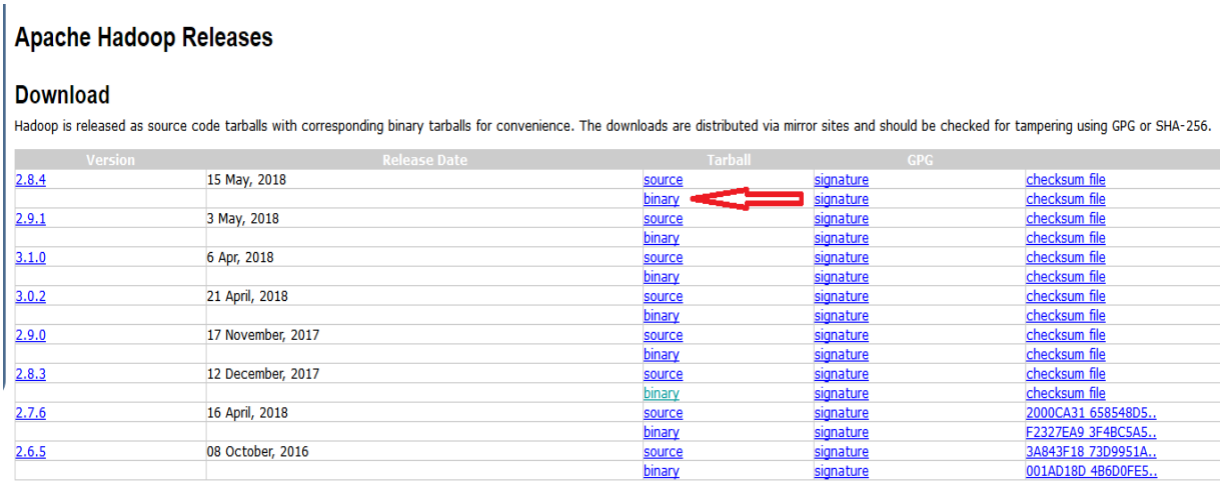
4 Εγκατάσταση Λογισμικού

Η εγκατάσταση του λογισμικού που παρουσιάστηκε στο προηγούμενο κεφάλαιο (Κεφάλαιο 3) πραγματοποιήθηκε σε ένα σύστημα με επεξεργαστή “Intel Core i7 6700K, 4 Cores 8 Threads @ 4.4GHz”, μνήμης RAM “24GB DDR4 @ 3000MHz” και μέσω αποθήκευσης “SSD 500GB Sata 3”. Το Debian Linux 9 που χρησιμοποιήθηκε για την εγκατάσταση και εκτέλεση των Apache Spark και Hadoop, εγκαταστάθηκε σε “Virtual Machine” με την χρήση του προγράμματος “VMware Workstation Pro 12” όπου του κατανεμήθηκαν “18GB” μνήμης RAM, “60GB” αποθηκευτικού χώρου και “6 vCPUs” (6 Threads). Για την χρήση του Gephi, η εγκατάσταση του πραγματοποιήθηκε σε Microsoft Windows 10 και είχε στην διάθεση του όλους τους διαθέσιμους πόρους του συστήματος.

4.1 Εγκατάσταση Apache Hadoop

4.1.1 Εγκατάσταση και παραμετροποίηση

Η εγκατάσταση που θα παρουσιαστεί έγινε σε λειτουργικό σύστημα Linux Debian 9, παρόμοια είναι και η εγκατάσταση σε λειτουργικό MacOS όπως και σε άλλα Unix συστήματα. Αρχικά επισκεπτόμαστε την ιστοσελίδα http://hadoop.apache.org/releases.html και επιλέγουμε την έκδοση που επιθυμούμε να «κατεβάσουμε» επιλέγοντας από την λίστα το “binary” όπως παρουσιάζεται στην Εικόνα 4.1:



Apache Hadoop Releases

Download

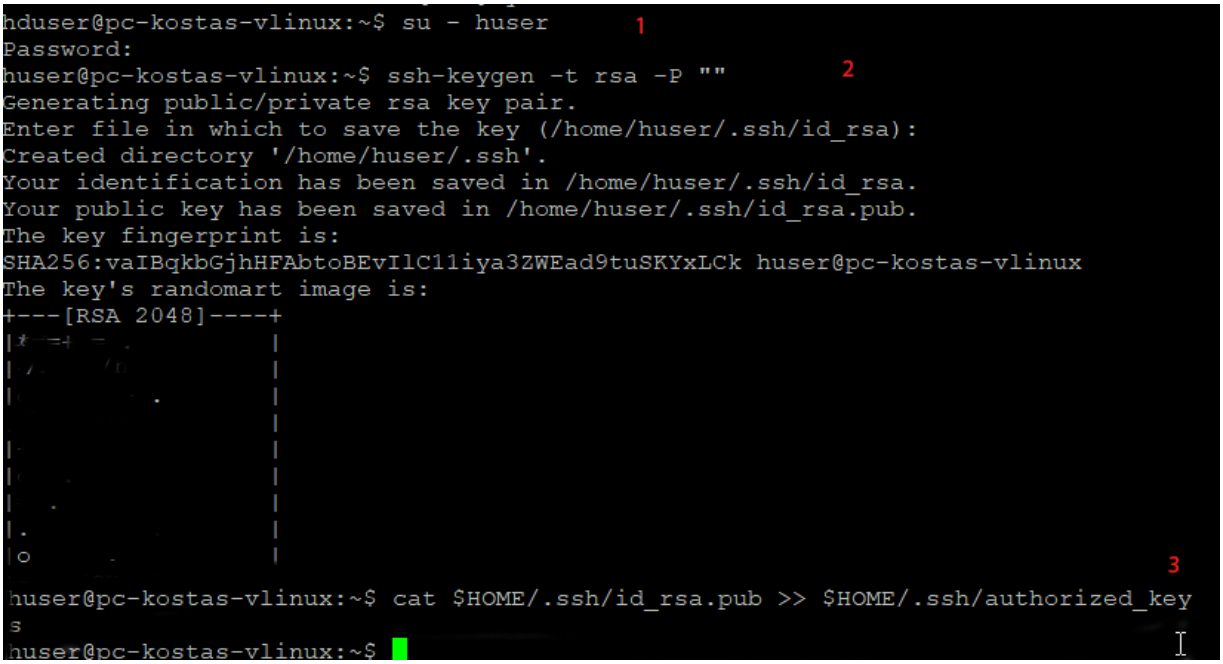
Hadoop is released as source code tarballs with corresponding binary tarballs for convenience. The downloads are distributed via mirror sites and should be checked for tampering using GPG or SHA-256.

Version	Release Date	Tarball	GPG	
2.8.4	15 May, 2018	source	signature	checksum file
		binary	signature	checksum file
2.9.1	3 May, 2018	source	signature	checksum file
		binary	signature	checksum file
3.1.0	6 Apr, 2018	source	signature	checksum file
		binary	signature	checksum file
3.0.2	21 April, 2018	source	signature	checksum file
		binary	signature	checksum file
2.9.0	17 November, 2017	source	signature	checksum file
		binary	signature	checksum file
2.8.3	12 December, 2017	source	signature	checksum file
		binary	signature	checksum file
2.7.6	16 April, 2018	source	signature	2000CA31 658548D5..
		binary	signature	F2327EA9 3F4BC5A5..
2.6.5	08 October, 2016	source	signature	3A843F18 73D9951A..
		binary	signature	001AD18D 4B6D0FE5..

Εικόνα 4.1 Λήψη Apache Hadoop

Στην συνέχεια, αποσυμπιέζουμε το αρχείο που έχει «κατέβει» με την χρήση της εντολής "tar -xvf hadoop-x.x.x.tar.gz" και δημιουργείται ως αποτέλεσμα ένας φάκελος "hadoop-x.x.x". Για λόγους διευκόλυνσης κάνουμε μετονομασία με πιο απλό όνομα με την εντολή mv "./hadoop-x.x.x ./hadoop". Πριν συνεχίσουμε σε επιμέρους παραμετροποίηση, πρέπει να εγκαταστήσουμε το "Java JDK", είτε της Oracle είτε το OpenJDK σε πρόσφατη έκδοση. Σε "Debian/Ubuntu" επιτυγχάνεται εκτελώντας την εντολή "sudo apt install openjdk-8-jdk". Μόλις τελειώσει η διαδικασία, μπορούμε να αρχίσουμε την παραμετροποίηση στο Apache Hadoop, ώστε να λειτουργεί σε "Single Node mode ή Pseudo-cluster", όπως και έχει χρησιμοποιηθεί στην παρούσα εργασία.

Αρχικά, πρέπει να δημιουργήσουμε έναν χρήστη στο λειτουργικό μας σύστημα, όπου θα εκτελείται το Hadoop. στην περίπτωση του Debian επιτυγχάνεται πληκτρολογώντας, "sudo addgroup hadoo" και "sudo adduser –ingroup hadoop huser". Πλέον, αφού δημιουργήθηκε χρήστης ο οποίος θα εκτελεί το Apache Hadoop, θα παραχωρήσουμε τα δικαιώματα του φακέλου εγκατάστασης όπως και την ιδιοκτησία του σε αυτόν με την εντολή "sudo chown -R huser:hadoop /home/huser/hadoop". Υποθέτουμε ότι το τρέχον μονοπάτι που είχαμε επιλέξει εξαρχής είναι το "/home/huser". Επειδή το Hadoop απαιτεί πρόσβαση "ssh" για να διαχειρίζεται τους κόμβους του cluster, και το σύστημα όπου θα εκτελείται το ίδιο όπως συμβαίνει στο "Single Node mode", πρέπει να το ρυθμιστεί για τον νέο χρήστη, εκτελώντας τις παρακάτω εντολές όπως εμφανίζονται στην Εικόνα 4.2.



Εικόνα 4.2 Δημιουργία SSH πρόσβασης στον νέο χρήστη

Στην συνέχεια, πρέπει να ορισθούν κάποια μονοπάτια για την εκπλήρωση των απαιτήσεων της εγκατάστασης του Hadoop αλλά και για δικιά μας διευκόλυνση, με πιο σημαντικό το "JAVA_HOME", στην Εικόνα 4.3 εμφανίζεται ότι εντολές πρέπει να εισάγουμε μέσα στο αρχείο ".bashrc". Δίνοντας την εντολή "nano ~/.bashrc" , πηγαίνουμε στο τέλος του αρχείου και προσθέτουμε τις γραμμές, που φαίνονται στην Εικόνα 4.3, χωρίς αυτές που ξεκινάνε με δίεση και στη συνέχεια, πληκτρολογούμε "source ~/.bashrc" για να εφαρμοστούν οι αλλαγές.

```
# $HOME/.bashrc
# Set Hadoop-related environment variables
export HADOOP_HOME=/home/huser/project/hadoop

# Set JAVA_HOME (we will also configure JAVA_HOME directly for Hadoop later on)
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/

# Add Hadoop bin/ directory to PATH
export PATH=$PATH:$HADOOP_HOME/bin
```

Εικόνα 4.3 Επεξεργασία αρχείου .bashrc

Αφού έχουμε ολοκληρώσει την παραπάνω διαδικασία, αποκτάμε πρόσβαση στις εντολές του HDFS δίχως να χρειάζεται να είμαστε στο μονοπάτι εγκατάστασης του "Hadoop", δηλαδή στην

περίπτωση μας στο "/home/huser/project/hadoop". Πλέον μπορούμε - μόλις ολοκληρώσουμε την εγκατάσταση και την παραμετροποίηση του - να εκτελούμε εντολές όπως "dfsadmin" που είδαμε προηγουμένως. Μεταβαίνουμε στο μονοπάτι "/home/huser/hadoop" και ξεκινάμε την παραμετροποίηση του από το αρχείο "hadoop-env.sh", χρησιμοποιώντας την εντολή "nano ./etc/hadoop/hadoop-env.sh" και πρέπει να διαμορφωθεί όπως παρουσιάζεται στην εικόνα 4.4.

```
# Set Hadoop-specific environment variables here.

# The only required environment variable is JAVA_HOME.  All others are
# optional.  When running a distributed configuration it is best to
# set JAVA_HOME in this file, so that it is correctly defined on
# remote nodes.

# The java implementation to use.
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre/

# The jsvc implementation to use. Jsvc is required to run secure datanodes
# that bind to privileged ports to provide authentication of data transfer
# protocol.  Jsvc is not required if SASL is configured for authentication of
# data transfer protocol using non-privileged ports.
#export JSVC_HOME=${JSVC_HOME}

export HADOOP_CONF_DIR=${HADOOP_CONF_DIR:-"/etc/hadoop"}
```

Μονοπάτι JAVA εγκατάστασης

Εικόνα 4.4 Παραμετροποίηση αρχείου hadoop-env.sh

Συνεχίζοντας, πρέπει να δημιουργήσουμε τα μονοπάτια όπου θα χρησιμοποιεί για προσωρινή αποθήκευση δεδομένων το "HDFS", με την εντολή "sudo mkdir -p /app/hadoo/tmp" και στην συνέχεια, παραχωρούμε δικαιώματα «ιδιοκτησίας» στον χρήστη του "Hadoop",με την εντολή "sudo chown huser:hadoop /app/hadoop/tmp". Ενισχύουμε την ασφάλεια με την εντολή "sudo chmod 750 /app/hadoop/tmp". Είναι σημαντικό βήμα διότι δεν θα είναι δυνατή η εκτέλεση του "Hadoop" λόγω έλλειψης δικαιωμάτων. Αφού έχουμε τρέχον μονοπάτι το "/home/huser/hadoop/", θα γίνει παραμετροποίηση στα αρχεία του "core-site.xml", "mapred-site.xml","hdfs-site.xml", σύμφωνα με την Εικόνα 4.5, 4.6 και 4.7, όπου βρίσκονται στον κατάλογο "./etc/hadoop/".

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
  <description>A base for other temporary directories.</description>
</property>

<property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:54310</value>
  <description>The name of the default file system.  A URI whose
  scheme and authority determine the FileSystem implementation.  The
  uri's scheme determines the config property (fs.SCHEME.impl) naming
  the FileSystem implementation class.  The uri's authority is used to
  determine the host, port, etc. for a filesystem.</description>
</property>
</configuration>
```

Ο κατάλογος που δημιουργήσαμε για προσωρινή αποθήκευση

Εικόνα 4.5 Τελική μορφή αρχείου core-site.xml

Όπως παρατηρούμε στην Εικόνα 4.5, όλες οι παράμετροι που θέλουμε να χρησιμοποιεί το Hadoop βρίσκονται υποχρεωτικά μεταξύ των ετικετών “<configuration></configuration>”, όπου και ενθυλακώνουν ετικέτες “<property></property>” όπου συμπεριλαμβάνουν τις παραμέτρους. Αυτή είναι η μορφοποίηση όπου ακολουθούν όλα τα αρχεία “XML”, που περιλαμβάνονται στον κατάλογο “./etc/hadoop/”.

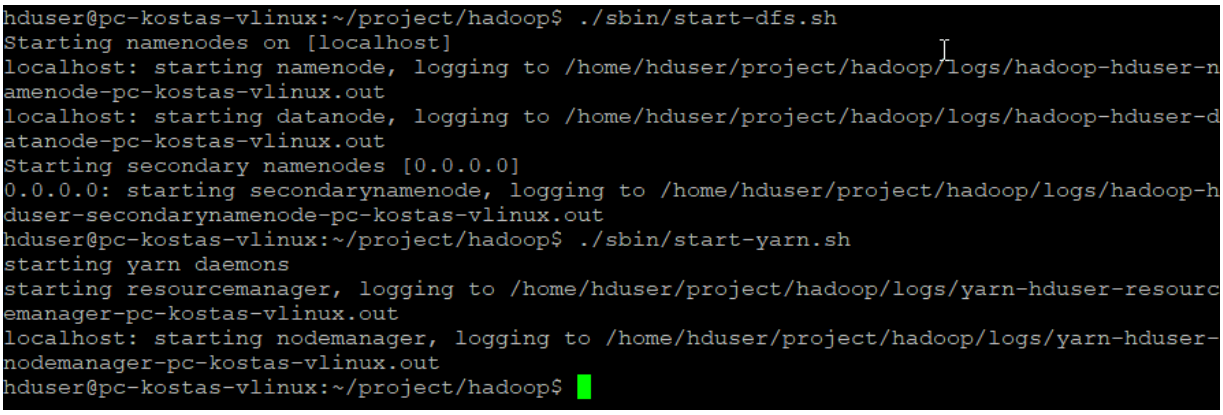
```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
  <name>dfs.replication</name>
  <value>1</value>
  <description>Default block replication.
  The actual number of replications can be specified when the file is created.
  The default is used if replication is not specified in create time.
  </description>
</property>
<property>                                  τοποθεσία της επιλογής μας για να αποθηκεύει το DataNode
  <name>dfs.datanode.data.dir</name>
  <value>/media/storage/dfs/dn</value>
  <description>Default block replication.
  The actual number of replications can be specified when the file is created.
  The default is used if replication is not specified in create time.
  </description>
</property>

</configuration>
```

Εικόνα 4.6 Τελική μορφή αρχείου hdfs-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
  <name>mapred.job.tracker</name>
  <value>localhost:54311</value>
  <description>The host and port that the MapReduce job tracker runs
  at.  If "local", then jobs are run in-process as a single map
  and reduce task.
  </description>
</property>
</configuration>
```

Εικόνα 4.7 Τελική μορφή αρχείου mapred-site.xml

Αφότου κάνουμε την παραμετροποίηση και αυτών των αρχείων, πλέον είμαστε έτοιμοι να εκτελέσουμε τις υπηρεσίες “HDFS” και “YARN” με τις εντολές “./sbin/start-dfs.sh” και “./sbin/start-yarn.sh” και μπορούμε να τερματίσουμε την εκτέλεση τους με “/sbin/stop-dfs.sh” και “./sbin/stop-yarn.sh



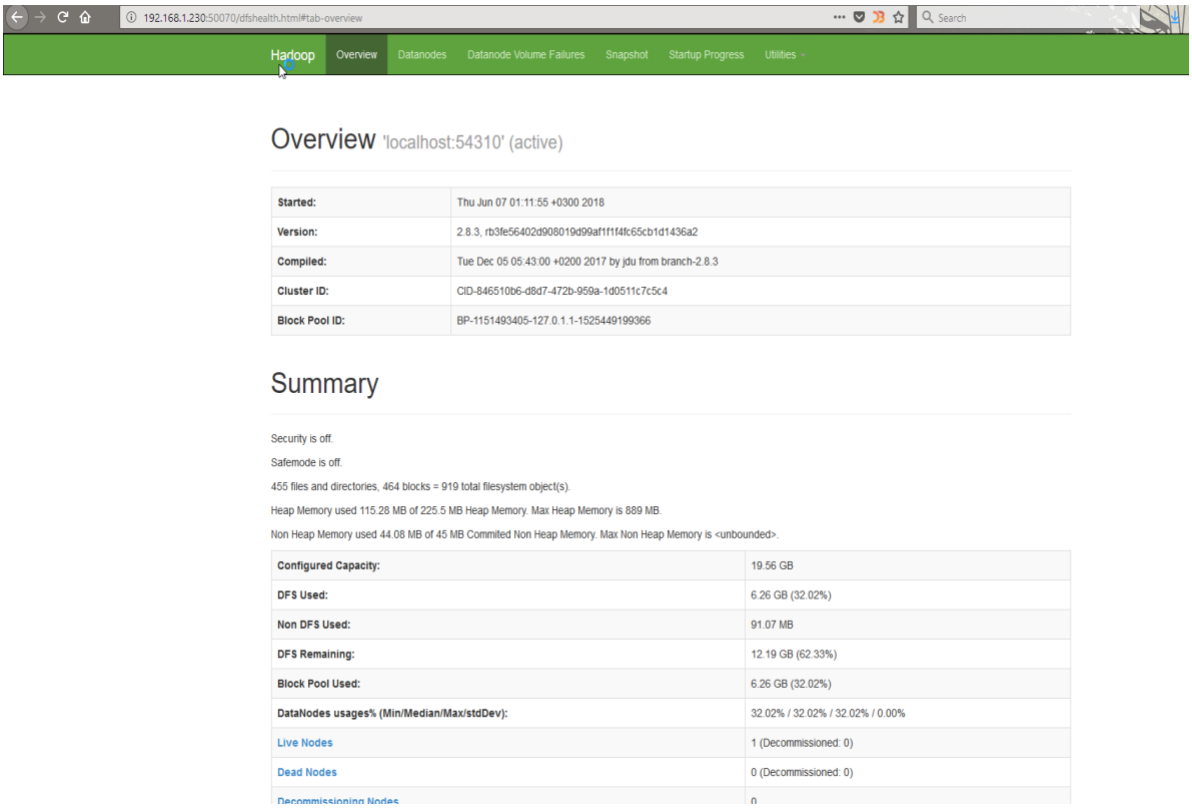
Εικόνα 4.8 Εκτέλεση Hadoop HDFS και YARN

Εικόνα 4.9 Τερματισμό εκτέλεσης Hadoop HDFS και YARN

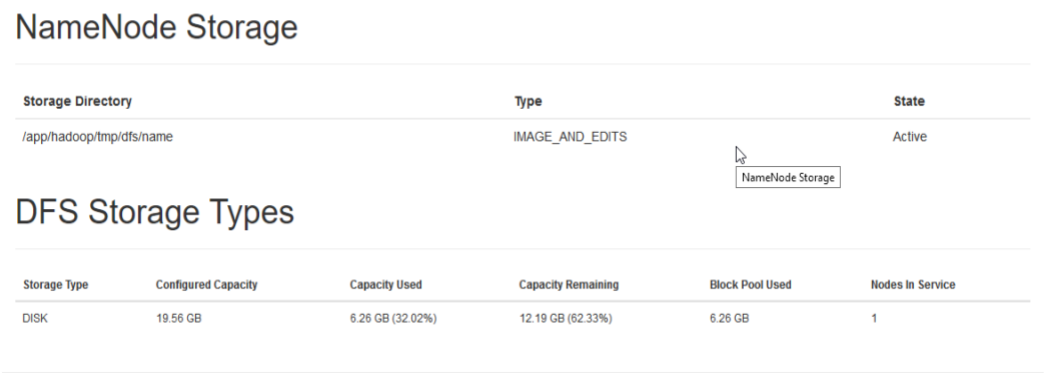
Όπως φαίνεται στην Εικόνα 4.8 με την εκκίνηση του “NameNode”, ότι εκκινεί και το “DataNode” το οποίο ανήκει στο “NameNode” το οποίο αναφέραμε προηγουμένως. Το “DataNode” είναι ο αποθηκευτικός χώρος που είναι διαθέσιμος στο “Single Node” cluster μας. Πλέον, έχει ενεργοποιηθεί και το περιβάλλον του χρήστη, που είναι καθαρά για σκοπούς παρακολούθησης της κατάστασης του συστήματος αρχείων και των εργασιών που θα αναλυθεί παρακάτω στην Υπο-ενότητα 4.1.2.

4.1.2 Περιβάλλον χρήστη και διαχείρισης

Το περιβάλλον του χρήστη για την παρακολούθηση της κατάστασης του "NameNode" στο σύνολό του είναι προσβάσιμο μέσω φυλλομετρητή (web browser) και χωρίζεται σε τρεις υπηρεσίες. Για πρόσβαση στο εργαλείο επιτήρησης του συστήματος αρχείων "HDFS" πληκτρολογούμε στο "browser" μας http://localhost:54310 , για πρόσβαση στο "DataNode" http://localhost:50010 και τέλος για πρόσβαση το εργαλείο επιτήρησης και παρακολούθησης εργασιών στο "YARN" http://localhost:8088. Στις Εικόνες 4.10, 4.11, 4.12 που ακολουθούν εμφανίζονται βασικά σημεία από κάθε περιβάλλον.

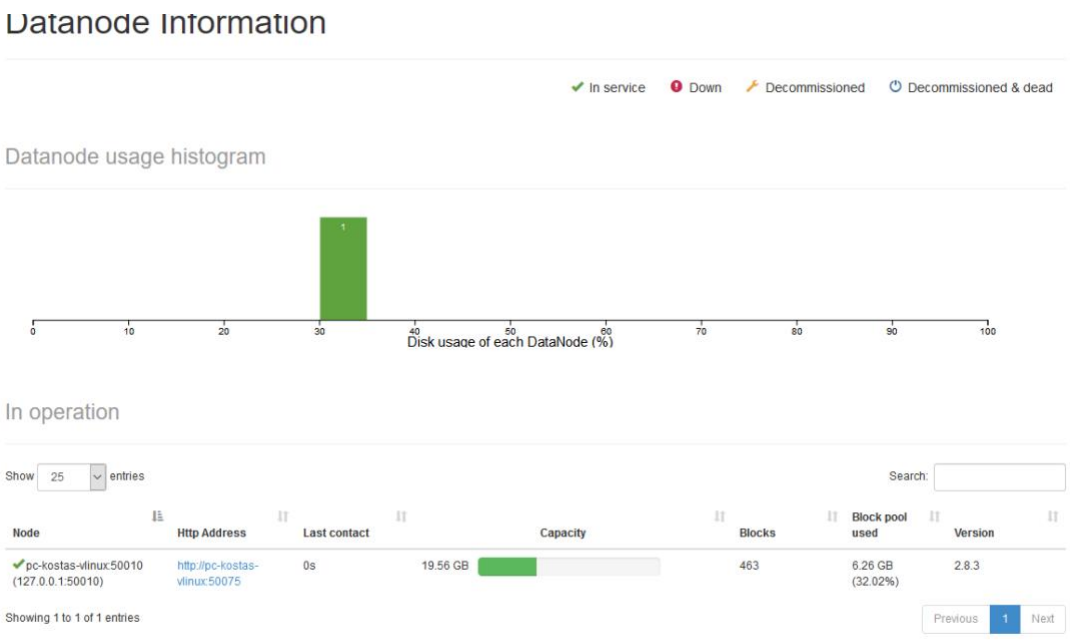


Εικόνα 4.10 Περιβάλλον κατάστασης HDFS

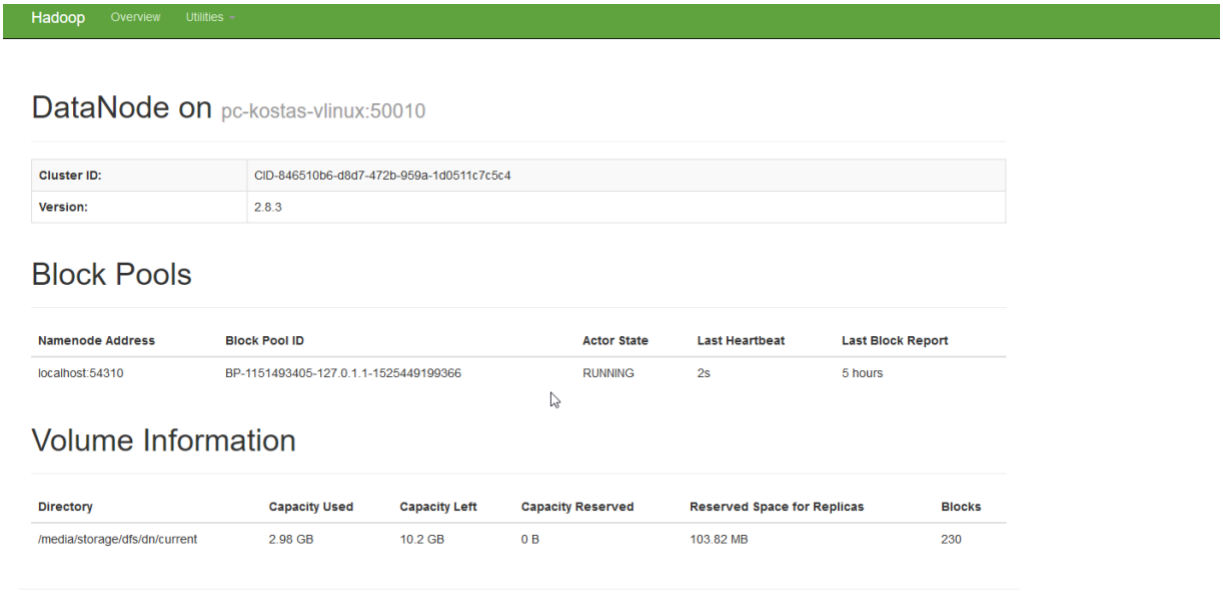


Εικόνα 4.11 Στοιχεία για το NameNode και DataNode

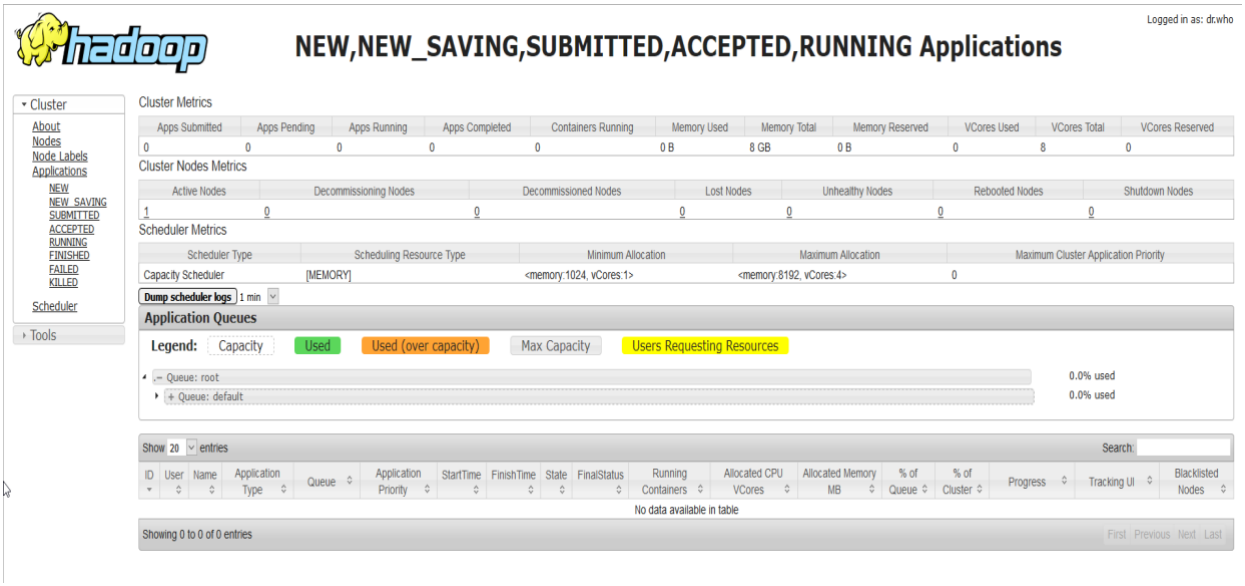
Στην Εικόνα 4.11, η στήλη "Storage Type" μπορεί να εμφανιστεί ανάλογα με την παραμετροποίηση και το υλισμικό του συστήματος η ετικέτα, "SSD", "RAM_DISK", "ARCHIVE". Το κάθε είδος υποδηλώνει τι πολιτική θα ακολουθήσει το "NameNode" , ως προς την εγγραφή των δεδομένων.



Εικόνα 4.12 Κατάσταση χωρητικότητας HDFS



Εικόνα 4.13 Περιβάλλον παρακολούθησης DataNode



Εικόνα 4.14 Scheduler YARN- Περιβάλλον παρακολούθησης εργασιών cluster

Στην Εικόνα 4.14 εμφανίζεται το περιβάλλον όπου δείχνει τις εργασίες που έχουν εισέλθει, ολοκληρωθεί και εκτελούνται αυτή τη στιγμή στο cluster.

Το περιβάλλον διαχείρισης είναι καθαρά “shell” σε αυτό το είδος της εγκατάστασης, δηλαδή ό,τι λειτουργία θέλουμε να κάνουμε που αφορά το “HDFS” πρέπει να δώσουμε εντολές σε περιβάλλον “shell”. Μερικές από τις πιο πολυχρησιμοποιημένες εντολές είναι “mkdir”, “rm” ,

“get” , “put”, “ls”. Αυτό που παρατηρούμε αμέσως είναι ότι είναι ίδιες με αυτές του “Linux”, ωστόσο ακολουθούν διαφορετικό συντακτικό. Εάν θέλουμε να δούμε τι περιέχει ένας κατάλογος, θα εκτελέσουμε “hdfs dfs -ls /project”, βάζοντας την παράμετρο “dfs” εννοούμε ότι προορίζεται να είναι εντολή για το σύστημα αρχείων “HDFS”. Οι εντολές “get” και “put” χρησιμοποιούνται για την λήψη και αποστολή δεδομένων μεταξύ του τοπικού συστήματος αρχείων και του “HDFS” και συντάσσονται “hdfs dfs -get <hdfs src path> … <local dest path>” και “hdfs dfs -put <local src path> … <hdfs dest path>“. Στην Εικόνα 4.15 εμφανίζεται ένα παράδειγμα εκτέλεσης των εντολών “mkdir” και “ls”, στην Εικόνα 4.16 παρουσιάζεται η σύνταξη ορισμένων διαθέσιμων εντολών.

```
hduser@pc-kostas-vlinux:~/project/hadoop$ hdfs dfs -mkdir -p /project/test
hduser@pc-kostas-vlinux:~/project/hadoop$ hdfs dfs -ls /project
Found 6 items
drwxr-xr-x   - hduser supergroup          0 2018-05-18 01:01 /project/cc
drwxr-xr-x   - hduser supergroup          0 2018-06-01 11:27 /project/datacsv
-rw-r--r--   1 hduser supergroup     854362 2018-06-06 17:24 /project/facebook_combined.t
xt
drwxr-xr-x   - hduser supergroup          0 2018-06-07 10:13 /project/test
-rw-r--r--   1 hduser supergroup   14702029 2018-05-16 17:07 /project/twitter_activity.tx
t
-rw-r--r--   1 hduser supergroup  180426543 2018-05-16 17:06 /project/twitter_social.edge
list
```

Εικόνα 4.15 Παράδειγμα εκτέλεσης mkdir και ls σε HDFS

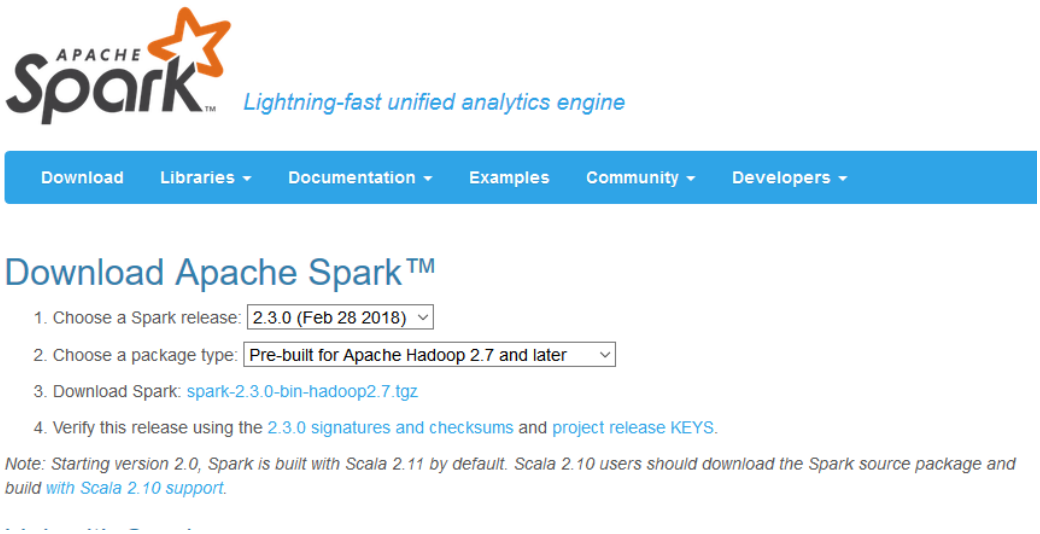
```
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] [-l] [-d] <localsrc> ... <dst>]
        [-copyToLocal [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-count [-q] [-h] [-v] [-t [<storage type>]] [-u] [-x] <path> ...]
        [-cp [-f] [-p | -p[topax]] [-d] <src> ... <dst>]
        [-createSnapshot <snapshotDir> [<snapshotName>]]
        [-deleteSnapshot <snapshotDir> <snapshotName>]
        [-df [-h] [<path> ...]]
        [-du [-s] [-h] [-x] <path> ...]
        [-expunge]
        [-find <path> ... <expression> ...]
        [-get [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-getfacl [-R] <path>]
        [-getfattr [-R] {-n name | -d} [-e en] <path>]
        [-getmerge [-nl] [-skip-empty-file] <src> <localdst>]
        [-help [cmd ...]]
        [-ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [<path> ...]]
        [-mkdir [-p] <path> ...]
        [-moveFromLocal <localsrc> ... <dst>]
        [-moveToLocal <src> <localdst>]
        [-mv <src> ... <dst>]
        [-put [-f] [-p] [-l] [-d] <localsrc> ... <dst>]
        [-renameSnapshot <snapshotDir> <oldName> <newName>]
        [-rm [-f] [-r|-R] [-skipTrash] [-safely] <src> ...]
        [-rmdir [--ignore-fail-on-non-empty] <dir> ...]
        [-setfacl [-R] [{-b|-k} {-m|-x <acl_spec>} <path>]|[--set <acl_spec> <path>]]
        [-setfattr {-n name [-v value] | -x name} <path>]
        [-setrep [-R] [-w] <rep> <path> ...]
        [-stat [format] <path> ...]
```

Εικόνα 4.16 Μερικές εντολές σχετικές με το HDFS

4.2 Εγκατάσταση Apache Spark

4.2.1 Εγκατάσταση και παραμετροποίηση

Όπως είδαμε η διαδικασία του Apache Hadoop απαιτεί τη δημιουργία καταλόγων και νέου χρήστη. Παρόμοια απαίτηση υπάρχει και για το Apache Spark. Για τις ανάγκες αυτής της εργασίας το Apache Spark εγκαταστάθηκε σε επιτραπέζιο υπολογιστή με Linux Debian 9, με παρόμοιο τρόπο πραγματοποιείται και η εγκατάσταση σε Unix-like (macOS, Linux διανομές) συστήματα αλλά και Microsoft Windows. Το πρώτο βήμα είναι να προβούμε στη λήψη του Apache Spark από την ιστοσελίδα του http://spark.apache.org/downloads.html, επιλέγοντας την έκδοση του Spark και του πακέτου υποστήριξης για την αντίστοιχη έκδοση Hadoop που έχουμε εγκαταστήσει, όπως εμφανίζεται στην Εικόνα 4.17.



Εικόνα 4.17 Λήψη Apache Spark

Στην συνέχεια, προχωράμε στην αποσυμπίεση, με την εντολή "tar -xvf spark-x.x.x-bin-hadoop2.x.tgz", Δημιουργείται ένας φάκελο με όνομα "spark-x.x.x", τον οποίο μετονομάζουμε με την χρήση της εντολής "mv ./spark-x.x.x-bin-hadoop2.x ./spark" για λόγους ευχρηστίας. Παραχωρούμε δικαιώματα και ιδιοκτησία στον χρήστη "huser" που δημιουργήσαμε προηγουμένως με την χρήση της εντολής "sudo chown -R huser:hadoop ./spark".

Θα δημιουργήσουμε μερικές συντομεύσεις που διευκολύνουν την χρήση του spark, προσθέτοντας στο αρχείο "~/.bashrc" τις γραμμές που φαίνονται στην Εικόνα 4.18, με την χρήση της εντολής "nano ~/.bashrc".

```
export SPARK_HOME=/home/huser/project/spark
export PATH=$PATH:$SPARK_HOME/bin
```

Εικόνα 4.18 Προσθήκη μονοπατιού για το Apache Spark ως μεταβλητή

Με αυτόν τον τρόπο, θα μπορούμε, όπως θα δούμε στη συνέχει να εκκινούμε το "Spark-shell' χωρίς να χρειάζεται να είμαστε στο μονοπάτι όπου εγκαταστάθηκε το Spark. Μετά και από την προσθήκη αυτή το αρχείο .bashrc, θα πρέπει να περιέχει αυτές τις γραμμές που εμφανίζονται στην Εικόνα 4.19.

```
# $HOME/.bashrc
# Set Hadoop-related environment variables
export HADOOP_HOME=/home/huser/project/hadoop

# Set JAVA_HOME (we will also configure JAVA_HOME directly for Hadoop later on)
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/

# Add Hadoop bin/ directory to PATH
export PATH=$PATH:$HADOOP_HOME/bin

export SPARK_HOME=/home/huser/project/spark
export PATH=$PATH:$SPARK_HOME/bin
```

Εικόνα 4.19 Τελική μορφή bashrc αρχείου

Για να γίνει η παραμετροποίηση το Apache Spark και να είναι έτοιμο για να αρχίσει η εκτέλεση του πρέπει να τροποποιηθούν δύο αρχεία που βρίσκονται στον κατάλογο "./spark/conf", το "spark-defaults.conf.template" και το "spark-env.sh.template". Αρχικά, δημιουργούμε αντίγραφα των πρωτότυπων αρχείων με την εντολή "cp -T ./spark/conf/spark-defaults.conf.template ./spark/conf/spark-defaults.conf" και "cp -T ./spark/conf/spark-env.sh.template ./spark/conf/spark-env.sh". Στη συνέχεια, με την εντολή "nano ./spark/conf/spark-env.sh", θα προσθέσουμε τις γραμμές όπου εμφανίζονται στην Εικόνα 4.20 και με την εντολή "nano ./spark/conf/spark-defaults.conf" προσθέτουμε τις γραμμές όπου εμφανίζονται στην Εικόνα 4.21.

```
export SPARK_DIST_CLASSPATH=$(/home/hduser/project/hadoop/bin/hadoop classpath)
export SPARK_DIST_CLASSPATH=$(/home/hduser/project/hadoop/bin/hadoop --config /home/hduser/project/hadoop/etc/hadoop)
```

Εικόνα 4.20 Αρχείο spark-env.sh τελική μορφή

Να σημειωθεί ότι, η πρώτη μεταβλητή καθορίζει την τοποθεσία εγκατάστασης του Apache Hadoop και η δεύτερη καθορίζει την διαδρομή όπου βρίσκονται τα αρχεία παραμετροποίησης του Apache Hadoop. Στην εγκατάσταση και την παραμετροποίηση των Apache Spark και Hadoop με τον χρήστη "hduser", εν αντιθέσει όπου παρουσιάζεται η εγκατάσταση με τον χρήστη "huser".

```
spark.master                     spark://master:7077
spark.eventLog.enabled           true
spark.eventLog.dir               hdfs://localhost:54310/user/hduser/spark-log
spark.local.dir                  /media/storage/tmp
#spark.serializer                 org.apache.spark.serializer.KryoSerializer
#spark.driver.memory              10g
#spark.default.parallelism        3
#spark.executor.memory           3192m
#spark.num.executors             6
```

Εικόνα 4.21 Αρχείο spark-defaults.con τελική μορφή

Οι πρώτες τέσσερις γραμμές της Εικόνας 4.21 είναι αυτές που χρησιμοποιεί το Apache Spark κατά την εκτέλεση του, αυτή τη στιγμή, οι υπόλοιπες είναι ενδεικτικές για το τι επιλογές υπάρχουν. Οι περισσότερες παράμετροι είναι εφικτό να καθοριστούν κατά την εκτέλεση του "Spark Shell", ώστε να είναι άμεσα μεταβαλλόμενες, διότι οι απαιτούμενοι πόροι κάθε εφαρμογής που έχουμε προς εκτέλεση, εξαρτώνται από τον όγκο των δεδομένων που θα εισάγουμε στην εφαρμογή μας. Με τον προαναφερθέντα τρόπο μας είναι εύκολο να διαχειριστούμε τους πόρους συστήματος, κυρίως αν είναι περιορισμένοι όπως και σε αυτήν την εργασία. Για τον λόγο αυτό, οι παράμετροι που είναι ανενεργοί στην Εικόνα 4.21 έχουν καθοριστεί κατά την εκτέλεση του "Spark Shell", όπως εμφανίζεται στην Εικόνα 4.22.

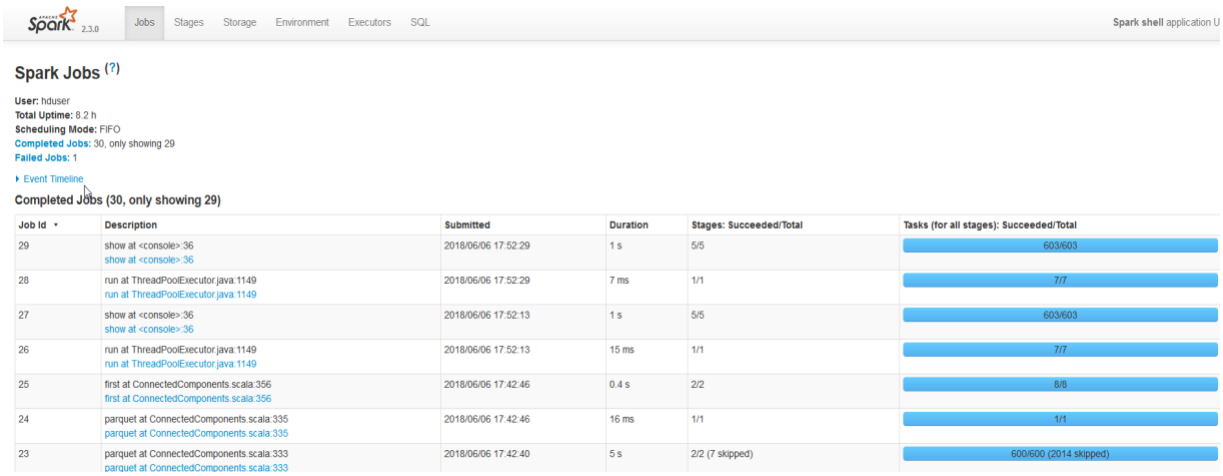
```
hduser@pc-kostas-vlinux:~/project$ spark-shell --master local[6] --driver-memory 14g --executor-memory 4096m --num-exec
utors 6 --executor-cores 1  --packages graphframes:graphframes:0.5.0-spark2.1-s_2.11, com.databricks:spark-csv_2.11:1.5
.0 --conf "spark.driver.extraJavaOptions=-Xss10m"
```

Εικόνα 4.22 Καθορισμός παραμέτρων κατά την εκτέλεση του Spark Shell

Τελειώνοντας όλη την παραμετροποίηση, πλέον η εκτέλεση ως “Standalone mode” εκκινεί με την εντολή “spark-shell –master local[x]”, όπου x είναι ο αριθμός των πυρήνων που θέλουμε να συμμετέχουν στην εκτέλεση του Spark. Ωστόσο η εκτέλεση μπορεί να ποικίλλει ανάλογα με τις ανάγκες. Ένα παράδειγμα εκτέλεσης φαίνεται της Εικόνας 4.22.

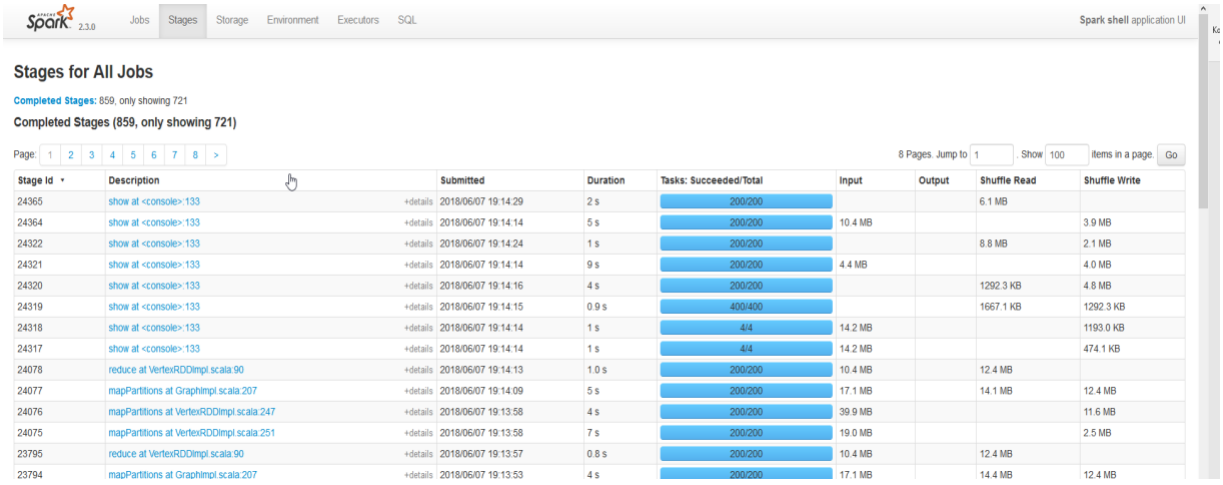
4.2.2 Περιβάλλον χρήστη και διαχείρισης

Η διαχείριση (Apache Software Foundation, 2018) γίνεται μέσω του κώδικα που θα γράψουμε για την εφαρμογή μας, μέσα από το “spark-shell”. Το περιβάλλον του χρήστη είναι μορφής “shell”, αλλά το περιβάλλον επιτήρησης είναι σε μορφή “Web” και είναι προσβάσιμο μέσω της διεύθυνσης http://localhost:4040. Εκεί φαίνεται η κατάσταση των εργασιών, των αιτημάτων όπου έχουν έρθει για εξυπηρέτηση είτε από την ίδια εφαρμογή, είτε στη περίπτωση του cluster από άλλες εφαρμογές.

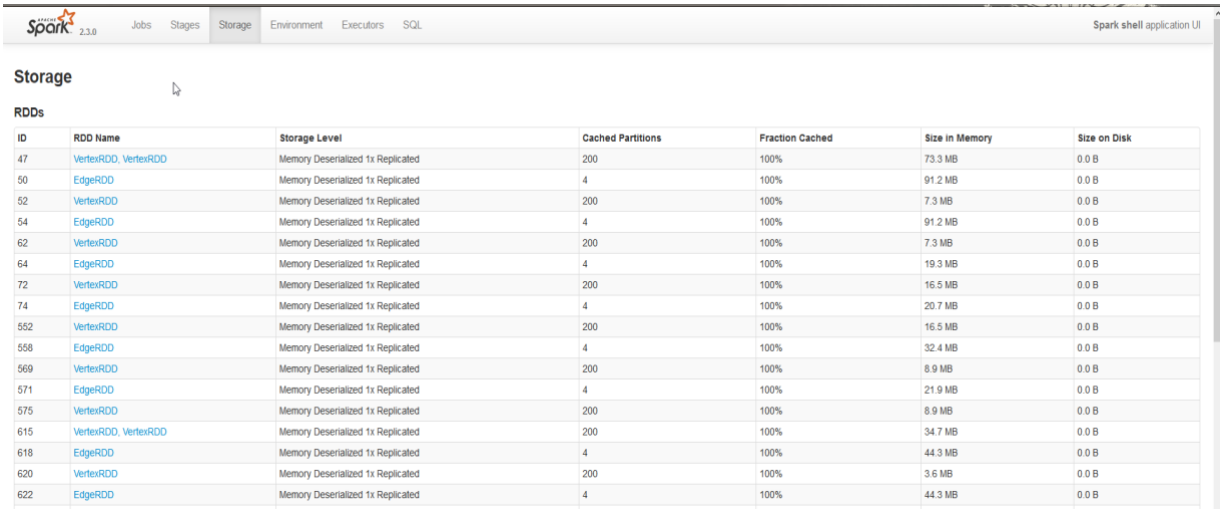


Εικόνα 4.23 Αρχική σελίδα περιβάλλον εποπτείας Spark

Στην Εικόνα 4.23, παρουσιάζεται ο αριθμός των ολοκληρωμένων εργασιών (Completed Jobs), ο χρόνος που εκτελείται το “Spark-shell” και ότι χρησιμοποιείται η μέθοδος “FIFO” για την εξυπηρέτηση των εργασιών.



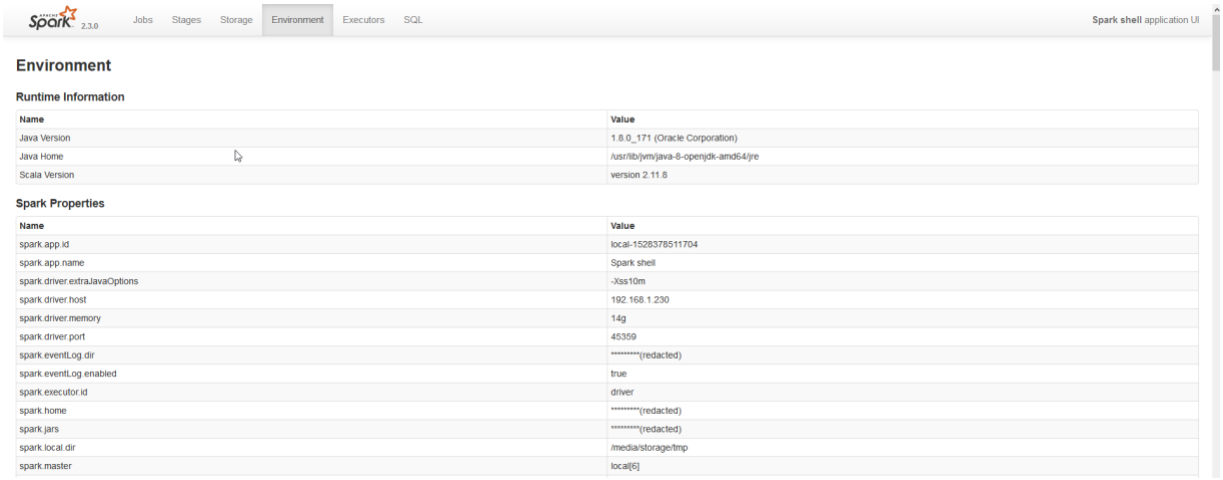
Εικόνα 4.24 Προβολή ολοκληρωμένων σταδίων για κάθε εργασία που εισήλθε



Εικόνα 4.25 Αποθηκευμένα στοιχεία σε RDD μορφή

Όπως φαίνεται στην Εικόνα 4.25, όλες οι εγγραφές δεσμεύουν μηδενικό χώρο στον δίσκο, ωστόσο δεσμεύουν χώρο στην μνήμη RAM, αυτό γίνεται αντιληπτό και από την στήλη "Storage Level" όπου αναγράφει το " Memory Deserialized 1x Replicated". Αυτό συμβαίνει, διότι το Apache Spark έχει ως πρωτεύον μέσω επεξεργασίας των δεδομένων την μνήμη RAM, διατηρώντας προσωρινά όλα τα αποτελέσματα μέχρις ότου να ζητηθεί να τα αποθηκεύσει κάπου. Αυτό συμβαίνει για λόγους ταχύτητας, αλλά και αρχιτεκτονικής του Apache Spark.

Όπως αναφέρθηκε το RDD λειτουργεί βέλτιστα αν επιλέγει σαν μέσο αποθήκευσης η μνήμη RAM (Williams, 2017).



Εικόνα 4.26 Πληροφορίες εγκατάστασης Apache Spark

5 Πειραματική Μελέτη

Με τη χρήση του Apache Spark και του Hadoop θα πραγματοποιηθεί ανάλυση γράφων σε δεδομένα του Facebook και του Twitter. Τα δεδομένα θα μελετηθούν με τη χρήση των αλγορίθμων PageRank, Triangle Counting και Label Propagation Algorithm που παρουσιάστηκαν στην Ενότητα 2.1, 2.2 και 2.5 αντίστοιχα. Πρόκειται για μια μελέτη για την συμπεριφορά των ανωτέρω αλγορίθμων σε διαφορετικούς τύπους δεδομένων. Αναμένουμε να δούμε διαφοροποίηση στα αποτελέσματα μας σε μεγάλο βαθμό μεταξύ των δύο “data-sets” που θα χρησιμοποιηθούν. Η μελέτη που γίνεται, μας επιτρέπει να εξετάσουμε πως η διαμόρφωση των δικτύων επηρεάζει τον τρόπο λειτουργίας των ατόμων και ομάδων που μετέχουν σε δικτυακές κοινότητες και θα δώσει μια έμμεση ιδέα για τη συμπεριφορά των αλγορίθμων αυτών απέναντι σε διάφορες συστάδες δεδομένων, που μπορεί να τύχει να επεξεργαστούμε.

Για τις ανάγκες της εργασίας χρησιμοποιήθηκαν δεδομένα από το “SNAP” [10], το οποίο είναι μια συλλογή δεδομένων που διατίθεται από το Stanford University που ξεκίνησε το 2004. Τα δεδομένα είναι ανώνυμα, δεν εκθέτουν πληροφορίες χρηστών και χρησιμοποιούνται για

ερευνητικούς σκοπούς από το ίδιο το Stanford αλλά και από άλλους ερευνητές, Χρησιμοποιήθηκαν δεδομένα που συλλέχθηκαν από το Twitter [18] https://snap.stanford.edu/data/higgs-twitter.html, στις 4 Ιουλίου 2012, την ημέρα που ανακαλύφθηκε το σωματίδιο "Higgs". Επίσης χρησιμοποιήθηκαν δεδομένα του Facebook [17] https://snap.stanford.edu/data/egonets-Facebook.html που αναπαριστά το δίκτυο που σχηματίζεται από τις συνδέσεις (ακμές) μεταξύ των χρηστών (κόμβοι).

Η μεθοδολογία που ακολουθείτε έχει ως στόχο την απεικόνιση των τελικών αποτελεσμάτων με την χρήση του Gephi τα οποία προέρχονται από την επεξεργασία των δεδομένων μέσω του Apache Spark

5.1 Facebook Graph Analytics

Αρχικά στην Εικόνα 5.1, εισάγουμε τα δεδομένα στο Apache Spark, από το αρχείο μορφής κειμένου όπου τα βρίσκουμε και στην συνέχεια θα δημιουργήσουμε κάποια "DataFrames" για την δημιουργία του γράφου.

```
001 import org.graphframes._
002 import org.apache.spark.sql._
003 import org.apache.spark.sql.functions._
004 import org.apache.spark.sql.types.{StructType, StructField, StringType,
IntegerType};
005
006 val customSchema = StructType(Array(StructField("src", IntegerType,
true),StructField("dst", IntegerType, true)))
007 val df = spark.read.option("header","true").option("delimiter", "
").schema(customSchema).csv("hdfs://localhost:54310/project/facebook_com-
bined.txt")
008 val edges = df.select("src","dst")
009 val n1 = edges.select("src").distinct()
010 val n2 = edges.select("dst").distinct()
011 val n = n1.union(n2).withColumnRenamed("src","name").distinct()
012 val nodes = n.withColumn("id", n("name"))
013 val g1 = GraphFrame(nodes, edges)
```

Εικόνα 5.1 Δημιουργία DataFrames και εισαγωγή δεδομένων στο Apache Spark, από αρχείο κειμένου

```
014 val k = g1.degrees.sort(desc("degree"))
015 k.show()
016 val pr2 = g1.pageRank.resetProbability(0.15).maxIter(10).run()
017 pr2.vertices.show()
018 val pr3 = pr2.vertices.sort(desc("pagerank"))
019 pr3.show()
```

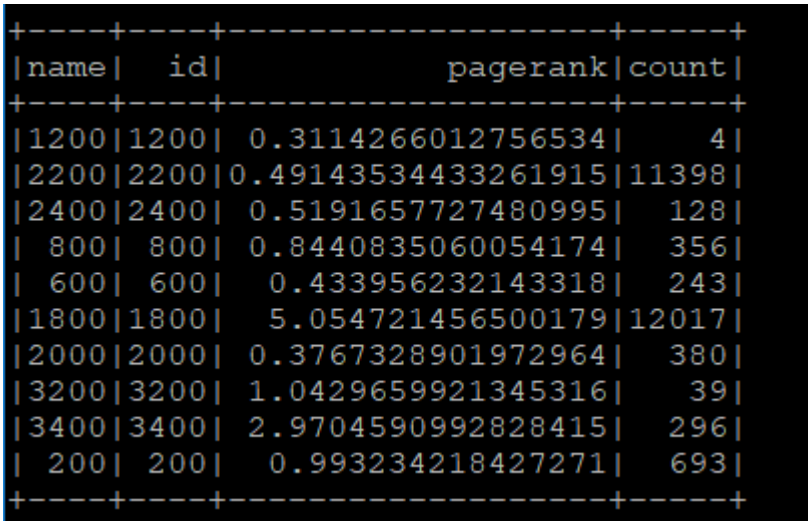
Εικόνα 5.2 Εμφάνιση βαθμών κορυφών και εκτέλεση αλγορίθμου PageRank

Ο κώδικας στην Εικόνα 5.2 εμφανίζει τους βαθμούς του γράφου και εκτελεί τον αλγόριθμο PageRank για το γράφο “g1”, στη συνέχεια ταξινομεί με καθοδική σειρά το DataFrame “pr2.vertices” και το προβάλει .

```
020 val tri = g1.triangleCount.run()
021 tri.persist()
022 tri.sort(desc("count")).limit(5).show()
023 val plotscat = pr2.vertices.join(tri, Seq("name", "id"))
024 plotscat.limit(10).show
```

Εικόνα 5.3 Εκτέλεση αλγορίθμου "Triangle count" εμφάνιση 5 μεγαλύτερων αποτελεσμάτων

Στην Εικόνα 5.3, δημιουργείται ένα νέο “DataFrame” από τον αλγόριθμο “Triangle Count” (Ενότητα 2.2). Στην συνέχεια γίνεται πράξη ένωσης δύο “DataFrames” ώστε να δημιουργήσουμε το “DataFrame” που θα χρησιμοποιήσουμε για την καρτεσιανή προβολή, όπου θα απεικονίζει “Triangles vs PageRank”.



name	id	pagerank	count
1200	1200	0.3114266012756534	4
2200	2200	0.49143534433261915	11398
2400	2400	0.5191657727480995	128
800	800	0.8440835060054174	356
600	600	0.433956232143318	243
1800	1800	5.0547214565001 79	12017
2000	2000	0.3767328901972964	380
3200	3200	1.0429659921345316	39
3400	3400	2.9704590992828415	296
200	200	0.993234218427271	693

Εικόνα 5.4 DataFrame plotscat που πρόκυψε μετά την ένωση δύο DataFrame

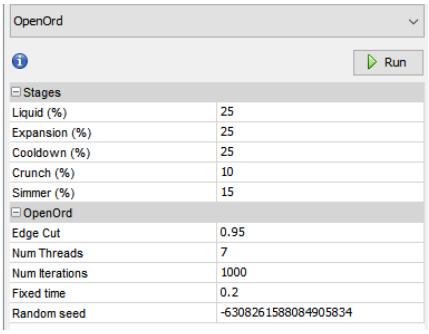
Στον κώδικα που εμφανίζεται στην Εικόνα 5.5, εκτελούμε τον αλγόριθμο “Label Propagation Algorithm” για να προβεί σε εντοπισμό κοινοτήτων και στην συνέχεια προχωρούμε σε ένωση “DataFrames” για την δημιουργία ενός νέου “DataFrame”, όπου στο τέλος εξάγουμε τρία αρχεία τύπου “csv” που περιέχουν κόμβους, ακμές και τα δεδομένα για την δημιουργία του καρτεσιανού γραφήματος.

```
025 val community_fb = pr2.labelPropagation.maxIter(10).run()
026 community_fb.show()
027 val stats_fb = community_fb.join(plotscat, Seq("name","id","pagerank"))
028
029
plotscat.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databricks.spark.csv").option("header","true").save("hdfs://localhost:54310/project/datacsv/plotscat/fb")
030
pr2.edges.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databricks.spark.csv").option("header","true").save("hdfs://localhost:54310/project/datacsv/graphs/fb_edges")
031
stats_fb.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databricks.spark.csv").option("header","true").save("hdfs://localhost:54310/project/datacsv/graphs/fb_nodes")
032
```

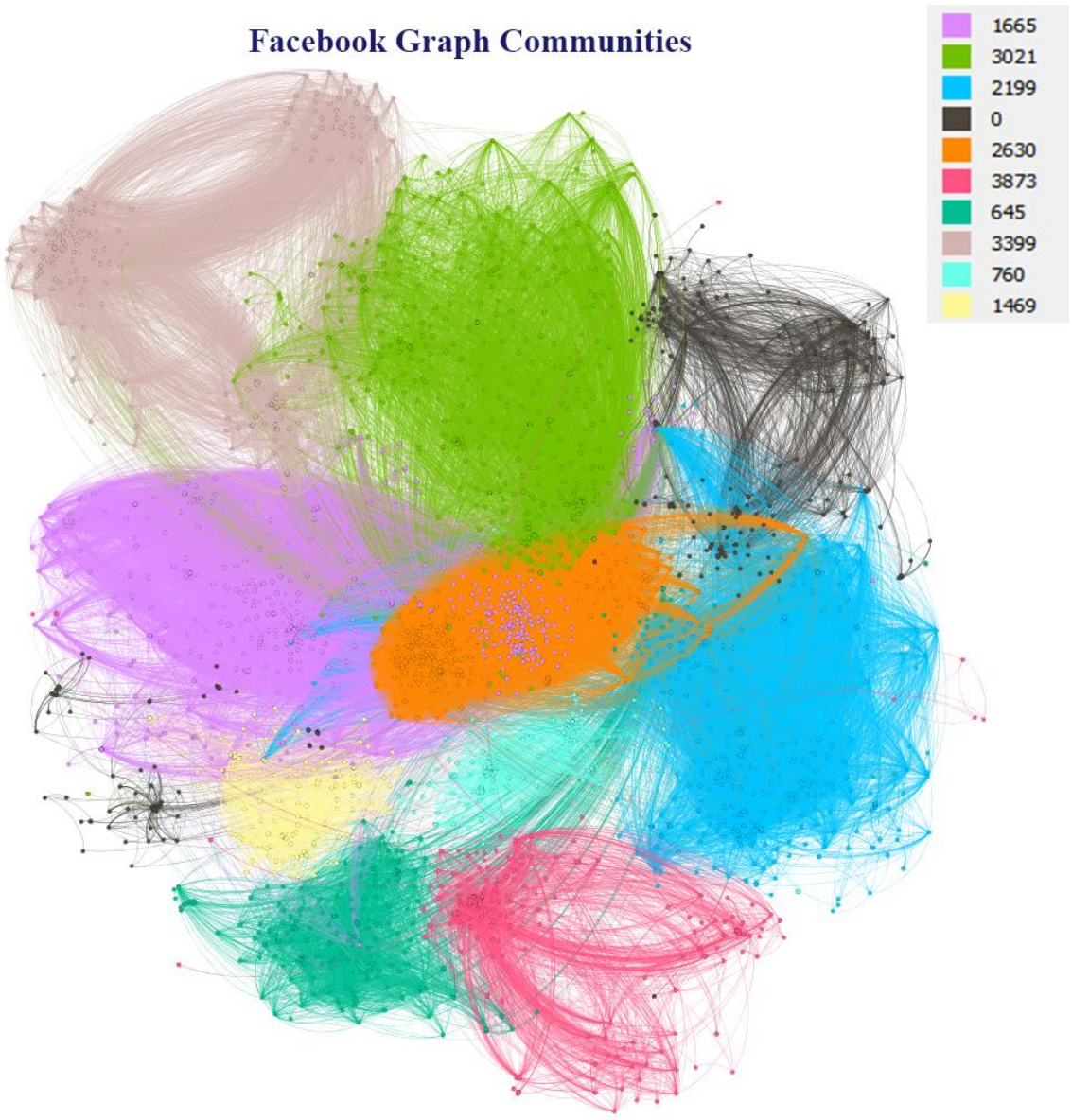
Εικόνα 5.5 Εντοπισμός κοινοτήτων από αλγόριθμο Label Propagation και εξαγωγή αρχείων CSV

5.1.1 Απεικόνιση Facebook Graph analysis

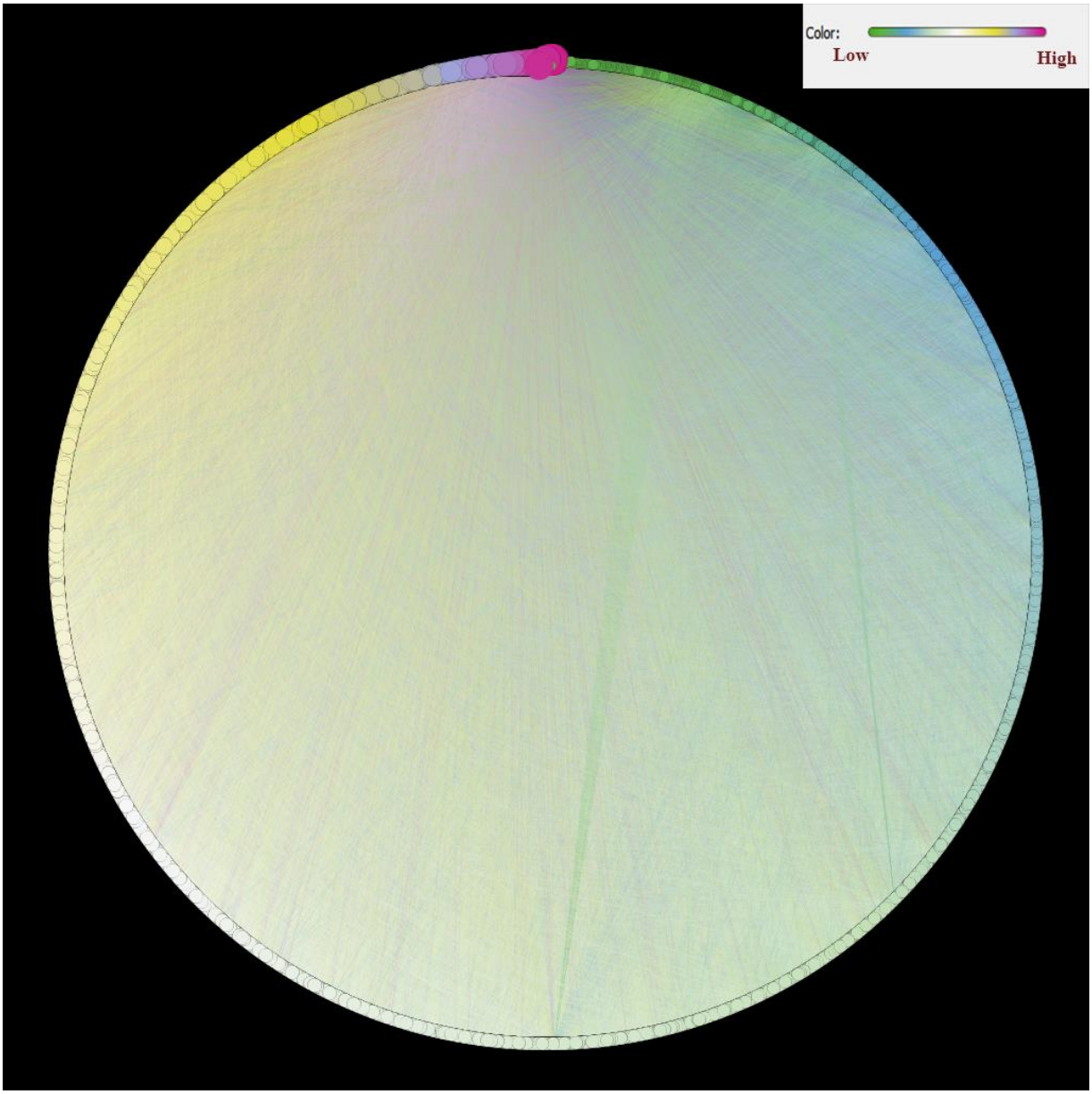
Με τη χρήση του Gephi και του αλγορίθμου “OpenOrd” με τις παραμέτρους που εμφανίζονται στην Εικόνα 5.6, μπορέσαμε να οπτικοποιήσουμε το γράφο με τις δέκα μεγαλύτερες κοινότητες που περιέχει, όπως φαίνεται στην Εικόνα 5.7. Οι κοινότητες δεν έχουν εμφανή διασπορά των κόμβων τους με αποτέλεσμα, όλα τα μέλη να είναι συγκεντρωμένα στην περιοχή της κάθε κοινότητας.



Εικόνα 5.6 OpenOrd παράμετροι

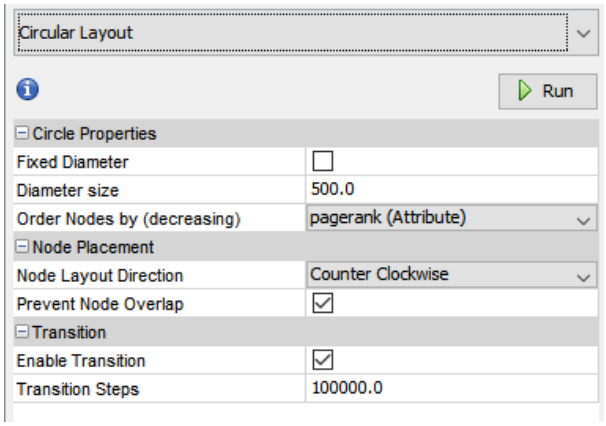


Εικόνα 5.7 Απεικόνιση κοινοτήτων δεδομένων Facebook



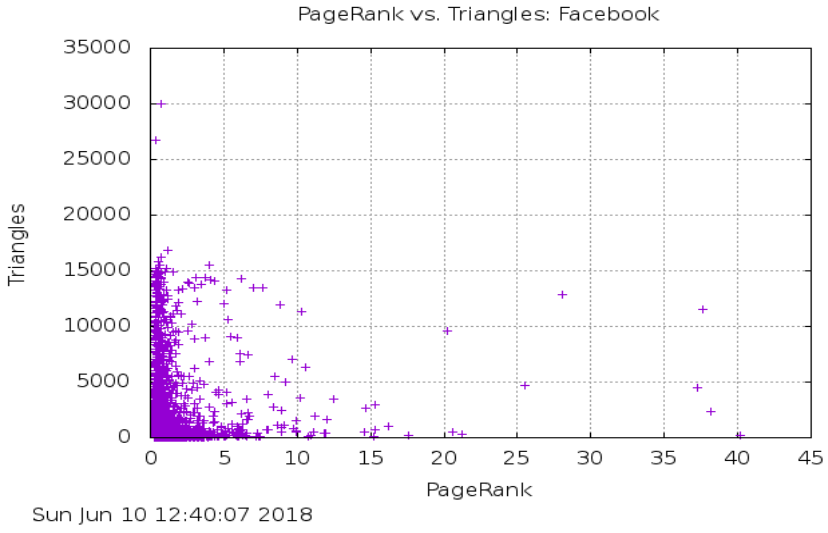
Εικόνα 5.8 Απεικόνιση γράφου Facebook με κατάταξη PageRank

Στην Εικόνα 5.8 απεικονίζονται οι κόμβοι με κατάταξη σύμφωνα με το "PageRank". Χρησιμοποιήθηκε ο αλγόριθμος "Circular Layout" για την απεικόνιση , με τις παραμέτρους που εμφανίζονται στην Εικόνα 5.9.



Εικόνα 5.9 Παραμετροποιήσεις Circular Layout

Όπως φαίνεται από το καρτεσιανό γράφημα της Εικόνας 5.10, το υψηλό PageRank δεν σημαίνει και απαραίτητα υψηλό αριθμό διερχομένων τριγώνων από τους κόμβους, για παράδειγμα βλέπουμε κόμβους που έχουν μικρότερος της μονάδας "PageRank" αλλά διέρχονται χιλιάδες τρίγωνα από αυτούς, και στην περίπτωση του υψηλού PageRank, έχουμε τρείς κόμβους από τους οποίους διέρχονται χιλιάδες τρίγωνα. Το αποτέλεσμα της Εικόνας 5.10 προήλθε με τα δεδομένα όπου χρησιμοποιήθηκαν στην προηγούμενη διαδικασία (Ενότητα 5.2) και εξήχθη με την χρήση "GNUPlot" και ενός "shell-script" που βρίσκεται στο Παράρτημα



Εικόνα 5.10 Καρτεσιανό γράφημα PageRank vs, Triangles

5.2 Twitter Higgs Activity Graph Analytics

Για την αν ανάλυση χρησιμοποιήθηκαν τα δεδομένα που βρίσκονται στο SNAP, "Higgs Activity Time" https://snap.stanford.edu/data/higgs-activity_time.txt.gz και "Social Network edgelist" https://snap.stanford.edu/data/higgs-social_network.edgelist.gz , όπου περιέχουν τις ενέργειες τον χρηστών στο διάστημα που ανακοινώθηκε η ανακάλυψη του σωματιδίου "Higgs" και τα "Follows" όπου έγιναν εκείνη την περίοδο, αντίστοιχα. Πρέπει να σημειωθεί ότι το αρχείο "Higgs Activity Time" περιέχει περίπου τριακόσιες δέκα χιλιάδες κόμβους και σχεδόν ένα εκατομμύριο ακμές.

```
001 import org.graphframes._
002 import org.apache.spark.sql._
003 import org.apache.spark.sql.functions._
004 import org.apache.spark.sql.types.{StructType, StructField, StringType,
IntegerType};
005 import org.graphframes.GraphFrame
006 import scala.collection.JavaConversions._
```

Εικόνα 5.11 Εισαγωγή απαραίτητων βιβλιοθηκών

Συνεχίζοντας δηλώνουμε κάποιες συναρτήσεις όπου θα μας βοηθήσουν στην εύρεση του νέων υπογράφων, εύρεση "Follows" και "Followers" των 20 κόμβων με το υψηλότερο PageRank από κάθε υπογράφο και στον διαχωρισμό των κοινοτήτων ανάλογα με το πλήθος των κόμβων όπου ανήκουν σε αυτές.

```
009 def nodes_filter(dfin:DataFrame, s:String):DataFrame ={
010 val schema = StructType(List(StructField("src", StringType,
false),StructField("dst", StringType, false),StructField("interaction",
StringType, false)))
011 var e = spark.createDataFrame(sc.emptyRDD[Row], schema)
012 if ((s=="RT")||(s=="MT")||(s=="RE")){
013 e = edges.select("src", "dst",
"interaction").filter(col("interaction")===s)
014 }
015 else{e = dfin}
016 val n1 = e.select("src").distinct()
017 val n2 = e.select("dst").distinct()
018 val n = n1.union(n2).withColumnRenamed("src","name").distinct()
019 val df_tmp = n.withColumn("id", n("name"))
020
021 df_tmp.select(df_tmp("name").cast("String"), df_tmp("id").cast("Int"))
022 }
```

Εικόνα 5.12 Συνάρτηση όπου διαχωρίζει τους κόμβους

Η συνάρτηση της Εικόνας 5.12 διαχωρίζει τους χρήστες (κόμβους), σύμφωνα με το είδος αλληλεπίδρασής τους όπου μπορεί να είναι "Retweet","Reply" ή "Mention", ώστε στη συνέχεια να μπορούμε να βγάλουμε υπό-γράφους σύμφωνα με την αλληλεπίδραση όπως φαίνεται στην Εικόνα 5.13. Επιπλέον στον κώδικα της Εικόνας 5.13, γίνεται εισαγωγή του αρχείου "Higgs Activity Time" σε "DataFrame". Με την χρήση των συναρτήσεων που δηλώσαμε αρχικοποιούμε τα "DataFrames" για τα γραφήματα σύμφωνα με την κατεύθυνση της πληροφορίας (για ενέργεια Retweet userB σε userA).

```
073 val df = spark.read.option("header","false").option("delimiter", "
").csv("hdfs://localhost:54310/project/twitter_activity.txt").toDF("userA",
"userB", "timestamp", "interaction")
074 val edges = df.select("userA" , "userB", "interaction").toDF("src",
"dst", "interaction")
075 println("===== Initializing graphs depending interaction =====")
076 val nrt = nodes_filter(edges,"RT")
077 val e2 = edges.select("dst", "src", "interaction").filter("interaction =
'RT'").toDF("src", "dst", "interaction")
078 val g_rt = GraphFrame(nrt, e2)
079 val nmt = nodes_filter(edges,"MT")
080 val e3 = edges.filter("interaction = 'MT'")
081 val g_mt = GraphFrame(nmt, e3)
082 val nre = nodes_filter(edges,"RE")
083 val e4 = edges.filter("interaction = 'RE'")
084 val g_re = GraphFrame(nre, e4)
085 val nodes = nodes_filter(edges,"g")
086 val uedges = (e2.union(e3)).union(e4)
087 uedges.persist()
088 val g = GraphFrame(nodes, uedges)
089 val pr_g = g.pageRank.resetProbabilit
```

Εικόνα 5.13 Αρχικοποίηση γράφων

Στην Εικόνα 5.14 η συνάρτηση βρίσκει στο Γράφο "Social Network edgelist" τους "Followers" και τα "Follows" του κάθε χρήστη, ως όρισμα δέχεται ένα "DataFrame".

```
023 def social_sts(graph_df: DataFrame):DataFrame ={
024 val schema = StructType(List(StructField("id", StringType,
false),StructField("follow", IntegerType, false)))
025 var tmp = spark.createDataFrame(sc.emptyRDD[Row], schema)
026 var tmp2 = spark.createDataFrame(sc.emptyRDD[Row], schema)
027 var follow_df = spark.createDataFrame(sc.emptyRDD[Row], schema)
028 var follower_df = spark.createDataFrame(sc.emptyRDD[Row], schema)
029 var social_df = spark.createDataFrame(sc.emptyRDD[Row], schema)
030 val nodelist = graph_df.select("name").limit(20).collectAsList()
031 var cnt = 0
032 for(i <- nodelist){
033 tmp = (g_social.outDegrees.select("id",
"outDegree").filter(g_social.outDegrees("id")===
nodelist(cnt).getString(0))).toDF("id", "Follow")
034 tmp2 = (g_social.inDegrees.select("id",
"inDegree").filter(g_social.inDegrees("id")===
nodelist(cnt).getString(0))).toDF("id", "Followers")
035 follow_df = follow_df.union(tmp)
036 follower_df = follower_df.union(tmp2)
037 cnt = cnt+1
038 }
039 (follow_df.join(follower_df, Seq("id"))).toDF("id","follow","followers")
040 }
```

Εικόνα 5.14 Συνάρτηση εύρεσης Follows και Followers χρηστών

Η Εικόνα 5.15 απεικονίζει την εισαγωγή του αρχείου "Social Network Edgelist" σε "DataFrame" και την δημιουργία νέου γράφου. Στην συνέχεια, καλείτε η συνάρτηση "social_sts" της Εικόνας 5.15, για κάθε υπό-γράφο που έχει δημιουργηθεί από τον κώδικά της Εικόνας 5.16 και επιστρέφει τους "Followers" και τα "Follows", των είκοσι πιο δημοφιλών κόμβων από κάθε υπό-Γράφο.

```
099 val twitter_social_df =
spark.read.option("header","false").option("delimiter", "
").csv("hdfs://localhost:54310/project/twitter_social.edgelist").toDF("src",
"dst")
100 val social_nodes = nodes_filter(twitter_social_df,"tw")
101 val g_social = GraphFrame(social_nodes, twitter_social_df)
140 var social_rt = social_sts(result_pr_rt)
142 var social_re = social_sts(result_pr_re)
144 var social_mt = social_sts(result_pr_mt)
146 social_rt.persist()
147 social_re.persist()
148 social_mt.persist()
149 var top_social =
((social_rt.union(social_re)).union(social_mt)).distinct()
150 top_social.persist()
151 top_social.show()
```

Εικόνα 5.15 Εύρεση Follow και Followers

Στην Εικόνα 5.16 εκτελούμε τον αλγόριθμο PageRank για κάθε Γράφο και τον εμφανίζουμε τα αποτελέσματα με φθίνουσα κατάταξη.

```
089 val pr_g = g.pageRank.resetProbability(0.15).tol(0.01).run()
090 val result_pr_g = pr_g.vertices.sort(desc("pageRank"))
091 val pr_rt = g_rt.pageRank.resetProbability(0.15).tol(0.01).run()
092 val result_pr_rt = pr_rt.vertices.sort(desc("pagerank"))
093 val pr_mt = g_mt.pageRank.resetProbability(0.15).tol(0.01).run()
094 val result_pr_mt = pr_mt.vertices.sort(desc("pagerank"))
095 val pr_re = g_re.pageRank.resetProbability(0.15).tol(0.01).run()
096 val result_pr_re = pr_re.vertices.sort(desc("pagerank"))
```

Εικόνα 5.16 Εκτέλεση αλγορίθμου PageRank για κάθε Γράφο

```
042 def splitby(graphdf:DataFrame):DataFrame ={
043 val schema= StructType(List(StructField("label", StringType,
false),StructField("count", IntegerType, false)))
044 var tmp = spark.createDataFrame(sc.emptyRDD[Row], schema)
045 var bigc_nodes = spark.createDataFrame(sc.emptyRDD[Row], schema)
046 var tmp2 = graphdf.groupBy("label").agg(count("label").alias("members"))
047 val lblist= tmp2.select("label").filter("members > 1000").collectAsList()
048 var cnt =0
049 for(i <- lblist){
050 tmp = graphdf.select("name","id").filter(col("label") ===
lblist(cnt).getLong(0))
051 bigc_nodes = bigc_nodes.union(tmp)
052 cnt = cnt+1
053 }
054 bigc_nodes.toDF("name","id")
055 }
```

Εικόνα 5.17 Συνάρτηση όπου ξεχωρίζει τις μεγαλύτερες κοινότητες από έναν Γράφο

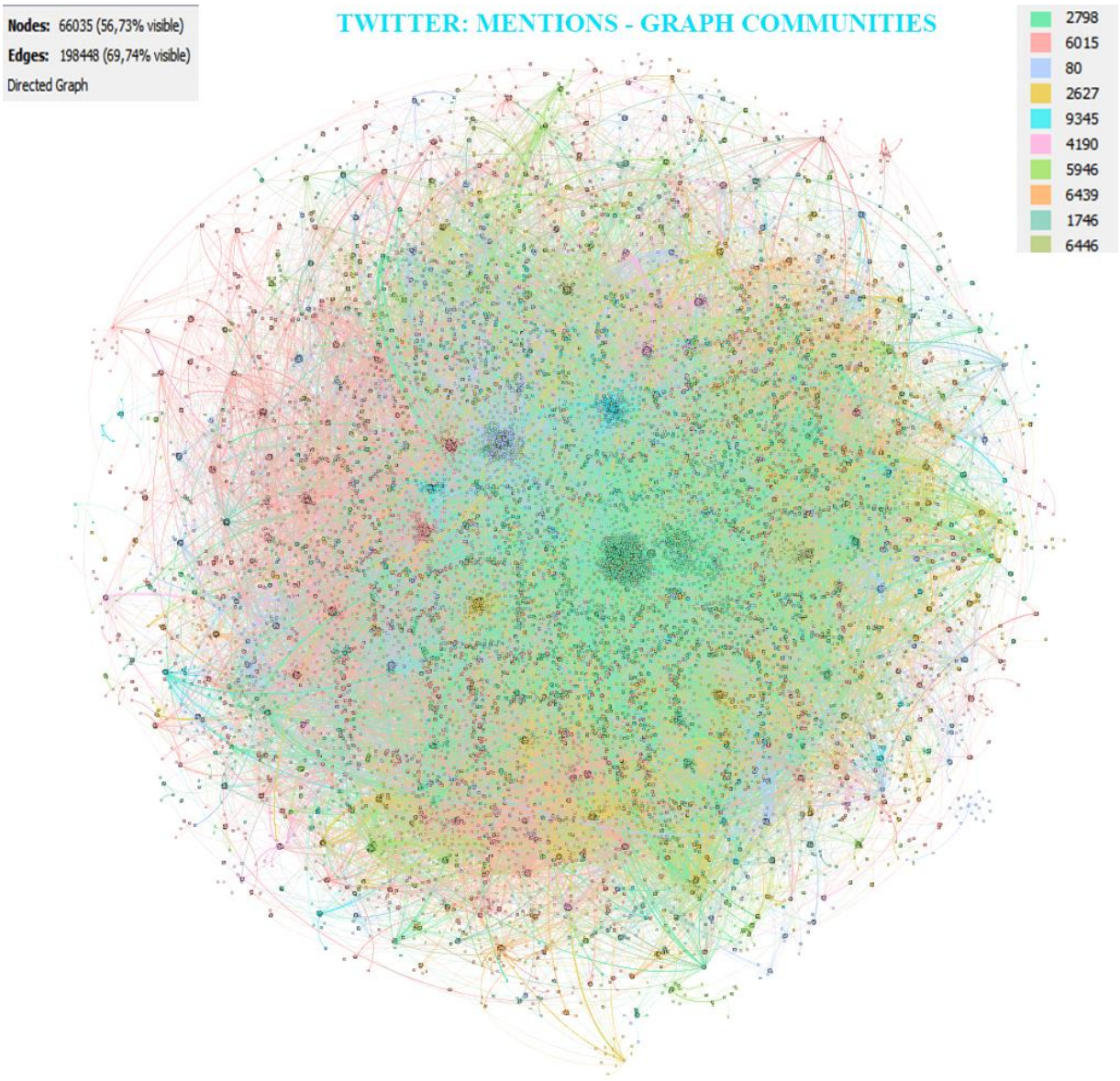
Στην Εικόνα 5.17 φαίνεται μία συνάρτηση όπου ομαδοποιεί τις κοινότητες των τριών υπογράφων λαμβάνοντας υπόψην το πλήθος των μελών μιας κοινότητας, όπου πρέπει να είναι τουλάχιστον 1001 μέλη και τις εκχωρεί σε ένα νέο "DataFrame" με στήλες "name" και "id". Ο σκοπός της συνάρτησης είναι η δημιουργία ενός γράφου με τις μεγαλύτερες κοινότητες από κάθε υπογράφο.

5.2.1 Απεικόνιση Twitter Graph analysis

Στο Παράρτημα Α υπάρχει ο πλήρης κώδικας, όπου παρουσιάζει την εξαγωγή των αρχείων csv, ώστε να εισαχθούν στο Gephi για να γίνει η απεικόνιση των γράφων με τον αλγόριθμο "OpenOrd", ο οποίος θα διακρίνει τις κοινότητες κόμβων ενώ ο "Circular Layout" ο οποίος θα διακρίνει το "PageRank" των κόμβων όπως παρουσιάζεται στην Εικόνα 6.7 στα δεδομένα του Facebook.

Στην Εικόνα 5.18 αναπαρίσταται οι δέκα κορυφαίες κοινότητες του γράφου "Twitter users: Mentions", δηλαδή των χρηστών που έχουν κάνει "Mention" κάποιον άλλον χρήστη. Αυτό που παρατηρείται είναι ότι υπάρχει πολύ μεγάλη διασπορά των κόμβων, με αποτέλεσμα οι

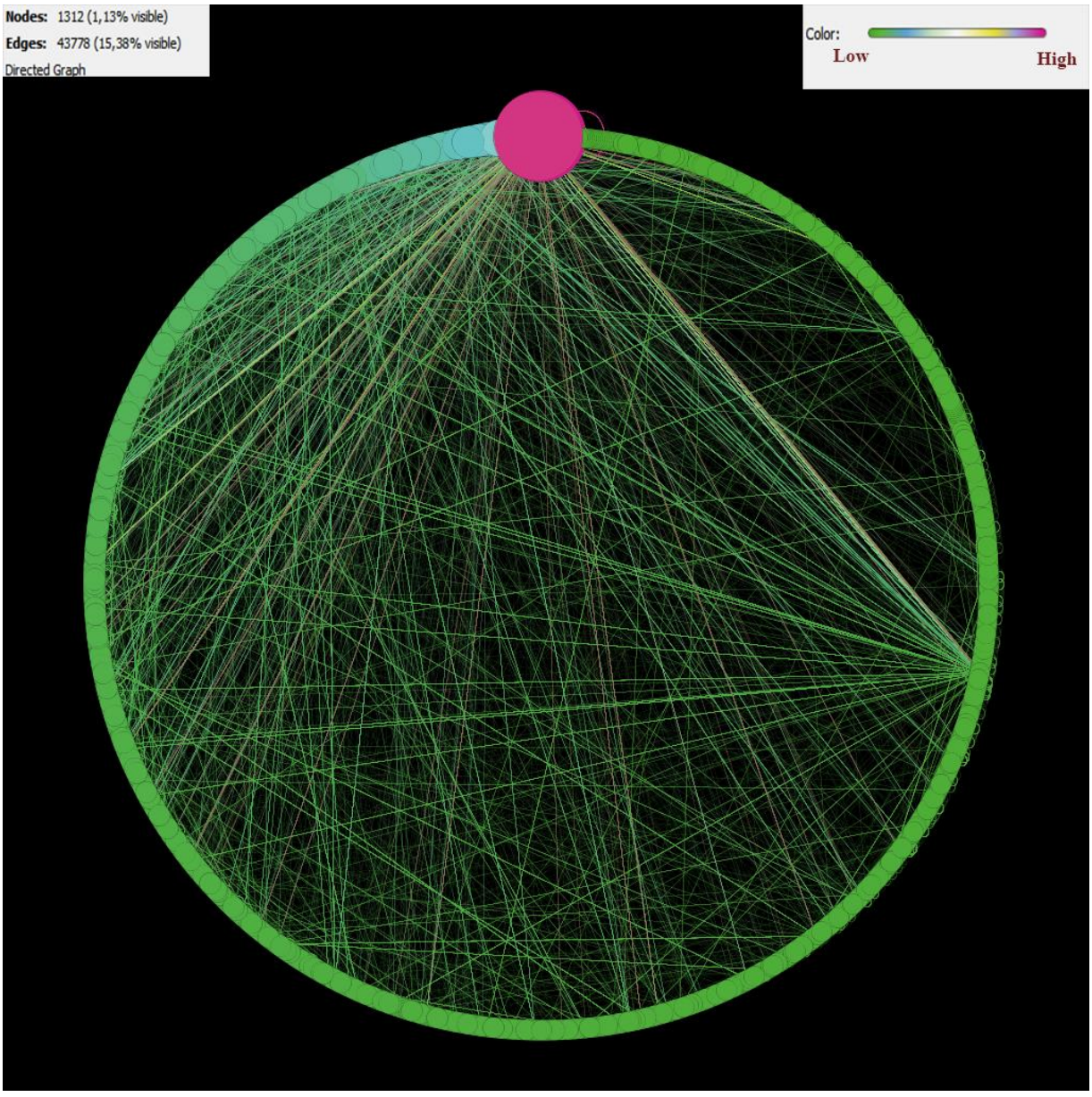
κοινότητες να μην είναι διακριτές εύκολα. Επιπλέον στην αναπαράσταση φαίνεται ο αριθμός των κόμβων που συμμετέχουν και οι ακμές όπου συμμετέχουν στην δημιουργία του.



Εικόνα 5.18 Κοινότητες υπογράφου Twitter users: Mentions με αλγόριθμο OpenOrd

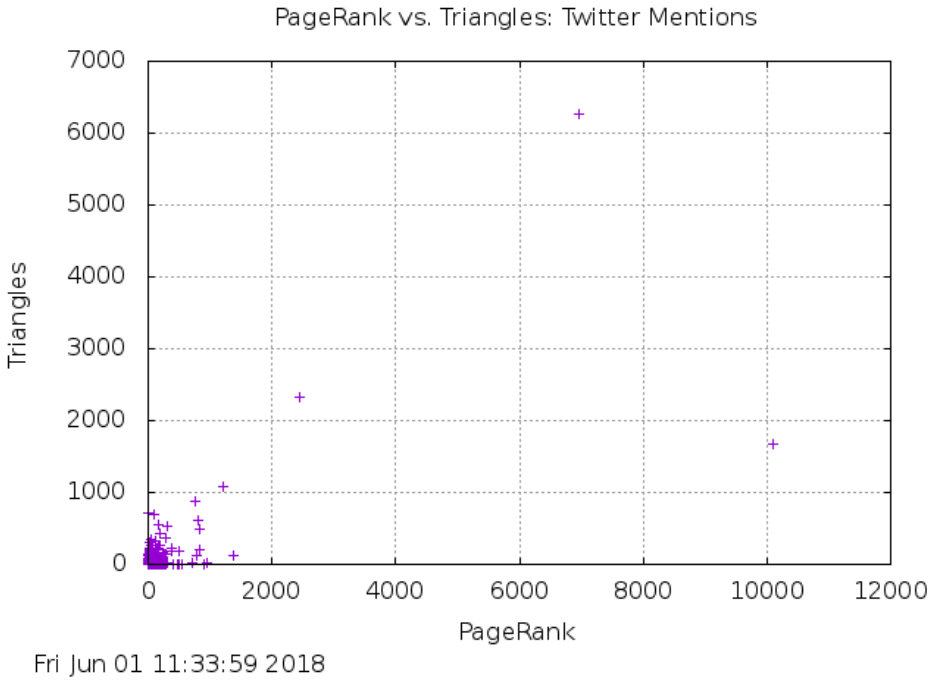
Στην Εικόνα 5.19 φαίνεται μια κυκλική αναπαράσταση όπου πραγματοποιείται με την χρήση του Circular Layout αλγορίθμου. Η κατάταξη των κόμβων γίνεται με την χρήση των

αποτελεσμάτων από τον αλγόριθμο PageRank, όπου το χρώμα και το μέγεθος του κόμβου καθορίζει τον βαθμό του PageRank, δηλαδή από το πράσινο προς το μωβ χρώμα αυξάνεται το PageRank.

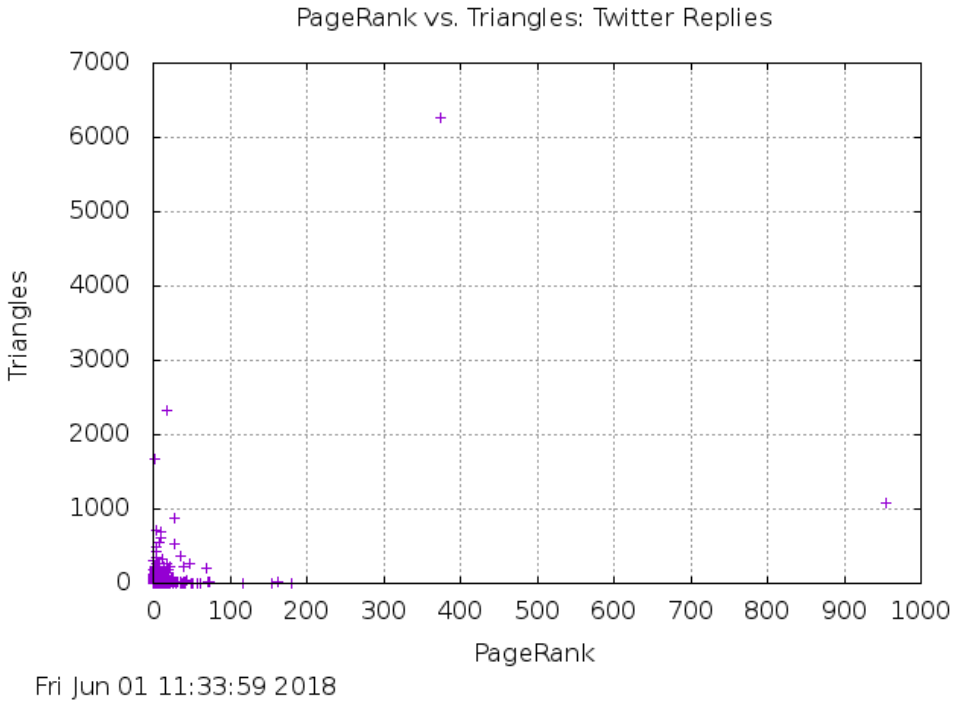


Εικόνα 5.19 Circular Layout με PageRank, Twitter users: Mentions

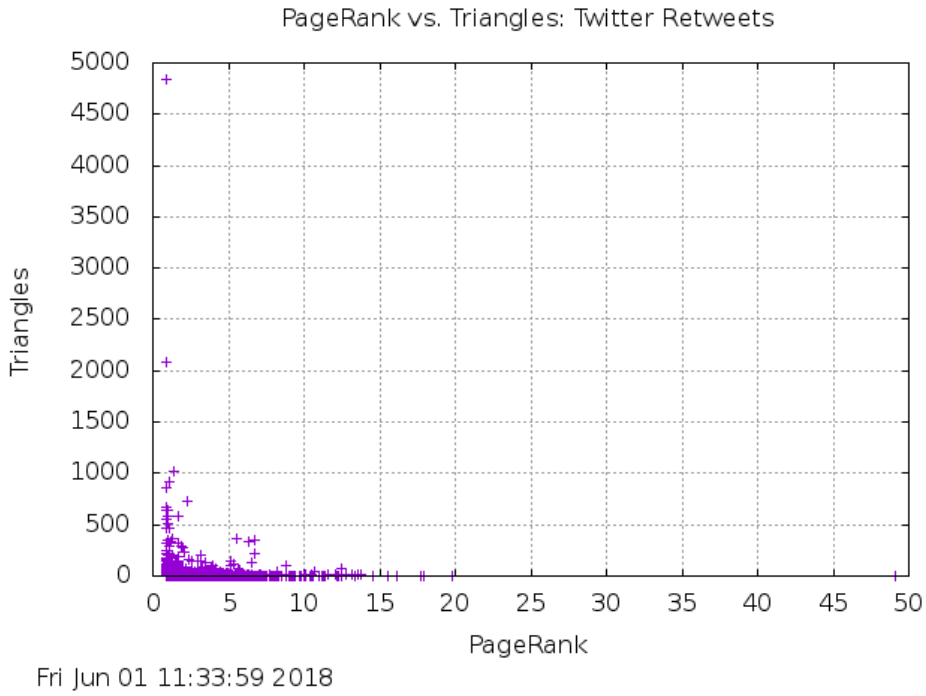
Στην Εικόνα 5.20, φαίνεται ένα καρτεσιανό γράφημα όπου αναπαριστά των αριθμό των τριγώνων όπου διέρχονται από τους κόμβους σε σχέση με τον βαθμό PageRank αυτών. Αφορά τον γράφο “Twitter users: Mentions”. Όπως και στην Υποενότητα 5.1.1, παρατηρείται ότι ο αριθμός των τριγώνων δεν εξαρτάται από το PageRank. Η διαφορά με το αποτέλεσμα της Υποενότητας 5.1.1 είναι ότι εμφανίζεται πολύ μικρός αριθμός τριγώνων και αυτό συμβαίνει λόγω της φύσης των δεδομένων, όπου αφορούν συγκεκριμένη ενέργεια των χρηστών και δεν είναι απλά δεδομένα ενός τυχαίου γράφου. Το ίδιο συμβαίνει και στα καρτεσιανά γραφήματα των γράφων “Twitter users: Replies” και “Twitter users: Retweet”, όπου αναπαρίστανται στην Εικόνα 5.21 και 5.22.



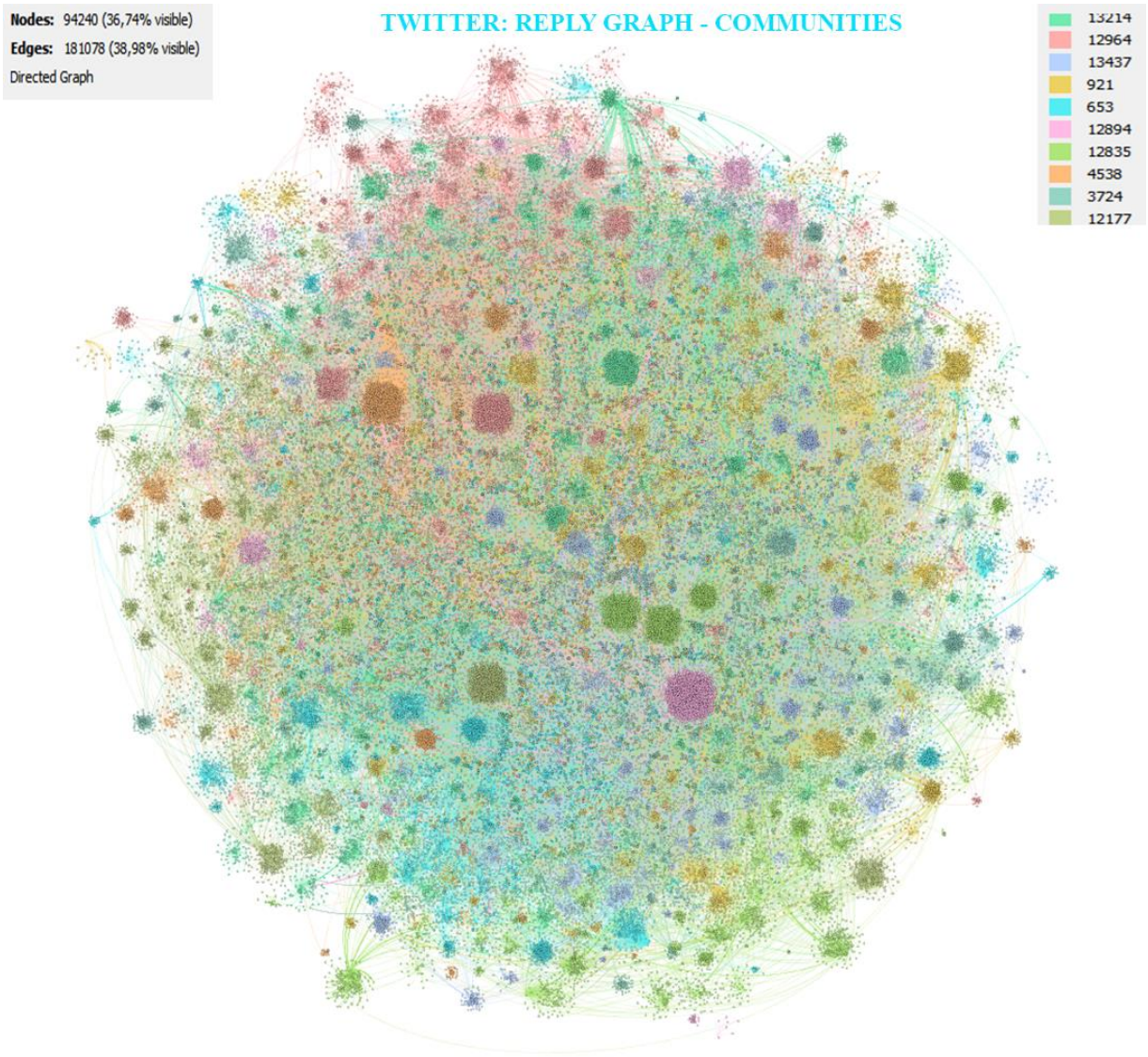
Εικόνα 5.20 Triangles count vs PageRank Twitter:Mentions



Εικόνα 5.21 Triangles count vs PageRank Twitter: Replies



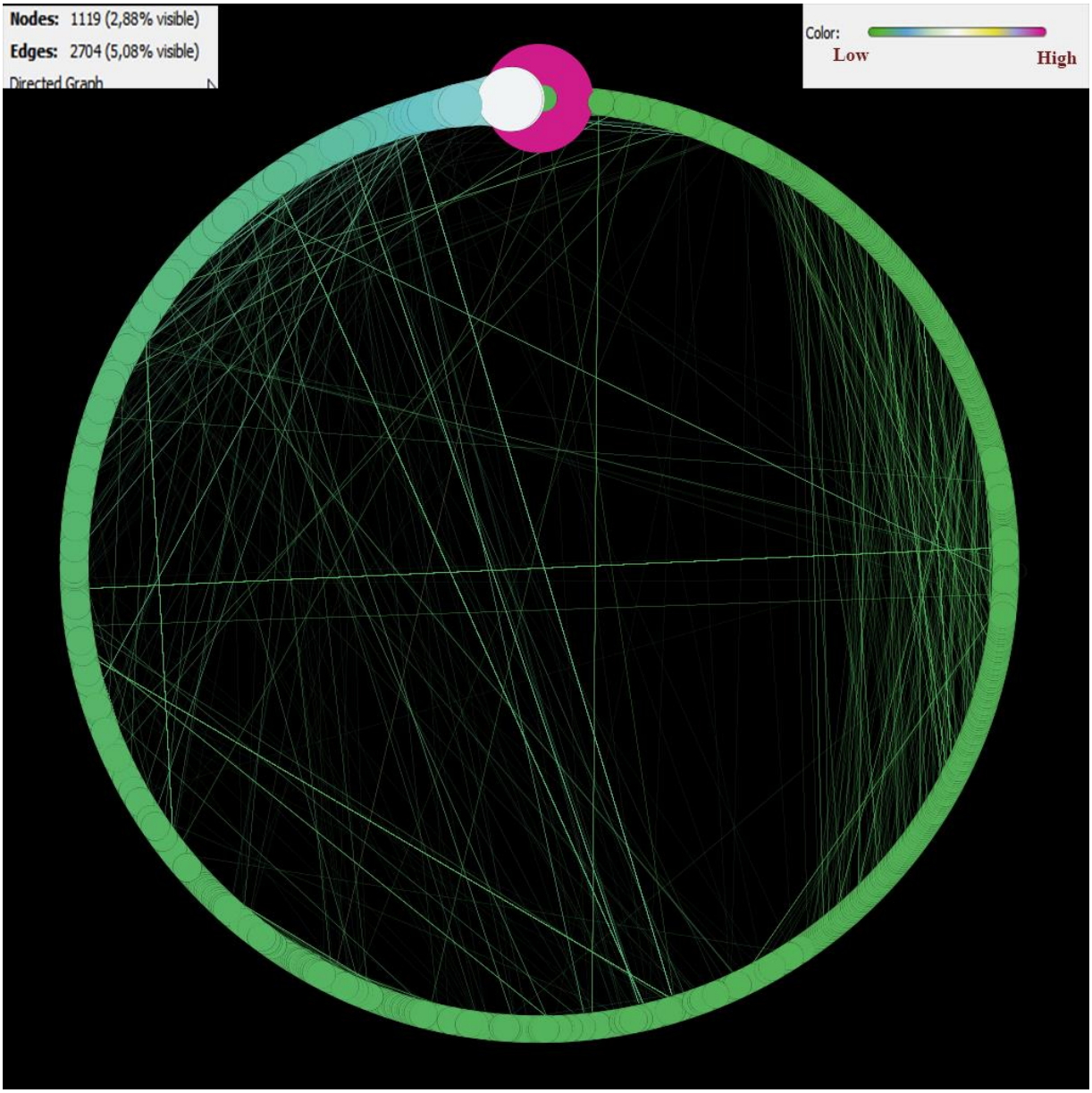
Εικόνα 5.22 Triangles count vs PageRank Twitter: Retweets



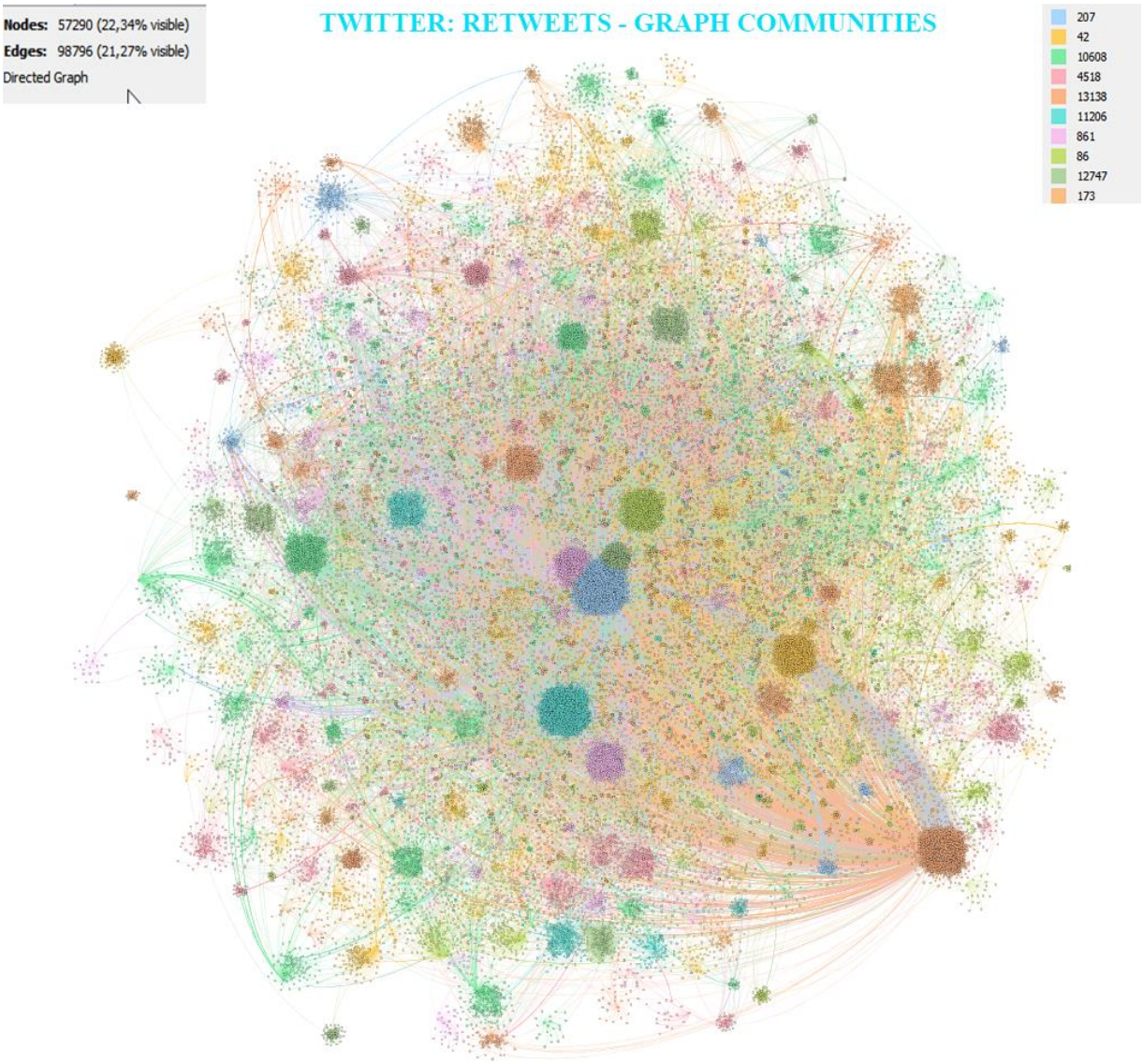
Εικόνα 5.23 Κοινότητες υπογράφου Twitter users: Replies με αλγόριθμο OpenOrd

Στην Εικόνα 5.23, αναπαρίστανται οι δέκα μεγαλύτερες κοινότητες του υπογράφου "Twitters users: Replies" όπου περιέχει τους χρήστες όπου έχουν αλληλεπιδράση "Reply". Αυτό που παρατηρείται είναι ότι υπάρχει πολύ μεγάλη διασπορά των κόμβων, ωστόσο μικρότερη από αυτή της Εικόνας 5.18, με συνέπεια οι κοινότητες να μην είναι ευδιάκριτες. Όπως και στην Εικόνα 5.18, στην αναπαράσταση φαίνεται ο αριθμός των κόμβων που συμμετέχουν και οι ακμές όπου συμμετέχουν στην δημιουργία του.

Στην Εικόνα 5.24 φαίνεται μια κυκλική αναπαράσταση όπου πραγματοποιείται με την χρήση του Circular Layout αλγορίθμου. Η κατάταξη των κόμβων γίνεται με την χρήση των αποτελεσμάτων από τον αλγόριθμο PageRank, όπου το χρώμα και το μέγεθος του κόμβου καθορίζει τον βαθμό του PageRank, δηλαδή από το πράσινο προς το μωβ χρώμα αυξάνεται το PageRank. Παρατηρούμε ότι στην Εικόνα 5.24 σε σχέση με την Εικόνα 5.19, οι κόμβοι έχουν κατά μέσο όρο μεγαλύτερο PageRank.



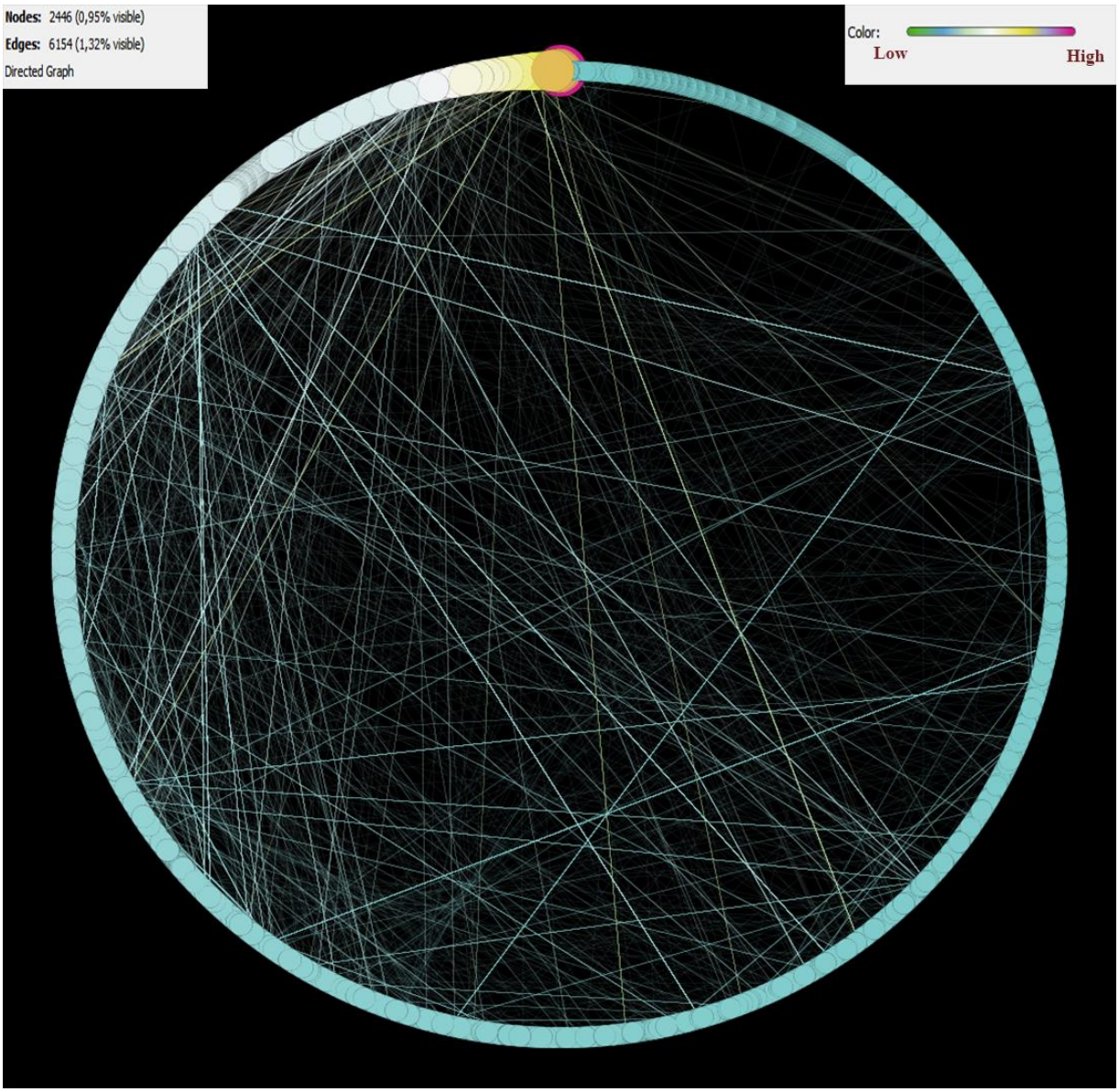
Εικόνα 5.24 Circular Layout με PageRank, Twitter users: Replies



Εικόνα 5.25 Κοινότητες υπογράφου Twitter users: Retweets με αλγόριθμο OpenOrd

Στην Εικόνα 5.25, αναπαρίστανται οι δέκα μεγαλύτερες κοινότητες του υπογράφου "Twitters users: Replies" όπου περιέχει τους χρήστες όπου έχουν αλληλεπίδραση "Retweet". Αυτό που παρατηρείται είναι ότι υπάρχει πολύ μεγάλη διασπορά των κόμβων, ωστόσο μικρότερη από

αυτή της Εικόνας 5.18 αλλά παρόμοια με αυτή της Εικόνας 5.23, με συνέπεια οι κοινότητες να μην είναι ευδιάκριτες. Όπως και στην Εικόνα 5.18, στην αναπαράσταση φαίνεται ο αριθμός των κόμβων που συμμετέχουν και οι ακμές όπου συμμετέχουν στην δημιουργία του.



Εικόνα 5.26 Circular Layout με PageRank, Twitter users: Retweets

Στην Εικόνα 5.26 φαίνεται μια κυκλική αναπαράσταση όπου πραγματοποιείται με την χρήση του Circular Layout αλγορίθμου. Η κατάταξη των κόμβων γίνεται με την χρήση των

αποτελεσμάτων από τον αλγόριθμο PageRank, όπου το χρώμα και το μέγεθος του κόμβου καθορίζει το βαθμό του PageRank, δηλαδή από το πράσινο προς το μωβ χρώμα αυξάνεται το PageRank. Παρατηρούμε ότι στην Εικόνα 5.26 σε σχέση με την Εικόνα 5.24, οι κόμβοι έχουν κατά μέσο όρο μεγαλύτερο PageRank και εμφανίζει διαβαθμισμένη αύξηση του PageRank.

5.3 Συζήτηση

Οι γράφοι βοηθούν στην επιστημονική προσέγγιση και επίλυση πολλών καθημερινών προβλημάτων, αλλά βοηθούν και στην εξέλιξη τομέων όπως ηλεκτρονικό εμπόριο και τον κοινωνικών μέσων δικτύωσης. Τα σώματα ασφαλείας που δρούνε σε παγκόσμιο επίπεδο αλλά και εσωτερικά στη χώρα τους, εφαρμόζουν τεχνικές ανάλυσης γράφων με συνεχής τροφοδότηση δεδομένων για να μπορούνε να προβλέπουν τρομοκρατικές ενέργειες, ή βίαια γεγονότα. Οι κολοσσοί στον χώρο του ηλεκτρονικού εμπορίου, όπως είναι το eBay και το Amazon, συνεχώς προσπαθούν να δώσουνε πρωτότυπες υπηρεσίες και να προβαίνουν σε μείωση χρόνων παράδοσης των προϊόντων. Με την χρήση γράφων μπορούν να βάλουν τα δεδομένα σε ένα πλαίσιο κατανοητό, ώστε να τα αναλύσουν και να βελτιώσουν τις διαδρομές που ακολουθούν τα προϊόντα τους κατά την αποστολή, παράδοση ή να προτείνουν προϊόντα που πιθανώς ο πελάτης να επιθυμεί. Η πλοήγηση με την χρήση εφαρμογών όπως είναι το Google Maps, αποτελεί υλοποίηση ανάλυσης γράφων διότι προτείνει διαδρομές ανάλογα με την κίνηση εκείνη τη στιγμή, τη προτίμηση του χρήστη και το μέσω μεταφοράς. Τέλος τα μέσα κοινωνικής δικτύωσης όπου χρησιμοποιούν σχεδόν σε οτιδήποτε την ανάλυση γράφων, ώστε να τροφοδοτούν συνεχώς τον χρήστη με νέες πληροφορίες, οι οποίες είναι εξατομικευμένες για αυτόν. Για παράδειγμα οι προτάσεις φίλων, σελίδων και διαφημίσεων στο Facebook είναι αποτέλεσμα ανάλυσης γράφων.

6 Συμπεράσματα και τελευταίες σκέψεις

Όπως ήδη έχουμε αντιληφθεί ο τομέας της ανάλυσης γράφων έχει ενσωματωθεί με στον τομέα των "Big Data", διότι αποτελεί έναν τρόπο ανάλυσης δεδομένων που είναι πιο ευέλικτος για την εξαγωγή συμπερασμάτων που μπορεί να διαβάσει και ένας απλός άνθρωπος χωρίς γνώσεις πάνω στην Επιστήμη των Υπολογιστών. Στη καθημερινότητα μας έχει εισέλθει μέσω των μέσων κοινωνικής δικτύωσης, των εφαρμογών πλοήγησης όπως και με άλλους τρόπους που δεν μας είναι αντιληπτό.

Όσον αφορά την εργασία, παρατηρούμε ότι όσο πιο συγκεκριμένα δεδομένα έχουμε, τα οποία έχουν μια μορφή και τα επεξεργαζόμαστε με ανάλογο τρόπο, δηλαδή δεν βλέπουμε απλά ένα τυχαίο γράφημα κόμβων αλλά χρήστες του Twitter που αλληλεπιδρούν με διάφορες ενέργειες, τόσο αλλάζει η μορφή του γράφο στην απεικόνιση. Αυτό συμβαίνει διότι έχουμε διαφορετικής βαρύτητας κόμβους και ακμές, διότι ο κατακερματισμός του σε υπό-γράφους ανάλογα με την ενέργεια που έκανε ο κάθε χρήστης προς κάποιον άλλον δημιουργεί ένα γράφο με ορισμένη την πληροφορία που αναπαριστά δίχως να επηρεάζεται από επιμέρους πληροφορίες που αφορούν τον ίδιο χρήστη. Στην ανάλυση που πραγματοποιήθηκε στα δεδομένα του Facebook, που ήταν απλά ακμές και κόμβοι μεταξύ χρηστών παρατηρούμε ότι τα αποτελέσματα δεν διαφέρουν σε σχέση με έναν τυχαία αναπαραγμένο γράφο, δεν μπορούσε να υπάρχει σαφής εικόνα για το τι αναπαριστούν τα δεδομένα. Για αυτό τον λόγο προέκυψε ασυνήθιστα μεγάλος αριθμός τριγώνων όπου περνούν από τους κόμβους, διότι σε δεδομένα μέσων κοινωνικής δικτύωσης δεν είναι τόσο συνηθισμένο να συναντάμε μεγάλο αριθμό τριγώνων, λόγω της φύσης τους, πάντα με παράγοντα τον όγκο δεδομένων και το δείγμα, διότι σε μια κλειστή ομάδα όπως είναι το αμφιθέατρο ενός τμήματος στο ΤΕΙ Ηπείρου είναι πολύ πιθανό να έχουμε μεγάλο αριθμό τριγώνων αν και πάλι όχι τόσο όσο θα έπρεπε. Κλείνοντας, πιστεύω ότι ο τομέας αυτός θα προχωρήσει κι άλλο στο επίπεδο ενσωμάτωσης σε καθημερινές εφαρμογές, διότι προσφέρει νοημοσύνη ή πιο σωστά, «πρόληψη» όταν συνδυαστεί με εργαλεία που προσφέρονται από το Apache Spark.

Βιβλιογραφία

[1] Apache Software Foundation , 2013. *Apache Hadoop - MapReduce.* [Ηλεκτρονικό]
Available at: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html
[Πρόσβαση 06 Ιούνιος 2018].

[2] Apache Software Foundation Hadoop, 2018. *Apache Hadoop - HDFS.* [Ηλεκτρονικό]
Available at: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html
[Πρόσβαση 06 Ιούνιος 2018].

[3] Apache Software Foundation, 2018. *Apache Hadoop.* [Ηλεκτρονικό]
Available at: http://hadoop.apache.org/docs/r2.8.4/
[Πρόσβαση 06 Ιούνιος 2018].

[4] Apache Software Foundation, 2018. *Apache Hadoop- YARN.* [Ηλεκτρονικό]
Available at: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html
[Πρόσβαση 06 Ιούνιος 2018].

[5] Apache Software Foundation, 2018. *Apache Spark - Cluster Overview.* [Ηλεκτρονικό]
Available at: https://spark.apache.org/docs/latest/cluster-overview.html
[Πρόσβαση 06 Ιούνιος 2018].

[6] Apache Software Foundation, 2018. *Apache Spark - Overview.* [Ηλεκτρονικό]
Available at: http://spark.apache.org/docs/latest/index.html
[Πρόσβαση 07 Ιούνιος 2018].

[7] Apache Software Foundation, 2018. *Apache Spark - Programming Guide.* [Ηλεκτρονικό]
Available at: http://spark.apache.org/docs/latest/rdd-programming-guide.html
[Πρόσβαση 07 Ιούνιος 2018].

[8] Ferguson, M., 2016. *IBM Big Data & Analytics Hub.* [Ηλεκτρονικό]
Available at: http://www.ibmbigdatahub.com/blog/what-graph-analytics
[Πρόσβαση 06 Ιούνιος 2018].

[9] GraphFrames, 2016. *GraohFrames.* [Ηλεκτρονικό]
Available at: https://graphframes.github.io/user-guide.html
[Πρόσβαση 06 Ιούνιος 2018].

[10] Leskovec, J. & Krevl, A., 2014. *SNAP.* [Ηλεκτρονικό]
Available at: http://snap.stanford.edu/data
[Πρόσβαση 10 Ιούνιος 2018].

[11] Malak, M. S. & East, R., 2016. *Spark GraphX in Action,* s.l.: Manning Publications.

[12] Oracle, χ.χ. *Oracle.com.* [Ηλεκτρονικό]
Available at: https://www.oracle.com/big-data/guide/what-is-big-data.html
[Πρόσβαση 06 Ιούνιος 2018].

[13] Page, L., Brin, S., Motwani, R. & Winograd, T., 1999. *The PageRank Citation Ranking: Bringing Order to the Web.* [Ηλεκτρονικό]
Available at: http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf
[Πρόσβαση 06 Ιούνιος 2018].

[14] Ruohonen, K., Tamminem, J., Lee, K.-C. & Piché, R., 2013. *Graph Theory. Tampereen teknillinen yliopisto. Originally titled Graafiteoria, lecture notes translated by Tamminen, J., Lee, K." C. and Piché, R.* [Ηλεκτρονικό]
Available at: http://math.tut.fi/~ruohonen/GT_English.pdf
[Πρόσβαση 07 Ιούνιος 2018].

[15] Sas, χ.χ. *sas.com.* [Ηλεκτρονικό]
Available at: https://www.sas.com/en_us/insights/big-data/what-is-big-data.html
[Πρόσβαση 06 Ιούνιος 2018].

[16] Siddharth, S. & Vassilvitski, S., 2011. *Counting Triangles and the Curse of the Last Reducer. Proceedings.* [Ηλεκτρονικό]
Available at: https://theory.stanford.edu/~sergei/papers/www11-triangles.pdf
[Πρόσβαση 9 Ιούνιος 2018].

[17] SNAP: Network datasets, 2014. *SNAP: Network datasets: Social circles.* [Ηλεκτρονικό]

Available at: https://snap.stanford.edu/data/egonets-Facebook.html
[Πρόσβαση 01 Ιούνιος 2018].

[18] SNAP: Network datasets, 2015. *SNAP: Network datasets: Higgs Twitter Dataset.* [Ηλεκτρονικό]
Available at: https://snap.stanford.edu/data/higgs-twitter.html
[Πρόσβαση 01 Ιούνιος 2018].

[19] Spann, T., 2018. *DZone/ Big Data Zone.* [Ηλεκτρονικό]
Available at: https://dzone.com/articles/scala-vs-python-for-apache-spark
[Πρόσβαση 07 Ιούνιος 2018].

[20] Taufer, M., Bernd, M. & Kunkel, J. M., 2016. *High Performance Computing: ISC High Performance 2016 International Workshops, ExaComm, E-MuCoCoS, HPC-IODC, IXPUG, IWOPH, P^3MA, VHPC, WOPSSS, Frankfurt, Germany, June 19–23, 2016, Revised Selected Papers.* 1η επιμ. s.l.:Springer.

[21] Williams, M., 2017. *DZone/ Big Data Zone - Apache Vs. MapReduce.* [Ηλεκτρονικό]
Available at: https://dzone.com/articles/apache-spark-introduction-and-its-comparison-to-ma
[Πρόσβαση 07 Ιούνιος 2018].

[22] Zhu, X. & Ghahramani, Z., 2002. *Learning from labeled and unlabeled data with label propagation.* [Ηλεκτρονικό]
Available at: http://mlg.eng.cam.ac.uk/zoubin/papers/CMU-CALD-02-107.pdf
[Πρόσβαση 07 Ιούνιος 2018].

ΠΑΡΑΡΤΗΜΑ

A. Κώδικάς Twitter Graph Analytics

```
001 import org.graphframes._
002 import org.apache.spark.sql._
003 import org.apache.spark.sql.functions._
004 import org.apache.spark.sql.types.{StructType, StructField, StringType,
IntegerType};
005 import org.graphframes.GraphFrame
006 import scala.collection.JavaConversions._
007
008 println("\\\\Declaring fucntions///")
009 def nodes_filter(dfin:DataFrame, s:String):DataFrame ={
010 val schema = StructType(List(StructField("src", StringType,
false),StructField("dst", StringType, false),StructField("interaction",
StringType, false)))
011 var e = spark.createDataFrame(sc.emptyRDD[Row], schema)
012 if ((s=="RT")||(s=="MT")||(s=="RE")){
013 e = edges.select("src", "dst",
"interaction").filter(col("interaction")===s)
014 }
015 else{e = dfin}
016 val n1 = e.select("src").distinct()
017 val n2 = e.select("dst").distinct()
018 val n = n1.union(n2).withColumnRenamed("src","name").distinct()
019 val df_tmp = n.withColumn("id", n("name"))
020
021 df_tmp.select(df_tmp("name").cast("String"), df_tmp("id").cast("Int"))
022 }
023 def social_sts(graph_df: DataFrame):DataFrame ={
024 val schema = StructType(List(StructField("id", StringType,
false),StructField("follow", IntegerType, false)))
025 var tmp = spark.createDataFrame(sc.emptyRDD[Row], schema)
026 var tmp2 = spark.createDataFrame(sc.emptyRDD[Row], schema)
027 var follow_df = spark.createDataFrame(sc.emptyRDD[Row], schema)
028 var follower_df = spark.createDataFrame(sc.emptyRDD[Row], schema)
029 var social_df = spark.createDataFrame(sc.emptyRDD[Row], schema)
030 val nodelist = graph_df.select("name").limit(20).collectAsList()
031 var cnt = 0
032 for(i <- nodelist){
033 tmp = (g_social.outDegrees.select("id",
"outDegree").filter(g_social.outDegrees("id")===
nodelist(cnt).getString(0))).toDF("id", "Follow")
034 tmp2 = (g_social.inDegrees.select("id",
"inDegree").filter(g_social.inDegrees("id")===
nodelist(cnt).getString(0))).toDF("id", "Followers")
035 follow_df = follow_df.union(tmp)
036 follower_df = follower_df.union(tmp2)
037 cnt = cnt+1
038 }
039 (follow_df.join(follower_df, Seq("id"))).toDF("id","follow","followers")
040 }
041
042 def splitby(graphdf:DataFrame):DataFrame ={
043 val schema= StructType(List(StructField("label", StringType,
false),StructField("count", IntegerType, false)))
044 var tmp = spark.createDataFrame(sc.emptyRDD[Row], schema)
045 var bigc_nodes = spark.createDataFrame(sc.emptyRDD[Row], schema)
046 var tmp2 = graphdf.groupBy("label").agg(count("label").alias("members"))
047 val lblist= tmp2.select("label").filter("members > 1000").collectAsList()
```

```
048 var cnt =0
049 for(i <- lblist){
050 tmp = graphdf.select("name","id").filter(col("label") ===
lblist(cnt).getLong(0))
051 bigc_nodes = bigc_nodes.union(tmp)
052 cnt = cnt+1
053 }
054 bigc_nodes.toDF("name","id")
055 }
056
057 def cmedgelist(n:DataFrame, e:DataFrame):DataFrame ={
058 val schema= StructType(List(StructField("src", StringType,
false),StructField("dst", StringType, false),StructField("interaction",
StringType, false)))
059 var tmp_src = spark.createDataFrame(sc.emptyRDD[Row], schema)
060 var tmp_dst = spark.createDataFrame(sc.emptyRDD[Row], schema)
061 var edgelist = spark.createDataFrame(sc.emptyRDD[Row], schema)
062 var cnt = 0
063 val nlist = n.select("name").collect.map(_.getString(0))
064 for(i<-nlist){
065 tmp_src = e.select("src","dst","interaction").filter(col("src") ===
nlist(cnt)).repartition(4)
066 tmp_dst = e.select("src","dst","interaction").filter(col("dst")===
nlist(cnt)).repartition(4)
067 edgelist = (((tmp_src.union(tmp_dst))).union(edgelist)).repartition(4)
068 cnt = cnt+1
069 }
070 return edgelist
071 }
072 println("=====Loading graph files.....")
073 val df = spark.read.option("header","false").option("delimiter", "
").csv("hdfs://localhost:54310/project/twitter_activity.txt").toDF("userA",
"userB", "timestamp", "interaction")
074 val edges = df.select("userA" , "userB", "interaction").toDF("src",
"dst", "interaction")
075 println("===== Initializing graphs depending interaction =====")
076 val nrt = nodes_filter(edges,"RT")
077 val e2 = edges.select("dst", "src", "interaction").filter("interaction =
'RT'").toDF("src", "dst", "interaction")
078 val g_rt = GraphFrame(nrt, e2)
079 val nmt = nodes_filter(edges,"MT")
080 val e3 = edges.filter("interaction = 'MT'")
081 val g_mt = GraphFrame(nmt, e3)
082 val nre = nodes_filter(edges,"RE")
083 val e4 = edges.filter("interaction = 'RE'")
084 val g_re = GraphFrame(nre, e4)
085 val nodes = nodes_filter(edges,"g")
086 val uedges = (e2.union(e3)).union(e4)
087 uedges.persist()
088 val g = GraphFrame(nodes, uedges)
089 val pr_g = g.pageRank.resetProbability(0.15).tol(0.01).run()
090 val result_pr_g = pr_g.vertices.sort(desc("pageRank"))
091 val pr_rt = g_rt.pageRank.resetProbability(0.15).tol(0.01).run()
092 val result_pr_rt = pr_rt.vertices.sort(desc("pagerank"))
093 val pr_mt = g_mt.pageRank.resetProbability(0.15).tol(0.01).run()
094 val result_pr_mt = pr_mt.vertices.sort(desc("pagerank"))
095 val pr_re = g_re.pageRank.resetProbability(0.15).tol(0.01).run()
```

```
096 val result_pr_re = pr_re.vertices.sort(desc("pagerank"))
097
098 println("Importing Social edgelist and creating follow/follower graph")
099 val twitter_social_df =
spark.read.option("header","false").option("delimiter", "
").csv("hdfs://localhost:54310/project/twitter_social.edgelist").toDF("src",
"dst")
100 val social_nodes = nodes_filter(twitter_social_df,"tw")
101 val g_social = GraphFrame(social_nodes, twitter_social_df)
102 println("===Communities: Label Propagation Algorithm===")
103 val community_g = pr_g.labelPropagation.maxIter(10).run()
104 community_g.sort(desc("pagerank")).show()
105 val community_rt = pr_rt.labelPropagation.maxIter(10).run()
106 community_rt.sort(desc("pagerank")).show()
107 val community_re = pr_re.labelPropagation.maxIter(10).run()
108 community_re.sort(desc("pagerank")).show()
109 val community_mt = pr_mt.labelPropagation.maxIter(10).run()
110 community_mt.sort(desc("pagerank")).show()
111 println("===Triangle counting per graph===")
112 val tri_g = g.triangleCount.run()
113 tri_g.persist()
114 val tri_rt = g_rt.triangleCount.run()
115 tri_rt.persist()
116 val tri_mt = g_mt.triangleCount.run()
117 tri_mt.persist()
118 val tri_re = g_re.triangleCount.run()
119 tri_re.persist()
120 println("Initializing Plot scatter Triangles Count vs PageRank")
121 val plotscat_g = result_pr_g.join(tri_g, Seq("name","id"))
122 val plotscat_rt = result_pr_rt.join(tri_rt, Seq("name","id"))
123 val plotscat_mt = result_pr_mt.join(tri_mt, Seq("name","id"))
124 val plotscat_re = result_pr_re.join(tri_mt, Seq("name","id"))
125 println("===Initializing graph stats...")
126 val graph_stats_g = community_g.join(plotscat_g,
Seq("name","id","pagerank"))
127 val graph_stats_rt = community_rt.join(plotscat_rt,
Seq("name","id","pagerank"))
128 val graph_stats_re = community_re.join(plotscat_re,
Seq("name","id","pagerank"))
129 val graph_stats_mt = community_mt.join(plotscat_mt,
Seq("name","id","pagerank"))
130 println("===Initializing largest communities graph...")
131 val new_nrt= splitby(community_rt)
132 val new_nmt= splitby(community_mt)
133 val new_nre= splitby(community_re)
134 val new_nodes= ((new_nre.union(new_nmt)).union(new_nrt)).distinct()
135 new_nodes.persist()
136 val new_edges = cmedgelist(new_nodes,uedges)
137
138 println("===Finding Followers and Follows from nodes with the highest
pagerank @ every subgraph RT,RE,MT===")
139 println("===Social Stats for RT Graph top 20 PageRank===")
140 var social_rt = social_sts(result_pr_rt)
141 println("===Social Stats for RE Graph top 20 PageRank===")
142 var social_re = social_sts(result_pr_re)
143 println("===Social Stats for MT Graph top 20 PageRank===")
144 var social_mt = social_sts(result_pr_mt)
```

```
145 println("***Top nodes depending Social Stats...")
146 social_rt.persist()
147 social_re.persist()
148 social_mt.persist()
149 var top_social =
((social_rt.union(social_re)).union(social_mt)).distinct()
150 top_social.persist()
151 top_social.show()
152
153 println("====Exporting data to CSV")
154
pr_g.edges.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databricks.
spark.csv").option("header","true").save("hdfs://localhost:54310/project/data
csv/graphs/g_edges")
155
pr_rt.edges.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databricks
.spark.csv").option("header","true").save("hdfs://localhost:54310/project/dat
acsv/graphs/rt_edges")
156
pr_mt.edges.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databricks
.spark.csv").option("header","true").save("hdfs://localhost:54310/project/dat
acsv/graphs/mt_edges")
157
pr_re.edges.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databricks
.spark.csv").option("header","true").save("hdfs://localhost:54310/project/dat
acsv/graphs/re_edges")
158 println("==Exporting graph stats to csv...")
159
graph_stats_g.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databric
ks.spark.csv").option("header","true").save("hdfs://localhost:54310/project/d
atacsv/graphs/g_nodes")
160
graph_stats_rt.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databri
cks.spark.csv").option("header","true").save("hdfs://localhost:54310/project/
datacsv/graphs/rt_nodes")
161
graph_stats_re.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databri
cks.spark.csv").option("header","true").save("hdfs://localhost:54310/project/
datacsv/graphs/re_nodes")
162
graph_stats_mt.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databri
cks.spark.csv").option("header","true").save("hdfs://localhost:54310/project/
datacsv/graphs/mt_nodes")
163 println("==Exporting plot scatters to csv...")
164
plotscat_g.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databricks.
spark.csv").option("header","true").save("hdfs://localhost:54310/project/data
csv/plotscatters/g")
165
plotscat_rt.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databricks
.spark.csv").option("header","true").save("hdfs://localhost:54310/project/dat
acsv/plotscatters/rt")
166
plotscat_mt.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databricks
.spark.csv").option("header","true").save("hdfs://localhost:54310/project/dat
acsv/plotscatters/mt")
```

```
167
plotscat_re.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databricks
.spark.csv").option("header","true").save("hdfs://localhost:54310/project/dat
acsv/plotscatters/re")
168
top_social.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databricks.
spark.csv").option("header","true").save("hdfs://localhost:54310/project/data
csv/social_stats")
169
170 println("===Shortest Paths of Top 20 PageRank nodes===")
171 val nrt2  =
nrt.select(col("name").cast("String"),col("id").cast("String"))
172 val g_rt2 = GraphFrame(nrt2,e2)
173 val pr_rt2 = g_rt2.pageRank.resetProbability(0.15).tol(0.01).run()
174 val array_rt =
result_pr_rt.select("name").limit(20).collect.map(_.getString(0))
175 var short_path_rt = pr_rt2.shortestPaths.landmarks(array_rt).run()
176 val flatt =
short_path_rt.select(functions.explode(short_path_rt("distances"))).toDF("key
","value")
```

B. Κώδικας Facebook Graph Analytics

```
01 import org.graphframes._
02 import org.apache.spark.sql._
03 import org.apache.spark.sql.functions._
04 import org.apache.spark.sql.types.{StructType, StructField, StringType,
IntegerType};
05
06 val customSchema = StructType(Array(StructField("src", IntegerType,
true),StructField("dst", IntegerType, true)))
07 val df = spark.read.option("header","true").option("delimiter", "
").schema(customSchema).csv("hdfs://localhost:54310/project/facebook_combined
.txt")
08 val edges = df.select("src","dst")
09 val n1 = edges.select("src").distinct()
10 val n2 = edges.select("dst").distinct()
11 val n = n1.union(n2).withColumnRenamed("src","name").distinct()
12 val nodes = n.withColumn("id", n("name"))
13 val g1 = GraphFrame(nodes, edges)
14 val k = g1.degrees.sort(desc("degree"))
15 k.show()
16 val pr2 = g1.pageRank.resetProbability(0.15).maxIter(10).run()
17 pr2.vertices.show()
18 val pr3 = pr2.vertices.sort(desc("pagerank"))
19 pr3.show()
20 val tri = g1.triangleCount.run()
21 tri.persist()
22 tri.sort(desc("count")).limit(5).show()
23 val plotscat = pr2.vertices.join(tri, Seq("name", "id"))
24 plotscat.limit(10).show
25 val community_fb = pr2.labelPropagation.maxIter(10).run()
26 community_fb.show()
27 val stats_fb = community_fb.join(plotscat, Seq("name","id","pagerank"))
28
29
plotscat.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databricks.sp
ark.csv").option("header","true").save("hdfs://localhost:54310/project/datacs
v/plotscat/fb")
30
pr2.edges.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databricks.s
park.csv").option("header","true").save("hdfs://localhost:54310/project/datac
sv/graphs/fb_edges")
31
stats_fb.coalesce(1).write.mode(SaveMode.Overwrite).format("com.databricks.sp
ark.csv").option("header","true").save("hdfs://localhost:54310/project/datacs
v/graphs/fb_nodes")
```

Γ. Κώδικας Gnuplot plot scatters

```
01 #Filename: plot_fb.cmd
02 set datafile separator ","
03 set term png
04 set output "g_fb.png"
05 #
06 # Add titles and labels.
07 #
08 set xlabel "PageRank"
09 set ylabel "Triangles"
10 set title "PageRank vs. Triangles: Facebook"
11 unset key
12 #
13 # Add grid lines.
14 #
15 set grid
16 set size ratio 600
17 #
18 # Timestamp the plot.
19 #
20 set timestamp
21 plot 'datacsv/plotscatters/fb/part-00000-6c3970c1-d502-40e7-92df-
706fc56a4223-c000.csv' using 3:4
```

```
01 #Filename: plot_g.cmd
02 set datafile separator ","
03 set term png
04 set output "g_tw.png"
05 #
06 # Add titles and labels.
07 #
08 set xlabel "PageRank"
09 set ylabel "Triangles"
10 set title "PageRank vs. Triangles: Twitter"
11 unset key
12 #
13 # Add grid lines.
14 #
15 set grid
16 set size ratio 600
17 #
18 # Timestamp the plot.
19 #
20 set timestamp
21 plot 'datacsv/plotscatters/g/part-00000-8993f15e-7c1a-4375-afe0-
52dc81a6b617-c000.csv' using 3:4
```

```
01 #Filename: plot_re.cmd
02 set datafile separator ","
03 set term png
04 set output "g_re.png"
05 #
06 # Add titles and labels.
07 #
08 set xlabel "PageRank"
09 set ylabel "Triangles"
10 set title "PageRank vs. Triangles: Twitter Replies"
11 unset key
12 #
13 # Add grid lines.
14 #
15 set grid
16 set size ratio 600
17 #
18 # Timestamp the plot.
19 #
20 set timestamp
21 plot 'datacsv/plotscatters/re/part-00000-396db1f6-8a88-46be-bf05-
cb592ccad35c-c000.csv' using 3:4
```

```
01 #Filename: plot_mt.cmd
02 set datafile separator ","
03 set term png
04 set output "g_mt.png"
05 #
06 # Add titles and labels.
07 #
08 set xlabel "PageRank"
09 set ylabel "Triangles"
10 set title "PageRank vs. Triangles: Twitter Mentions"
11 unset key
12 #
13 # Add grid lines.
14 #
15 set grid
16 set size ratio 600
17 #
18 # Timestamp the plot.
19 #
20 set timestamp
21 plot 'datacsv/plotscatters/mt/part-00000-b96aa61b-6504-42ff-b6a6-
8d4eac7f9796-c000.csv' using 3:4
```