Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μεταπτυχιακή Διατριβή
στα Πληροφοριακά Συστήματα

Αποδοτικές Τεχνικές Εύρεσης Κοινωνικών Ομάδων σε Δίκτυα Κοινωνικής Δικτύωσης

Βασίλειος Τριζώνης

Επιβλέπων Καθηγητής
Ματθαίος Δαμίγος

Σεπτέμβριος 2012

Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Αποδοτικές Τεχνικές Εύρεσης Κοινωνικών Ομάδων σε Δίκτυα Κοινωνικής Δικτύωσης

Βασίλειος Τριζώνης

Επιβλέπων Καθηγητής
Ματθαίος Δαμίγος

Η παρούσα μεταπτυχιακή διατριβή υποβλήθηκε
προς μερική εκπλήρωση των απαιτήσεων για απόκτηση

μεταπτυχιακού τίτλου σπουδών
στα Πληροφοριακά Συστήματα

από τη Σχολή Θετικών και Εφαρμοσμένων Επιστημών
του Ανοικτού Πανεπιστημίου Κύπρου

Σεπτέμβριος 2012

Περίληψη

Η παρούσα μεταπτυχιακή διατριβή πραγματεύεται αποδοτικές τεχνικές εύρεσης κοινωνικών ομάδων σε δίκτυα κοινωνικής δικτύωσης. Ο κύριος στόχος της είναι η αποδοτική αντιμετώπιση του προβλήματος εύρεσης κοινωνικών ομάδων εντός κοινωνικού δικτύου. Αναγνωρίζοντας τον μεγάλο όγκο δεδομένων που παράγονται από ένα κοινωνικό δίκτυο, γίνεται χρήση των κατανεμημένων συστημάτων ώστε να μειωθεί ο χρόνος εύρεσης των κοινωνικών ομάδων και κατ' επέκταση να βελτιωθεί η αποδοτικότητα της επίλυσης του προβλήματος.

Όσον αφορά την εξέλιξη της επιστήμης των υπολογιστικών συστημάτων, τα τελευταία χρόνια έχει γίνει αρκετή έρευνα στην ανάπτυξη μεθόδων και αλγορίθμων που έχουν σχέση με τα κοινωνικά δίκτυα. Λειτουργώντας ως πρόκληση για την υλοποίηση της παρούσας διατριβής, η σύγχρονη προσέγγιση της διερεύνησης κοινωνικών ομάδων εντός μεγάλου όγκου δεδομένων, οδήγησε στην εφαρμογή και την υλοποίηση των θεωρητικών αλγορίθμων στην γλώσσα της πληροφορικής με έναν τρόπο ανεξάρτητο των αρχικών ερευνητών και γι' αυτό και εξίσου έγκυρο και ισχυρά επιστημονικό.

Η μεταπτυχιακή διατριβή ξεκινά με την παρουσίαση παραδειγμάτων αναπαράστασης συγκεκριμένων δικτύων τα οποία και μοντελοποιούνται με τη χρήση κοινωνικών γράφων, ένα μεθοδολογικό εργαλείο που διερευνάται και χρησιμοποιείται διεξοδικά. Στη συνέχεια, υλοποιείται ένας επιλεγμένος από την σχετική βιβλιογραφία αλγόριθμος σε κατανεμημένο σύστημα και αξιολογείται η χρήση του ως προς την αποδοτικότητά της. Τα αποτελέσματα των πειραμάτων δείχνουν το πόσο σημαντική μπορεί να είναι η βελτίωση που προσφέρει η χρήση κατανεμημένων συστημάτων στη διερεύνηση κοινωνικών ομάδων και σχέσεων εντός μεγάλου όγκου δεδομένων.

Χρησιμοποιώντας ως βάση τη σύγχρονη διεθνή βιβλιογραφία [28] αυτή η διατριβή προσπαθεί να ενισχύσει τα πρώτα αποτελέσματα των ερευνών πάνω στο σχετικό γνωστικό πεδίο, μέσα από τη δική της, παράλληλη και ανεξάρτητη ερευνητική διαδικασία. Τα αποτελέσματά της επιβεβαιώνονται από τα ερευνητικά ευρήματα και δείχνουν ότι με τη χρήση παράλληλων συστημάτων όσο αυξάνονται οι υπολογιστικοί κόμβοι (computing nodes) αυξάνεται και η ταχύτητα ανακάλυψης κοινωνικών σχέσεων σε κοινωνικούς γράφους μεγάλους τόσο σε όγκο δεδομένων όσο και σε πολυπλοκότητα.

Λέξεις Κλειδιά: Κοινωνικά Δίκτυα, Κοινωνικοί Γράφοι, Κατανεμημένα Συστήματα, Παραλληλοποίηση εργασιών, Hadoop, Map Reduce, NodeIterator++, Twitter, BerkStan σύνολα δεδομένων.

Summary

Current post graduate dissertation deals with applying effective techniques to identify social groups within social networks. It is known that large social networks with millions of nodes and relations such as Twitter or Facebook produce huge amount of data and the only efficient way to find social relationships through them is with the use of distributed systems.

The evolution of the computer science last years has produced a lot of research on the development of methods and algorithms that are related to the social networks. In this dissertation, the contemporary notion of finding social groups into large volume data, led to the implementation of theoretical algorithms into working software in a way vastly differing from the initial researchers' perspective and for that reason equally valid and scientifically powerful.

First, an effort was made for the visualization as graphs of two specific social networks (Twitter, BerkStan) and then, an algorithm chosen from relative literature was implemented at a distributed system (Hadoop). The results were evaluated in terms of efficiency of applying the particular algorithm at the specific datasets.

The results of the experiments illustrate the improvement of the effectiveness of the algorithm which finds out social groups within large datasets, using distributed computing architecture. The dissertation was based upon contemporary international literature [28], the results of which were reinforced through this parallel work. These results show that with the use of distributed systems an increase on the number of computing nodes, thus an increase in the parallelization of the algorithm execution plan, greatly decreases running time for mining social groups from within huge and complex social graphs.

Key - Words: Social Networks, Social Graphs, Distributed Systems, Parallelize works, Hadoop, Map Reduce, NodeIterator++, Twitter, BerkStan datasets.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά όλους όσους με τον έναν ή τον άλλο τρόπο συνέβαλαν στην εκπόνηση αυτής της μεταπτυχιακής διατριβής.

Πιο συγκεκριμένα, οφείλω ευχαριστίες στον επιβλέποντα καθηγητή μου Ματθαίο Δαμίγο και τον συνεργάτη του Νίκο Στασινόπουλο για την πολύτιμη βοήθειά τους και την άμεση ανταπόκρισή τους σε κάθε φάση της εργασίας αυτής.

Ακόμη δεν ξεχνώ στο σημείο αυτό να αναφέρω τον συμφοιτητή μου Στέλιο Μουλακάκη που με συντρόφεψε και με βοήθησε στο να ξεπεράσω τις δύσκολες στιγμές αυτής της προσπάθειας.

Τέλος, ευχαριστώ τη σύζυγό μου Ελευθερία και τις κόρες μου Στέλλα –3- και Μαρία -6- ετών που με υπομονή περίμεναν πότε ο μπαμπάς θα τελειώσει το διάβασμα για να παίξει μαζί τους.

Περιεχόμενα

Κεφάλαιο 1
Εισαγωγή

Ως κοινωνικό δίκτυο θεωρούμε μια δομή που αποτελείται από άτομα ή οργανισμούς τα οποία ονομάζονται «κόμβοι» και είναι συνδεδεμένα με έναν ή περισσότερους συγκεκριμένους τύπους αλληλεξάρτησης, όπως είναι η φιλία, η συγγένεια, το κοινό συμφέρον, η οικονομική σχέση ή οποιαδήποτε άλλη πιθανή σχέση διασύνδεσης.

1.1 Κίνητρο

Ως κίνητρο για την ενασχόλησή μας με το συγκεκριμένο θέμα θεωρούμε την πρόκληση να ενασχοληθεί κανείς με την αιχμή της τεχνολογίας και των παγκόσμιων εξελίξεων στο χώρο του διαδικτύου σε σχέση με την έρευνα των κοινωνικών δικτύων, των σχέσεων τους, των ιδιοτήτων τους και της αναπαράστασής τους σε γράφους.

Όσον αφορά την εξέλιξη της επιστήμης των υπολογιστικών συστημάτων, τα τελευταία χρόνια έχει γίνει αρκετή έρευνα στην ανάπτυξη μεθόδων και αλγορίθμων που έχουν σχέση με τα

κοινωνικά δίκτυα. Στην παρούσα μεταπτυχιακή διατριβή ζητείται η υλοποίηση αλγόριθμου map reduce με χρήση «big data», δηλαδή αρχεία μεγάλου όγκου δεδομένων, όπως τελευταία αποκαλούνται, κάτι νέο και καινοτόμο στην σκέψη και την υλοποίησή του.

Το σημαντικότερο τρέχον ερευνητικό ερώτημα σε σχέση με τα κοινωνικά δίκτυα είναι η ανάλυση της κοινωνικότητας τους, η οποία μελετάται μέσω της ανάλυσης των "σχέσεων" στα on line συστήματα κοινωνικής δικτύωσης (βλ. Twitter, Google+, Facebook, κ.λπ.), με σκοπό τον προσδιορισμό των κοινοτήτων που τα απαρτίζουν τόσο όσον αφορά τα ποιοτικά, όσο και, κυρίως σε πρώτη φάση, τα ποσοτικά τους χαρακτηριστικά.

1.2 Στόχος

Ο κύριος στόχος της συγκεκριμένης μεταπτυχιακής διατριβής είναι η συμμετοχή της στην προσπάθεια βελτιστοποίησης των αλγορίθμων εύρεσης κοινωνικών ομάδων. Ειδικότερα, αφενός μελετά τις κυριότερες μεθόδους καθορισμού κοινωνικών ομάδων που απορρέουν από ένα κοινωνικό δίκτυο που αναπαριστάται μέσω γράφου και προσπαθεί να αποφασίσει για την προσφορότερη μέθοδο (αλγόριθμο) αναζήτησης ομάδων, αφετέρου, αξιολογεί τη χρήση των κατανεμημένων συστημάτων (cloud computing αλγόριθμοι) ώστε να συνεισφέρει στη βελτιστοποίηση της χρήσης τους σε μεγάλο όγκο δεδομένων (large datasets).

1.3 Μεθοδολογία

Πιο συγκεκριμένα η μεταπτυχιακή διατριβή υλοποιείται βασιζόμενη στα παρακάτω βήματα:

1. Παρουσίαση παραδειγμάτων αναπαράστασης δικτύων (Twitter, BerkStan) ως γράφους, και μοντελοποίηση της συμπεριφοράς τους με τη μελέτη των βασικών ιδιοτήτων των γράφων.

2. Παρουσίαση των βασικότερων προσεγγίσεων ανάλυσης των κοινωνικών δικτύων με τη χρήση γράφων.

3. Υλοποίηση μιας συγκεκριμένης προσέγγισης (βάσει του BerkStan dataset) σε κατανεμημένα συστήματα (Hadoop) και αξιολόγηση της χρήσης της ως προς την αποδοτικότητα επιλεγμένου αλγορίθμου.

Στα αναμενόμενα αποτελέσματα της συγκεκριμένης μεταπτυχιακής διατριβής συμπεριλαμβάνονται η λεπτομερής παρουσίαση των βασικών προσεγγίσεων εύρεσης κοινωνικών ομάδων σε δίκτυα κοινωνικής δικτύωσης, η αξιολόγησή τους και η ανάπτυξη ενός επιλεγμένου και βέλτιστου για το συγκεκριμένο δίκτυο αλγόριθμου στο πλαίσιο Map-Reduce, ώστε να δοθεί η δυνατότητα αξιολόγησης της χρήσης των κατανεμημένων συστημάτων για την βελτίωση της απόδοσης των αλγορίθμων εύρεσης ομάδων σε μεγάλο όγκο δεδομένων.

1.4 Οργάνωση

Η μεταπτυχιακή διατριβή αποτελείται από έξι τμήματα: Την εισαγωγή στην οποία περιγράφεται αδρά ο στόχος. Η μεθοδολογία και το περιεχόμενο της διατριβής ακολουθεί το κεφάλαιο 2 στο οποίο παρουσιάζεται το θεωρητικό υπόβαθρο ταυτόχρονα με την παρουσίαση των μεθοδολογικών εργαλείων. Έτσι, γίνεται μια σχετική βιβλιογραφική επισκόπηση με τη βασική βιβλιογραφία που άπτεται του θέματος και στη συνέχεια αναλύονται τα Κοινωνικά Δίκτυα ως προς το περιεχόμενο, τα χαρακτηριστικά τους, τις ιδιότητές τους και την γραφική τους αναπαράσταση (γράφους). Ακόμη παρουσιάζεται το εργαλείο ανάλυσης των κοινωνικών δικτύων Hadoop, περιγράφεται το ιστορικό του όπως και η εσωτερική του δομή έως και τις αδυναμίες του. Στο ίδιο Κεφάλαιο περιγράφονται τα datasets που θα μελετηθούν, δηλαδή το Twitter dataset και το BerkStan dataset, το οποία αναλύονται επαρκώς σε επίπεδο στατιστικών στοιχείων, ενώ παράλληλα αναπαριστώνται και γραφικά με το εργαλείο Pegasus (παραγωγή γράφου Twitter dataset).

Στο Κεφάλαιο 3 προχωράμε σε μοντελοποίηση των δεδομένων με την καταγραφή των τρόπων που μπορεί αυτό να γίνει, μέσω της ανάλυσης των γράφων και των μετρικών που χρησιμοποιούνται στη μέτρηση των ιδιοτήτων των κοινωνικών δικτύων. Εδώ παρουσιάζονται ακόμη οι βασικοί αλγόριθμοι που χρησιμοποιούνται για τον εντοπισμό των κοινοτήτων των Κοινωνικών δικτύων, ενώ τεκμηριώνεται και η επιλογή του NodeIterator++ σε περιβάλλον MapReduce για την περίπτωση της παρούσης διατριβής. Στο Κεφάλαιο 4 παρουσιάζεται η σχεδίαση και η υλοποίηση του υφιστάμενου πειραματικού συστήματος, καθώς και οι τεχνολογίες που χρησιμοποιήθηκαν. Στο Κεφάλαιο 5 περιγράφεται η υλοποίηση των αναλύσεων και τα αποτελέσματά τους, ενώ στο Κεφάλαιο 6 αναπτύσσονται και τεκμηριώνονται τα συμπεράσματα που προκύπτουν από την παρούσα διατριβή καθώς και η πιθανή μελλοντική έρευνα που προκύπτει από την ανάπτυξη της παρούσας μεταπτυχιακής διατριβής. Η

μεταπτυχιακή διατριβή ολοκληρώνεται με την παράθεση της σχετικής βιβλιογραφίας και την επισύναψη των αντίστοιχων Παραρτημάτων.

1.5 Συνεισφορά

Τα κοινωνικά δίκτυα (Twitter, Google+, Facebook, κ.λπ.) μπορούν να αναπαρασταθούν ως γράφοι των οποίων τα χαρακτηριστικά[1] μελετάμε στην παρούσα μεταπτυχιακή διατριβή ιδιαίτερα όσον αφορά την κοινωνικότητά τους, δηλαδή τη δημιουργία στο εσωτερικό τους κοινωνικών ομάδων - ενοτήτων. Συνεισφέροντας στο γνωστικό πεδίο της επιστήμης των υπολογιστών, η παρούσα μεταπτυχιακή διατριβή συστηματοποιεί και ταξινομεί την υπάρχουσα βιβλιογραφία σχετικά με την λειτουργία και ταξινόμηση των γράφων των κοινωνικών δικτύων, παρουσιάζει με συγκριτικό τρόπο τους υπάρχοντες αλγορίθμους και φτάνει σε σημαντικά συμπεράσματα όσον αφορά την επιλογή του ερευνητή – ενδιαφερόμενου ανάμεσα στα εργαλεία ανάλυσης των κατευθυνόμενων -και μη- γράφων με τη χρήση κατανεμημένων συστημάτων.

[1] Το δίκτυο Twitter αναλύθηκε διεξοδικά και για πρώτη φορά από τους Haewoon et al (13), των οποίων κάποια συμπεράσματα ελέγχουμε στην παρούσα μεταπτυχιακή διατριβή

Κεφάλαιο 2
Θεωρητικό υπόβαθρο - εργαλεία

2.1 Βιβλιογραφική Επισκόπηση

Η δομή των κοινοτήτων μέσα στα κοινωνικά δίκτυα από τη σκοπιά της Πληροφορικής επιστήμης μελετήθηκε σχετικά πρόσφατα. Οι κοινότητες αποτελούν υποσύνολα του δικτύου και είναι περισσότερο συνδεδεμένες στο εσωτερικό τους απ' ότι είναι το δίκτυο συνολικά (M. Newman et M. Girvan, 2004, A. Rajaraman & J. Ullman, [22]). Αυτό πυροδότησε την μελέτη αλγορίθμων για ανίχνευση κοινοτήτων μέσα στα δίκτυα. Οι πρώτες προσεγγίσεις χώριζαν τους κόμβους σε κοινότητες μεγιστοποιώντας τις σχέσεις στο εσωτερικό τους και κρατώντας μικρό τον αριθμό των σχέσεων ανάμεσα στις κοινότητες. Σε αυτές τις πρώτες προσεγγίσεις, προτάθηκαν αλγόριθμοι (B. Kernighan et S. Li [16] και A. Pothen et al [25]) οι οποίοι αρχικά χωρίζουν το γράφο σε δυο κοινότητες, έπειτα τις χωρίζουν ξανά και ξανά ώσπου να φτάσουν στον αριθμό των κοινοτήτων που επέλεξε ο χρήστης. Το

μειονέκτημα αυτών των αλγορίθμων είναι η εκ των προτέρων προεπιλογή του τελικού αριθμού των κοινοτήτων.

Άλλες προσεγγίσεις εξέτασαν την ανίχνευση πολλαπλών, μη-επικαλυπτόμενων κοινοτήτων, οι οποίες προσπαθούν να διαχωρίσουν τους κόμβους σε μοναδικές μη-επικαλυπτόμενες (non-overlapping) κοινότητες. Ο G. Palla [24] πρότεινε την χρήση των k-cliques (Clique Percolation Method) για την ανίχνευση κοινοτήτων διαφορετικών μεγεθών, ενώ οι J. Baumes et al [04] πρότειναν μια παρόμοια προσέγγιση η οποία όμως αναζητεί πρώτα στενά συνδεδεμένους κόμβους. Ένας άλλος αλγόριθμος προτάθηκε από τους J. Leskovec et al [18] οι οποίοι μετρούν την ποιότητα της καλύτερης δυνατής κοινότητας με βάση τη διάχυση πληροφορίας εντός της, χρησιμοποιώντας τη μετρική conductance. Στην έρευνα τους παρατήρησαν ότι σχεδόν όλα τα δίκτυα που μελετήθηκαν, παρουσίασαν κοινότητες σε πολύ μικρά μεγέθη (100 κόμβων περίπου) και ότι όσο αυξανόταν το μέγεθος τόσο χειροτέρευε η τιμή conductance.

Παράλληλα, τα τελευταία χρόνια βιώνουμε μια πρωτοφανή έκρηξη στην ποσότητα των διαθέσιμων δεδομένων και σε ότι αφορά το μέγεθος και την συνεπαγομένη πολυπλοκότητα των κοινωνικών δικτύων. Έχει καταλήξει να είναι κάτι σχετικά συνηθισμένο το να ερευνά κανείς κοινωνικά δίκτυα που αποτελούνται από εκατομμύρια κόμβους και αντίστοιχο πλήθος σχέσεων. Ως αποτέλεσμα, η ποσότητα των διαθέσιμων προς ανάλυση δεδομένων έχει ξεπεράσει τις δυνατότητες της συμβατικής ισχύος και μνήμης που είναι διαθέσιμη όχι μόνο στο εμπορικό hardware, αλλά και σε υπερυπολογιστές. Έτσι, οι επιστήμονες στρέφονται σε παράλληλους αλγόριθμους για τους υπολογισμούς πάνω σε τέτοιου μεγέθους δεδομένα. Το MapReduce και η ανοιχτού κώδικα εφαρμογή του, το Hadoop, σήμερα παρουσιάζονται ως η καθιερωμένη πλατφόρμα για τον υπολογισμό ενός εκ των βασικών μέτρων για τη μέτρηση της κοινωνικότητας, τον συντελεστή ομαδοποίησης (clustering coefficient) η οποία βασίζεται στην μέτρηση τριγωνικών σχέσεων ανάμεσα στα μέλη του δικτύου.

Ο συντελεστής ομαδοποίησης (clustering coefficient) μετρά το βαθμό στον οποίο οι γείτονες ενός κόμβου είναι οι ίδιοι γείτονες μεταξύ τους. Το πεδίο της κοινωνιολογίας μας δίνει δύο θεωρίες για την σημαντικότητα αυτού του μέτρου: όσο πιο δεμένη είναι η κοινότητα τόσο το πιθανότερο είναι για αυτή και τα μέλη της να αλληλεπιδρούν μεταξύ τους και να ξέρουν τη φήμη κάθε άλλου μέλους. Ο J. Coleman [15] και ο Al.

Portes [01], ισχυρίζονται ότι αν ένα άτομο έκανε κάτι εναντίον των κοινωνικών κανόνων, οι συνέπειες σε μια σφιχτά δεμένη κοινότητα θα ήταν μεγαλύτερες επειδή περισσότεροι άνθρωποι θα ήξεραν για την προσβλητική συμπεριφορά ή πράξη του και περισσότεροι άνθρωποι θα ήταν ικανοί να επιβάλλουν κυρώσεις στο άτομο που υπέπεσε στην προσβλητική αυτή συμπεριφορά ή πράξη.

Αυτό το φαινόμενο βοηθά στην ενθάρρυνση ενός υψηλότερου βαθμού εμπιστοσύνης στην κοινότητα και επίσης βοηθά στην δημιουργία θετικών κοινωνικών προτύπων. Έτσι, όταν ο συντελεστής ομαδοποίησης ενός ατόμου δείχνει ότι ανήκει σε μια σφιχτά «δεμένη» κοινωνία τότε το άτομο έχει το πλεονέκτημα υψηλότερων επιπέδων εμπιστοσύνης ανάμεσα στα υπόλοιπα μέλη της κοινότητάς του και η ίδια η κοινότητα περισσότερες θετικές κοινωνικές νόρμες μέσα σε αυτήν.

Στη δεύτερη θεωρητική προσέγγιση, ο R. Burt [27] ισχυρίζεται ότι τα άτομα που λειτουργούν σαν γέφυρες ανάμεσα σε κοινότητες, μπορεί να ωφεληθούν. Η θεωρία των δομικών κενών (τρυπών, holes) ισχυρίζεται ότι το να δημιουργήσεις μια σχέση ανάμεσα σε δύο άτομα που διαφορετικά δεν θα μπορούσαν να την έχουν, επιτρέπει στον ενδιάμεσο να αποκτήσει βαρύνουσα σημασία. Ο ενδιάμεσος μπορεί να πάρει ιδέες και από τους δύο ανθρώπους και να καταλήξει με κάτι όλως διόλου καινούριο. Το άτομο στη μέση θα μπορούσε επίσης να πάρει ιδέες από τον ένα ή από τις επαφές αυτού και να τις χρησιμοποιήσει για να λύσει προβλήματα που αντιμετωπίζει ο άλλος. Ο R. Burt επίσης ισχυρίζεται ότι οι επιπτώσεις αυτού του τύπου της μεσιτείας δεν επεκτείνονται σε καμία κατεύθυνση του δικτύου. Έτσι, αν ο συντελεστής ομαδοποίησης ενός κόμβου μετρήσει ότι το τμήμα των κόμβων-γειτόνων είναι επίσης γείτονες, συλλαμβάνει το ποσοστό κατά το οποίο ο κόμβος αυτός λειτουργεί ή όχι σαν ενδιάμεσος – μεσίτης ανάμεσα στους φίλους του. Συνεπώς, αν ο συντελεστής ομαδοποίησης ενός ατόμου δείξει ότι πολλοί από τους γείτονές του δεν είναι γείτονες μεταξύ τους, αυτό σημαίνει ότι το πρόσωπο αυτό θα μπορούσε να απολαμβάνει προνόμια επιπλέον πληροφοριών που οφείλονται σε αυτού του τύπου την «μεσιτεία» του μέσα στο κοινωνικό δίκτυο.

Το πρόβλημα του υπολογισμού του συντελεστή ομαδοποίησης ή του σχεδόν ομοίου του, του υπολογισμού των τριγώνων σε ένα γράφο έχει πλούσια ιστορία. Ο κλασσικός αλγόριθμος επαναλαμβάνεται σε όλους τους κόμβους και ελέγχει αν υπάρχουν δύο γείτονες σε έναν συγκεκριμένο κόμβο v οι οποίοι να είναι γείτονες μεταξύ τους. Παρόλο που είναι απλός, ο αλγόριθμος αυτός δεν εκτελείται καλά σε γράφους με

κατανομές μεγάλης κλίσης καθώς ένας απλός κόμβος με πολλές σχέσεις (υψηλού βαθμού, degree) μπορεί να οδηγήσει σε πολύ μεγάλο χρόνο τρεξίματος (running time) $O(n^2)$.

Οι N. Chiba et T. Nishizeki [23] μελέτησαν αυτό το πρόβλημα και πρότειναν ένα βελτιστοποιημένο σειριακό αλγόριθμο. Η διαδικασία τους, K3 τρέχει σε $O(n)$ χρόνο σε επίπεδους γράφους και πιο γενικά σε $O(m\alpha(G))$ χρόνο όπου α(.) ορίζει την arboricity του γράφου. Στη διδακτορική του διατριβή, ο Th. Schank [30] περιγράφει περεταίρω και αναλύει έναν αριθμό από διαφορετικούς αλγόριθμους που αντιμετωπίζουν το συγκεκριμένο πρόβλημα του χρόνου στα μεγάλα κοινωνικά δίκτυα.

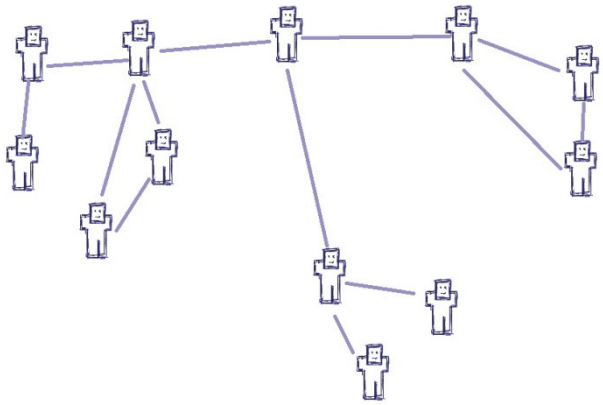
Για να λύσει κανείς το πρόβλημα ότι ένας γράφος μπορεί να είναι υπερβολικά μεγάλος για να χωράει στη μνήμη οι D. Coppersmith et R. Kumar [09] και οι L. Buriol et al. [20] πρότειναν αλγόριθμους ροής που υπολογίζουν τον συνολικό αριθμό των τριγώνων με υψηλή ακρίβεια ενώ χρησιμοποιούν περιορισμένο χώρο. Οι L. Becchetti et al. [19] προσπαθούν να αντιμετωπίσουν το δυσκολότερο πρόβλημα του υπολογισμού του αριθμού των τριγώνων σε κάθε κόμβο. Ο αλγόριθμός τους κάνει $O(\log n)$ διαδοχικές σαρώσεις στα δεδομένα, χρησιμοποιώντας $O(n)$ μνήμη για να δώσει ένα προσεγγιστικό αριθμό τριγώνων για κάθε κόμβο.

Οι εργασίες στων οποίων τα συμπεράσματα αλλά και κατευθύνσεις βασίζεται η παρούσα μεταπτυχιακή διατριβή είναι των Ch. Tsourakakis et. al [06] και S. Suri et S. Vassilvitskii, [28]. Στην πρώτη εργασία, οι συγγραφείς δίνουν μια τυχαιοποιημένη διαδικασία MapReduce για να μετρήσουν το συνολικό αριθμό των τριγώνων σε ένα γράφο και αποδεικνύουν ότι ο αλγόριθμος δίνει τον ακριβή αριθμό των αναμενόμενων τριγώνων. Επίσης περιορίζουν την διακύμανση και εμπειρικά παρουσιάζουν τις επιταχύνσεις που επιτυγχάνει ο αλγόριθμος τους πάνω στην κλασσική προσέγγιση. Οι αλγόριθμοι που προτείνονται και συγκρίνονται ως προς την απόδοση τους στη δεύτερη εργασία επιτυγχάνουν συγκρίσιμες επιταχύνσεις, δίνοντας τον συνολικό αριθμό τριγώνων που σχηματίζονται σε κάθε κόμβο, ενώ στην πρώτη εργασία δίνεται ο αριθμός των τριγώνων για το σύνολο του γράφου. Το να ξέρεις τον αριθμό τριγώνων σε κάθε κόμβο είναι σημαντικό για τον υπολογισμό του συντελεστή ομαδοποίησης κάθε κόμβου καθώς επίσης και η συγκρισιμότητα στις μετρήσεις των αποδόσεων των αλγορίθμων δίνουν μια δυνατή βάση για την ανάλυση σε αυτήν την μεταπτυχιακή διατριβή.

2.2 Κοινωνικά Δίκτυα – Βασικές ιδιότητες

Την τελευταία πενταετία έχει εξαπλωθεί η χρήση των κοινωνικών δικτύων και μέσα από αυτά αναδύεται ένας μεγάλος όγκος δεδομένων ο οποίος αρκετές φορές ακολουθεί συγκεκριμένους κανόνες και μονοπάτια. Επίσης μαζική πληροφορία εμπεριέχεται στην ανάλυση του όγκου αυτών των δεδομένων μεγάλης κλίμακας που μπορεί κανείς να εντοπίσει στα κοινωνικά δίκτυα. Ως παράδειγμα γνωστού κοινωνικού δικτύου είναι το Facebook παρά το γεγονός ότι δεν είναι το μόνο (βλ. Twitter, Google+, κα).

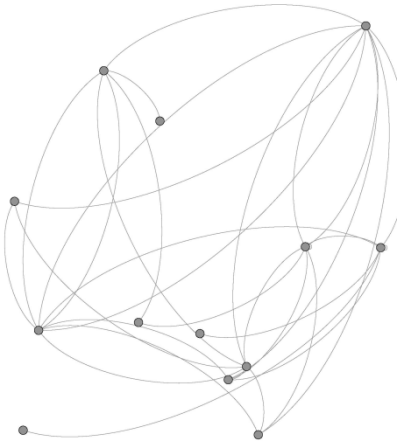
Ένα βασικό ερώτημα για την εξαγωγή των πληροφοριών από τέτοιου τύπου δίκτυα είναι ο προσδιορισμός των «κοινοτήτων» οι οποίες αποτελούνται από υποσύνολα κόμβων (ατόμων ή άλλου τύπου οντοτήτων που διαμορφώνουν ένα δίκτυο) οι οποίοι έχουν ανάμεσά τους ισχυρούς δεσμούς. Σημαντικό είναι το γεγονός ότι αυτές οι κοινότητες μεταξύ τους δεν αποτελούν σύνολα αμοιβαίως αποκλειόμενα αλλά αντιθέτως συχνά αλληλεπικαλύπτονται, έχοντας κοινούς κόμβους ή/και συνδέσεις. Παραδείγματος χάριν, στο Facebook μπορεί κανείς να ανήκει σε μια κοινότητα με φίλους ή συμμαθητές αλλά οι κοινότητες αυτές σπάνια έχουν κοινά μέλη μεταξύ τους, ωστόσο ο ίδιος αποτελεί κοινό κόμβο ανάμεσά τους. Δίκτυα όπως το Facebook εξυπηρετούν για την κατανόηση και τη μελέτη στις σχέσεις των «φίλων». Ένα βασικό ερώτημα είναι ο προσδιορισμός των κοινοτήτων του οι οποίες αποτελούν υποσύνολα κόμβων με πολύ δυνατή σχέση.



Σχήμα 2.1: μια πολύ απλή αποτύπωση σχέσεων μεταξύ χρηστών – οντοτήτων

Στο παραπάνω παράδειγμα παρατηρούμε ομάδες ανθρώπων συνδεδεμένων μεταξύ τους με σχέσεις φιλίας οι οποίες αναπαριστώνται με γραμμές που ενώνουν τους χρήστες. Ένα βασικό συμπέρασμα που έχει προκύψει και εδραιωθεί μαθηματικά είναι ότι τα κοινωνικά δίκτυα

μπορούν να αναπαρασταθούν γραφικά με ενός τύπου γράφημα το οποίο μπορεί να οριστεί ως γράφος.



Σχήμα 2.2: Παράδειγμα γραφική αποτύπωση των κοινωνικών με την μορφή γράφου

Πιο αναλυτικά, τα χαρακτηριστικά ενός κοινωνικού δικτύου είναι τα ακόλουθα:

1 αποτελεί συλλογή οντοτήτων (συνήθως ανθρώπων) που συμμετέχουν στο δίκτυο. Αυτές οι οντότητες είναι άτομα αλλά θα μπορούσε να είναι και κάτι διαφορετικό όπως τηλεφωνικά δίκτυα.

2 υπάρχει τουλάχιστον μια σχέση μεταξύ τους («φίλοι», follower, κ.λπ.) όπου σε κάποια δίκτυα η σχέση αυτή είναι διακριτή (φίλοι, οικογένεια, ομάδα, κλίκα, κ.λπ.). Μερικές φορές οι σχέσεις είναι αμοιβαίως αποκλειόμενες: ή είσαι «φίλος» είτε όχι. Όμως σε άλλα παραδείγματα κοινωνικών δικτύων μπορεί οι σχέσεις να είναι διαβαθμισμένες όπως είναι στους κύκλους της Google+.

3 υπάρχει μια παραδοχή για την ανίχνευση μη τυχαιότητας (non-randomness) ή «τοπικότητας». Αυτή η συνθήκη είναι η δυσκολότερη για να αποδοθεί μαθηματικά, αλλά αντιλαμβανόμαστε ότι αυτή η σχέση μπορεί να οριστεί ως εξής και τείνει προς cluster: Αν η οντότητα Α σχετίζεται με την οντότητα Β και την οντότητα Γ, υπάρχει κατά μέσο όρο υψηλότερη πιθανότητα ο Β και ο Γ να σχετίζονται. μεταξύ τους.

Οι σχέσεις στα κοινωνικά δίκτυα μπορούν να είναι διαφόρων ειδών πέρα από την έννοια των «φίλων» στο Facebook. Θα μπορούσαν να είναι:

1 Τηλεφωνικά Δίκτυα: Εδώ οι κόμβοι αντιπροσωπεύουν μοναδικούς αριθμούς τηλεφώνων και η σχέση μεταξύ τους ορίζεται αν ένα τηλεφώνημα συνοδεύει σε συγκεκριμένο χρόνο, οι σχέσεις αυτές θα μπορούσαν να εμπλουτισθούν με διαβάθμιση ανάλογα τον αριθμό των τηλεφωνημάτων που έχουν γίνει στο χρονικό αυτό διάστημα.

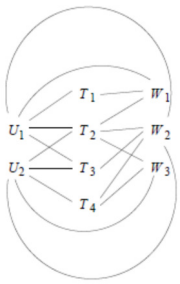
2 Δίκτυα ηλεκτρονικού Ταχυδρομείου: Οι κόμβοι αντιπροσωπεύουν διευθύνσεις ηλεκτρονικού ταχυδρομείου που είναι επίσης μοναδικές. Μια σχέση αντιπροσωπεύει το γεγονός ότι υπάρχει τουλάχιστον ένα μήνυμα ηλεκτρονικού ταχυδρομείου ανάμεσα στις δύο διευθύνσεις. Εναλλακτικά μπορούμε να τοποθετήσουμε μια σχέση αν υπάρχουν μηνύματα και στις δύο κατευθύνσεις. Με τον τρόπο αυτόν αποφεύγουμε τους spamers και τα θύματά τους να τους προσμετρήσουμε στις σχέσεις του δικτύου.

3 Δίκτυα Συνεργασίας: Αφορά ακαδημαϊκά δίκτυα που έχουν εκδώσει επιστημονικά άρθρα. Υπάρχει μια σχέση ανάμεσα στα άτομα που έχουν εκδώσει ένα ή παραπάνω άρθρα μαζί. Μπορούμε επίσης προαιρετικά να διαβαθμίσουμε τις σχέσεις ανάλογα με τον αριθμό των συσχετιζόμενων εκδόσεων. Ένας εναλλακτικός τρόπος για να δει κανείς αυτό το δίκτυο είναι να θεωρήσει ως κόμβους τα άρθρα και ως σχέση θα νοείται ότι έχουν έναν κοινό συγγραφέα.

4 Διμερή (bipartite) ή πολυμερή (k-partite) Δίκτυα διαφόρων κόμβων: υπάρχουν επιπλέον κοινωνικά φαινόμενα που συνδέονται μεταξύ τους με διαφόρων τύπων κόμβων, όπως για παράδειγμα στο www.Delicious.com , όπου οι χρήστες τοποθετούν ετικέτες (tags) σε διάφορες ιστοσελίδες. Μπορούμε να θεωρήσουμε ότι οι χρήστες συνδέονται αν τείνουν να χρησιμοποιούν τις ίδιες ετικέτες συχνά ή τις ίδιες ιστοσελίδες. Ομοίως οι ετικέτες θα μπορούσαν να συνδέονται αν εμφανίζονταν στις ίδιες σελίδες ή χρησιμοποιούνταν από τους ιδίους χρήστες, ενώ οι ιστοσελίδες θα μπορούσαν να συνδέονταν αν είχαν πολλές όμοιες ετικέτες ή αν οι ίδιοι χρήστες έβαζαν ετικέτες σε αυτές.

Ο τρόπος για να αναπαραστήσει κανείς αυτού του τύπου τα δίκτυα είναι ένας k-partite (πολυμερής) γράφος για k>1. γενικά, ένας πολυμερής γράφος (k-partite graph) αποτελείται από k μη συνδεδεμένα σύνολα κόμβων , χωρίς σχέσεις ανάμεσα στους κόμβους του ίδιου συνόλου. Στη συνέχεια παρουσιάζεται ένας τριμερής γράφος (k=3) με βάση το παράδειγμα του www.delicious.com ο οποίος αποτελείται από σύνολα χρηστών (U), ετικετών (T) και

ιστοσελίδων (W). Σημειώνεται ότι όλα τα άκρα συνδέουν κόμβους σε διαφορετικά σύνολα. Μπορούμε να υποθέσουμε ότι αυτός ο γράφος αναπαριστά πληροφορία για τρία είδη οντοτήτων, για παράδειγμα η σχέση (U1, T2) σημαίνει ότι ο χρήστης U1 έχει τοποθετήσει την ετικέτα T2 σε τουλάχιστον μια ιστοσελίδα. Σημειώνεται ότι αυτός ο γράφος δεν μας δίνει μια πληροφορία που θα μπορούσε να είναι σημαντική: ποιος τοποθέτησε ποια ετικέτα σε ποια σελίδα: αυτό θα απαιτούσε πίνακα τριπλής εισόδου σε βάση δεδομένων .



Σχήμα 2.3: Παράδειγμα τριμερή γράφου χρηστών (U), ετικετών (T) και ιστοσελίδων (W)

2.2.1 Τύποι κοινωνικών δικτύων – Βασικά στατιστικά

Βασικοί τύποι κοινωνικών δικτύων (βλ. πιο αναλυτικά και κεφάλαιο 4) σύμφωνα με το http://snap.stanford.edu/data/index.html είναι οι εξής:

1 Κατευθυνόμενα δίκτυα (Directed)
2 Μη-κατευθυνόμενα δίκτυα (Undirected)
3 Διμερή δίκτυα (Bipartite)
4 Δίκτυα πολλαπλών συνδέσεων ανάμεσα σε δύο κόμβους (Multigraph)
5 Χρονικά δίκτυα (Temporal): για κάθε κόμβο ή σχέση ξέρουμε πότε αυτή εμφανίστηκε στο δίκτυο
6 Διαβαθμισμένα (Labeled): τα δίκτυα αυτά περιέχουν σταθμίσεις (ταμπέλες: βάρη ή ιδιότητες) στους κόμβους ή τις σχέσεις μεταξύ τους.

Τα βασικά στατιστικά που περιγράφουν τις χαρακτηριστικές ιδιότητες ενός κοινωνικού δικτύου είναι τα εξής:

Στατιστικά κοινωνικών δικτύων (**Dataset statistics)**	
Κόμβοι (nodes)	Number of nodes in the network
Σχέσεις (Edges)	Number of edges in the network
Κόμβοι στο μεγαλύτερο WCC	Number of nodes in the largest weakly connected component (για κατευθυνόμενα δίκτυα)
Συνδέσεις στο μεγαλύτερο WCC	Number of edges in the largest weakly connected component (για κατευθυνόμενα δίκτυα)
Κόμβοι στο μεγαλύτερο SCC	Number of nodes in the largest strongly connected component (για κατευθυνόμενα δίκτυα)
Συνδέσεις στο μεγαλύτερο SCC	Number of edges in the largest strongly connected component (για κατευθυνόμενα δίκτυα)
Μέσος συντελεστής ομαδοποίησης (Average clustering coefficient)	Average clustering coefficient
Αριθμός των τριγώνων	Αριθμός τριάδων συνδεόμενων κόμβων για τα μη κατευθυνόμενα δίκτυα
Ποσοστό των κλειστών τριγώνων	Αριθμός των συνδεόμενων τριγώνων/αριθμό των συνδεόμενων μη-κατευθυνόμενων μονοπατιών βαθμού 2 (δύο συνδεόμενων κόμβων)
Διάμετρος (longest shortest path)	Το μήκος του μέγιστου μη κατευθυνόμενου κοντινότερου μονοπατιού (σε δείγμα πάνω από 1.000 τυχαίους κόμβους).
90% αποτελεσματική διάμετρος-(90 percentile effective diameter)	90% του μήκους της κατανομής του μη κατευθυνόμενου κοντινότερου μονοπατιού (σε δείγμα πάνω από 1.000 τυχαίους κόμβους).

Πηγή: http://snap.stanford.edu/data/index.html

Πίνακας 2.4: Στατιστικά κοινωνικών δικτύων

2.3 Hadoop & MapReduce: Περιγραφή εργαλείων

Το MapReduce έχει γίνει ένα πλέον ένα de facto πρότυπο για τους παράλληλους υπολογισμούς σε κλίμακες terabyte και petabyte δεδομένων. Εδώ περιγράφουμε περιληπτικά τον τρόπο με τομ οποίο το MapReduce οργανώνει τους υπολογισμούς:

2.3.1 Σύνοψη διαδικασιών MapReduce

Τα εισερχόμενα (input) και τα ενδιάμεσα δεδομένα αποθηκεύονται με την μορφή ζευγαριών {key, value} και οι υπολογισμοί γίνονται σε κύκλους(4). Κάθε κύκλος είναι κομματιασμένος σε τρεις διαδοχικές φάσεις: map, shuffle και reduce. Στη φάση map τα εισερχόμενα δεδομένα input επεξεργάζονται κατά μία «πλειάδα» (tuple) τη φορά. Αυτό επιτρέπει σε διαφορετικές πλειάδες (tuples) να επεξεργάζονται από διαφορετικές μηχανές και δημιουργεί την δυνατότητα για μαζική παραλληλοποίηση. Κάθε μηχανή εκτελεί την map λειτουργία, παράγοντας ένα νέο σύνολο {key,value} ζευγαριών τα οποία περνούν στην φάση shuffle. Αυτή αποτελεί το βήμα του συγχρονισμού ανάμεσα στο κόμβους τους συστήματος.

Σε αυτήν την φάση η δομή του MapReduce συλλέγει ανάμεσα σε όλες τις πλειάδες (tuples) που έχουν παραχθεί από τους mappers, οργανώνει τα υποσύνολα με το ίδιο key μαζί και τα τροφοδοτεί στο χώρο του cluster. Τελικά κάθε key μαζί με όλες τις τιμές (values) που σχετίζονται με αυτόν επεξεργάζονται μαζί κατά την φάση reduce σε έναν και μοναδικό κόμβο οποίος ονομάζεται reducer. Σε αυτό το σημείο επίσης οι λειτουργίες στα δεδομένα με το ίδιο key είναι ανεξάρτητες από τα δεδομένα με διαφορετικό key και μπορούν να επεξεργαστούν παράλληλα από διαφορετικές μηχανές (reducers).

2.3.2 Σύντομο ιστορικό για το Hadoop

Το Apache™ Hadoop™ (3) είναι ένα προγραμματιστικό περιβάλλον που δημιουργήθηκε από τον Doug Cutting, δημιουργό του Apache Lucene. Το Hadoop βασίζεται στο Apache Nutch, μια μηχανή αναζήτησης «ανοικτού κώδικα» επίσης μέρος του Lucene project.

Το όνομα δεν είναι ακρώνυμο αλλά όπως περιγράφει ο δημιουργός του Doug Cutting ήταν δημιουργία του γιου του για να ονοματίσει έναν κίτρινο λούτρινο ελέφαντα. Σύντομο και εύκολο

στη διατύπωση και την προφορά του, χωρίς νόημα και χωρίς να έχει χρησιμοποιηθεί αλλού, το χρησιμοποίησε για να ονοματοδοτήσει το δημιούργημα του.

Η δημιουργία μιας μηχανής αναζήτησης εκ του μηδενός ήταν ένας φιλόδοξος στόχος όχι μόνο γιατί ο κώδικας για την ανίχνευση και ταξινόμηση πολύπλοκων ονομάτων ιστοσελίδων, αλλά επίσης ήταν μια πρόκληση να εκτελεστεί χωρίς μια ενωμένη και λειτουργική ομάδα καθώς δεν υπήρχε σταθερή ομάδα εργασίας. Επίσης ήταν ένα ακριβό εγχείρημα: ο Mike Cafarella και ο Doug Cutting εκτίμησε ότι ένα σύστημα που υποστηρίζει ένα ευρετήριο ενός δισεκατομμυρίου σελίδων θα κόστιζε περίπου μισό εκατομμύριο δολάρια σε εξοπλισμό (hardware) με ένα μηνιαίο λειτουργικό κόστος $30.000. Παρόλα αυτά πίστεψαν ότι ήταν ένας εύκολα επιτεύξιμος στόχος καθώς θα γινόταν ανοιχτό λογισμικό και τελικά θα έδινε στο ευρύ κοινό την δυνατότητα πρόσβασης στον κώδικα αλγορίθμων μηχανών αναζήτησης.

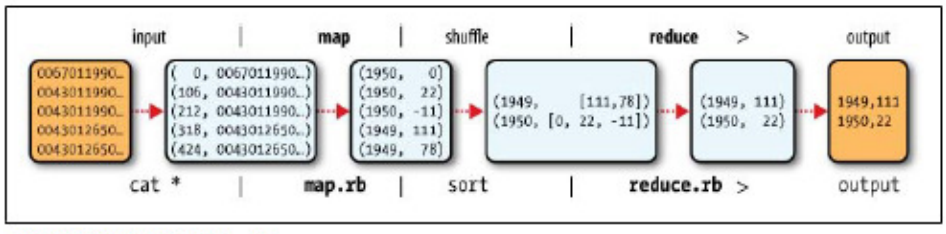
Ο Nutch ξεκίνησε το 2002 και γρήγορα ανέπτυξε το υπόβαθρο και ένα πρώτο σύστημα αναζήτησης ήταν έτοιμο. Παρόλα αυτά συνειδητοποίησαν ότι το αρχιτεκτόνημα δεν θα μπορούσε να λειτουργήσει στην κλίμακα των δισεκατομμυρίων των σελίδων στο διαδίκτυο. Η βοήθεια ήρθε με την δημοσίευση ενός άρθρου το 2003 το οποίο περιέγραφε την αρχιτεκτονική που χρησιμοποιούσε το κατανεμημένο σύστημα αρχείων (distributed file system) της Google (GFS). Το GFS ή κάτι παρόμοιο τέτοιο θα έλυνε τις ανάγκες αποθήκευσης για τους πολύ μεγάλους φακέλους που δημιουργούνταν ως τμήμα από την αναζήτηση στο διαδίκτυο καθώς και στη διαδικασία ευρετηρίασης (indexing process). Ιδιαίτερα το GFS θα ελευθέρωνε τον χρόνο που απαιτούνταν για το διαχειριστικό κόστος όπως η διαχείριση των κόμβων αποθήκευσης (managing storage nodes).

Την ίδια χρονιά η Google δημοσίευσε το άρθρο που εισήγαγε το Map Reduce στον κόσμο της πληροφορικής. Στις αρχές του 2005 οι προγραμματιστές του Nutch δημιούργησαν μια εφαρμογή σε Map Reduce στο Nutch, και μέχρι τα μέσα αυτού του χρόνου όλοι οι κύριοι αλγόριθμοι του Nutch είχαν μετατραπεί ώστε να λειτουργούν χρησιμοποιώντας Map Reduce και NDFS. Το NDFS και οι εφαρμογή του Map Reduce στο Nutch ήταν σε εφαρμογή πέρα από το ερευνητικό επίπεδο και τον Φεβρουάριο του 2006 μετακινήθηκαν έξω από το Nutch για να δημιουργήσουν ένα ανεξάρτητο υπο-έργο του Lucene ονομαζόμενο «Hadoop». Περίπου στον ίδιο χρόνο, ο Doug Cutting συνδέθηκε με τη Yahoo! και σύντομα ανακοινώθηκε ότι η ευρετηρίαση της μηχανής αναζήτησης της δημιουργήθηκε από έναν Hadoop cluster με 10000 cores. Στην συνέχεια το Hadoop εντάχθηκε στην Apache και μέχρι χρησιμοποιείται από αρκετές άλλες εταιρείες εκτός του Yahoo!, όπως Facebook., Twitter, Amazon, New York Times, κ.α.

Σε ένα αρκετά προβεβλημένο επίτευγμα, οι New York Times χρησιμοποίησαν ένα νέφος δεδομένων της Amazon EC2 για να «ροκανίσει» τέσσερα terabytes σκαναρισμένων αρχείων από χαρτί και να τα μετατρέψει σε PDF για το διαδίκτυο. Η διαδικασία διήρκησε λιγότερο από 24 ώρες για να εκτελεστεί χρησιμοποιώντας 100 μηχανές και το έργο πιθανώς δεν θα είχε καν ξεκινήσει χωρίς τον συνδυασμό της του μοντέλου της Amazon και το εύχρηστο μοντέλο παράλληλου προγραμματισμού του Hadoop. Τον Απρίλιο του 2008 το Hadoop έσπασε ένα παγκόσμιο ρεκόρ όντας το γρηγορότερο σύστημα για να ταξινομήσει ένα terabyte δεδομένων. Τρέχοντας σε 910-node cluster, Hadoop ταξινόμησε ένα terabyte σε 209 δευτερόλεπτα (λιγότερο από 3½ λεπτά), ξεπερνώντας τον νικητή της προηγούμενης χρονιάς των 297 δευτερολέπτων (βλ. "Terabyte Sort on Apache Hadoop" σ.553 [12]). Το Νοέμβριο του ίδιου έτους η Google ανέφερε ότι η εφαρμογή της για Map Reduce ταξινόμησε ένα terabyte σε 68 seconds. Το Μάιο του 2009 ανακοινώθηκε ότι η ομάδα του Yahoo! χρησιμοποίησε το Hadoop για να ταξινομήσει ένα terabyte σε 62 δευτερόλεπτα.

2.3.3 Ανάλυση δεδομένων - Map Reduce

Για να εκμεταλλευτεί κανείς τις παράλληλες διαδικασίες που παρέχει το Hadoop, χρειάζεται να εκφράσει το ερώτημα (query) του σαν μια εργασία του Map Reduce. Αφού δοκιμασθεί τοπικά σε μικρής κλίμακας τεστ, τότε θα είναι ικανό να το τρέξει σε ένα σύστημα cluster μηχανών. Ο Map Reduce λειτουργεί με το να διαχωρίσει τη διαδικασία σε δύο φάσεις: την φάση map την φάση reduce. Κάθε φάση έχει ζευγάρια τιμών – κλειδιών σαν εισροές και εκροές τους τύπους των οποίων μπορεί να επιλέξει ο προγραμματιστής. Ο προγραμματιστής επίσης προσδιορίζει δύο λειτουργίες: τη λειτουργία map και τη λειτουργία reduce. Επιλέγουμε ένα κείμενο εισροής (text format) που θα μας δίνει κάθε γραμμή στο σώμα των δεδομένων (dataset) σαν text value.



MapReduce logical data flow

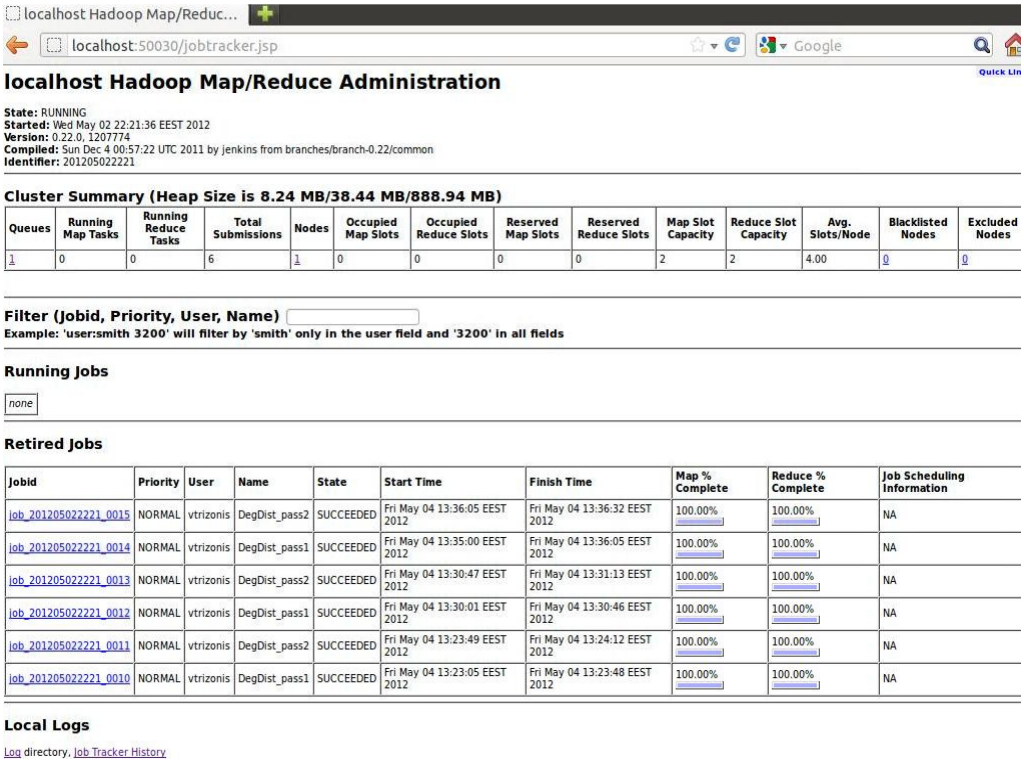
Σχήμα 2.5: Λογική ροή των δεδομένων στο MapReduce

2.3.4 Ροή δεδομένων (Data flow)

Αρχικά για να γίνουν κατανοητές οι διαδικασίες είναι απαραίτητη η παράθεση της σχετικής ορολογίας:

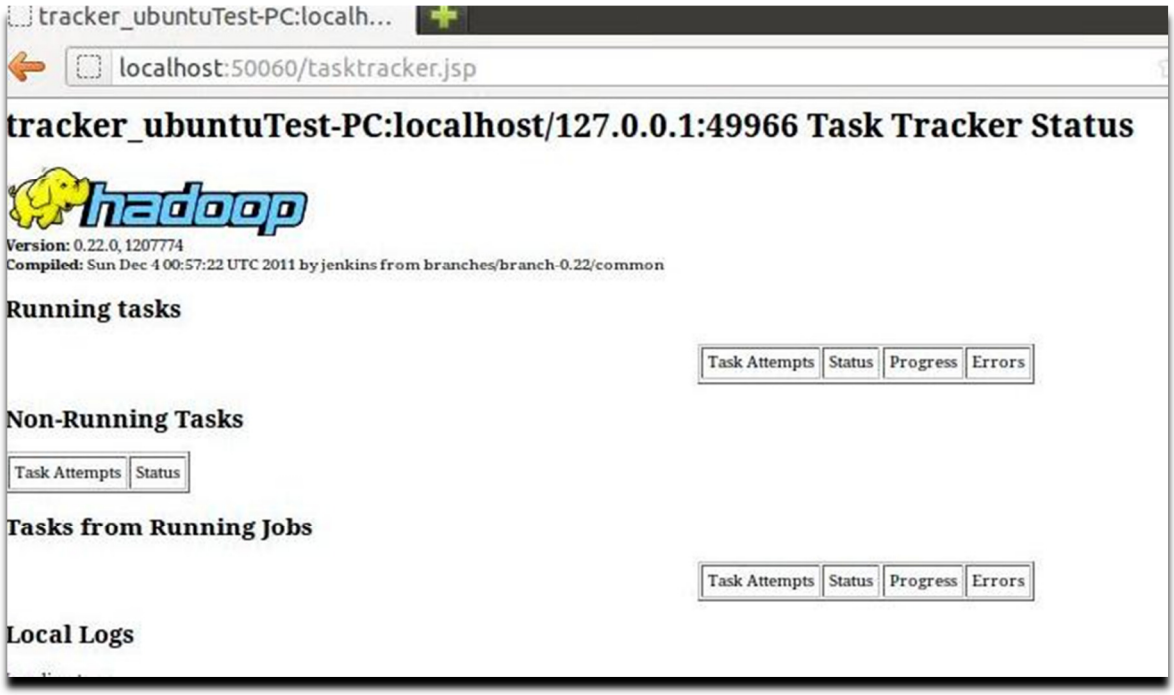
Μία διεργασία εντός του περιβάλλοντος Map Reduce είναι μια διεργασία στην οποία ο πελάτης (client) αναζητά την επεξεργασία κάποιων δεδομένων. Αποτελείται από τα δεδομένα εισροής (input data) και τις πληροφορίες διαμόρφωσης του περιβάλλοντος Hadoop ώστε να υποδεχθεί αυτά τα δεδομένα. Το Hadoop εκτελεί την εργασία διαχωρίζοντάς την βασικά σε δύο φάσεις: τη διεργασία map και τη διεργασία reduce.

Υπάρχουν επίσης δύο τύποι κόμβων που ελέγχουν την διαδικασία εκτέλεσης: ο jobtracker και ο tasktracker. Ο jobtracker συντονίζει όλες τις εργασίες του συστήματος σχεδιάζοντας τα διεργασίες για να εκτελεστούν στους tasktrackers.



Εικόνα 2.6: Η ιστοσελίδα με την ανάλυση του jobtracker

Οι tasktrackers εκτελούν διεργασίες και στέλνουν αναφορές προόδου (task reports) στον jobtracker, ο οποίος διατηρεί μια εγγραφή για κάθε διεργασία σε κάθε εργασία (job). Αν μία διεργασία αποτύχει, ο jobtracker μπορεί να την αναθέσει σε ένα διαφορετικό tasktracker.



Εικόνα 2.7: Η ιστοσελίδα με την ανάλυση του tasktracker

Για την καλύτερη κατανόηση θα χρησιμοποιήσουμε ένα παράδειγμα με τα δεδομένα (data set) του κοινωνικού δικτύου Twitter, περιγραφή των οποίων θα δοθεί σε επόμενη παράγραφο. Τα δεδομένα είναι αποθηκευμένα ως ASCII χαρακτήρες σε γραμμές και σε κάθε γραμμή υπάρχει η σχέση φιλίας (φίλος 1 [tab] φίλος2).

Η λογική του Map Reduce είναι να διαιρέσει τη διαδικασία σε δύο φάσεις. Τη φάση map και τη φάση Reduce. Και οι δύο φάσεις έχουν ως ορίσματα τις τιμές key – value, ως είσοδο και έξοδο.

Η είσοδος για τη φάση map είναι τα δεδομένα από το data set. Επιλέγουμε το format να είναι text, ώστε κάθε γραμμή της εισόδου να έρχεται ως text value. Η συγκεκριμένη διαδικασία (map) αποτελεί απλά μια φάση προετοιμασίας των δεδομένων και τροποποίησής τους έτσι, ώστε ο Reducer να μπορεί να κάνει σε αυτά τους υπολογισμούς του, που είναι η απεικόνιση των σχέσεων φιλίας. Κατά την φάση του map γίνεται και ένα έλεγχος – φιλτράρισμα των δεδομένων μας για πιθανά λάθη. Τέτοια μπορεί να είναι είτε διπλοεγγραφές ή λάθος είσοδος σε σχέση με τον τύπο δεδομένων που έχουμε προεπιλέξει.

Κατά την είσοδο στη Map διαδικασία του αρχείου δεδομένων, θα μπαίνουν ζεύγη που θα έχουν κλειδί την αύξουσα γραμμή και value την τιμή 1 που υποδηλώνει την παρουσία του κόμβου

Πρακτικά δηλαδή πραγματοποιείται μία απλή καταμέτρηση των σχέσεων αμοιβαίας φιλίας. Θέλουμε στην έξοδο της Map φάσης να λάβουμε ζεύγη της μορφής : χρήστης Α [tab] 1, όπου 1 είναι απλά η παρουσία μίας σχέση φιλίας που εκκινεί από τον χρήστη Α

Τα δεδομένα στην είσοδο του map θα έχουν την παρακάτω μορφή:

12 13
12 14
12 25
13 17
13 28
σε μορφή text και είναι διαχωρισμένα με tab (χαρακτήρας παραγράφου).

Μετά την εκτέλεση της map task, το αποτέλεσμα θα είναι:

key out, value out

12, 1
12, 1
12, 1
13, 1
13, 1
(όπου 1 είναι η παρουσία του κάθε κόμβου στην σχέση φιλίας).
Στη συνέχεια το Map Reduce περιβάλλον θα κάνει shuffle όλα τα παραπάνω ζεύγη ώστε να οργανωθούν και θα τα κατευθύνει προς τους reducers.

Συνεπώς, στην είσοδο της Reduce διαδικασίας θα βρίσκονται τα ζεύγη με ίδιο key. Στο συγκεκριμένο παράδειγμα θα δημιουργηθούν 2 reducers για key 12 και 13. Κάθε ένας θα προσμετρήσει τα 1 που έχει λάβει. Έτσι ο reducer 12 θα έχει συνολικό βαθμό 3 και ο 13 βαθμό 2. Η έξοδος θα είναι και αυτή της μορφής ζεύγους (key, value):

(12,3)

(13, 2)

Για να εξετασθεί η λειτουργία του Hadoop σε μεγάλο όγκο δεδομένων και σε διαφορετικούς υπολογιστές (nodes) πρέπει πρώτα να αναλύσουμε τον τρόπο λειτουργίας του στο Map Reduce:

Το Hadoop διαχωρίζει τις εισροές (inputs) μιας εργασίας (MapReduce job) σε συγκεκριμένου μεγέθους κομμάτια τα οποία τα ονομάζει τμήματα εισροής (input splits) ή απλά splits. Το Hadoop δημιουργεί ένα «χάρτη καθηκόντων» για κάθε ένα τμήμα εισροής, το οποίο τρέχει την από το χρήστη προσδιορισμένη λειτουργία map για κάθε εγγραφή στο τμήμα (split). Όταν υπάρχουν πολλά τμήματα σημαίνει ότι ο χρόνος που χρειάζεται για να ολοκληρωθεί η διαδικασία στο τμήμα είναι μικρός αν τον συγκρίνεις με το χρόνο που χρειάζεται για να ολοκληρωθεί η εργασία σε όλο το κομμάτι της εισροής (input). Έτσι, αν εμείς εκτελούμε τα τμήματα παράλληλα, η διαδικασία διαμοιράζεται καλύτερα και πιο ισορροπημένα αν τα τμήματα είναι μικρά, εφόσον μια πιο γρήγορη μηχανή θα είναι ικανή να εκτελέσει τμηματικά περισσότερα τμήματα από ότι μια πιο αργή μηχανή.

Το Hadoop καταφέρνει να τρέξει τον map task με βέλτιστο τρόπο σε έναν κόμβο όπου τα δεδομένα εισόδου (input data) επανα-προσδιορίζονται σε μέγεθος HDFS. Αυτό ονομάζεται τοπική βελτιστοποίηση δεδομένων (data locality optimization). Το βέλτιστο μέγεθος των τμημάτων είναι το ίδιο με το μέγεθος των block: είναι το μέγιστο μέγεθος εισροής (input) που μπορεί εγγυημένα να αποθηκευτεί σε έναν απλό κόμβο (node).

Αν το τμήμα επεκτεινόταν σε δύο blocks, θα ήταν απίθανο ότι κάθε HDFS κόμβος θα αποθηκευόταν και στα δύο blocks, έτσι κάποιο από το τμήμα θα έπρεπε να μεταφερθεί μέσω του δικτύου στον κόμβο που τρέχει εκείνη τη στιγμή στο map task, κάτι το οποίο είναι σαφώς λιγότερο αποτελεσματικό από το να τρέχει το map task χρησιμοποιώντας τα τοπικά δεδομένα. Τα Map tasks εγγράφουν το αποτέλεσμα τους στον τοπικό δίσκο και όχι στο HDFS. Για πιο λόγο γίνεται αυτό; Τα αποτελέσματα του Map αποτελούν ένα άμεσο αποτέλεσμα: επεξεργάζονται από τα reduce tasks για να παραχθεί το τελικό προϊόν και εφόσον η εργασία έχει ολοκληρωθεί τα εξαγόμενα του map μπορούν να πεταχτούν. Έτσι, αν τα αποθηκεύσουμε στα HDFS, με την αναπαραγωγή θα γινόταν υπέρβαση (overkill). Αν ο τρέχων κόμβος στο map task αποτύχει πριν από την ανάλωση του αποτελέσματός του από το reduce task, τότε το Hadoop αυτόματα θα ξανατρέξει το map task σε έναν άλλο κόμβο για να ξαναδημιουργήσει το αποτέλεσμα του map (output).

Τα reduce tasks δεν έχουν το πλεονέκτημα της τοπικότητας των δεδομένων. Η εισροή για ένα απλό reduce task είναι κανονικά αποτέλεσμα από όλους τους mappers. Στο παράδειγμα που δόθηκε έχουμε έναν απλό reduce task το οποίο τροφοδοτείται από όλα τα map tasks. Συνεπώς, τα ταξινομημένα αποτελέσματα των map πρέπει να μεταφερθούν από το δίκτυο στον κόμβο όπου το reduce task τρέχει και στη συνέχεια περνάει στην από το χρήστη προσδιορισμένη

συνάρτηση μείωσης (reduce function). Το αποτέλεσμα του reduce συνήθως αποθηκεύεται στο HDFS για λόγους αξιοπιστίας. Για κάθε HDFS block του αποτελέσματος του reduce, το πρώτο αντίγραφο αποθηκεύεται στον τοπικό κόμβο με τα υπόλοιπα αντίγραφα να αποθηκεύονται σε off-rack κόμβους. Συνεπώς, γράφοντας το αποτέλεσμα των reduce καταναλώνεται bandwidth του δικτύου. Η συνολική ροή των δεδομένων για ένα απλό reduce task εμφανίζεται στο Σχήμα 2.8. Τα παραλληλόγραμμα με τις διακεκομμένες γραμμές σημειώνουν τους κόμβους, τα λεπτά τόξα σημειώνουν τις μεταφορές των δεδομένων σε έναν κόμβο και τα μεγαλύτερα τόξα δείχνουν τις μεταφορές ανάμεσα στους κόμβους.

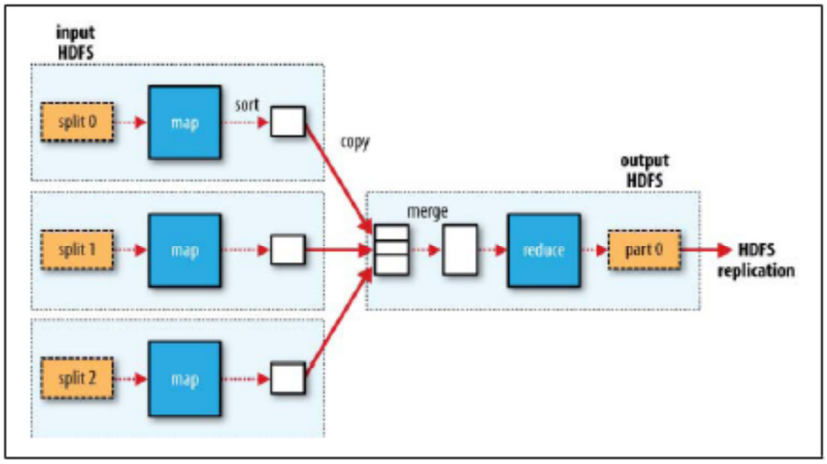


Figure 2-2. MapReduce data flow with a single reduce task

Σχήμα 2.8: Λογική ροή των δεδομένων στο MapReduce για ένα απλό καθήκον μείωσης (reduce task)

Ο αριθμός των reduce tasks δεν καθορίζεται από το μέγεθος της εισροής (input) αλλά καθορίζεται ανεξάρτητα. Όταν υπάρχουν πολλαπλοί reducers, οι διεργασίες map διαχωρίζουν σε partition το αποτέλεσμα τους (output), κάθε ένα δημιουργεί ένα particion για κάθε reduce task. Μπορούν να υπάρξουν πολλά κλειδιά (και οι σχετικές τους τιμές) σε κάθε διαμέρισμα, αλλά οι εγγραφές για κάθε δεδομένο κλειδί βρίσκονται όλες στο ίδιο διαμέρισμα. Η δημιουργία διαμερισμάτων (partitioning) μπορεί να ελέγχεται από μία συνάρτηση διαχωρισμού διαμορφωμένη από τον χρήστη (user-defined partitioning function) λειτουργία διαχωρισμού διαμερισμάτων, αλλά συνήθως ο εξ' ορισμού διαχωριστής (default partitioner) — ο οποίος δημιουργεί λίστες με keys (buckets) χρησιμοποιώντας μία πολύπλοκη συνάρτηση — λειτουργεί πολύ καλά.

Η ροή των δεδομένων για την γενική περίπτωση των πολλαπλών καθηκόντων μείωσης (reduce tasks) παρουσιάζεται στο Σχήμα 2.9: Σε αυτό το διάγραμμα διασαφηνίζεσαι γιατί η ροή των

δεδομένων ανάμεσα στο map και στις διεργασίες reduce (reduce tasks) ονομάζεται ως «the shuffle» (μτφ: το ανακάτεμα) καθώς κάθε reduce task τροφοδοτείται από πολλά map.

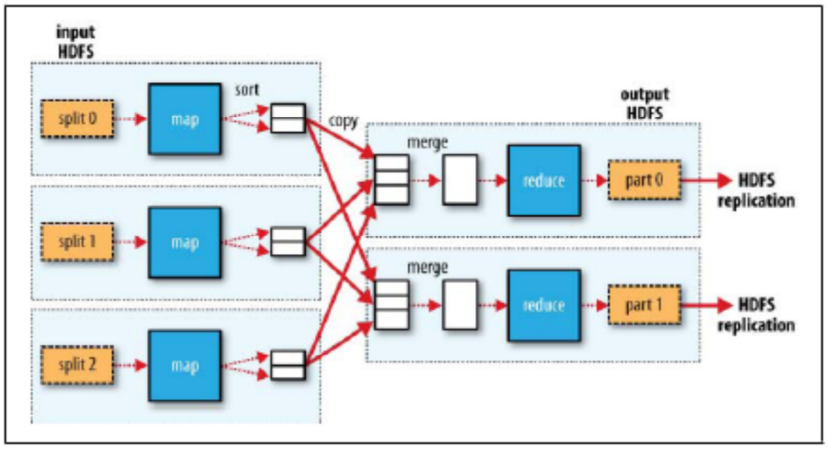


Figure 2-3. MapReduce data flow with multiple reduce tasks

Σχήμα 2.9: Λογική ροή των δεδομένων στο MapReduce με πολλαπλά καθήκοντα μείωσης (reduce tasks)

Υπάρχει και η πιθανότητα να μην έχουμε reduce tasks. Σε αυτήν την περίπτωση η μόνη μεταφορά δεδομένων εκτός κόμβου γίνεται όταν τα task map γράφουν στον HDFS, βλ. Σχήμα 2.10:.

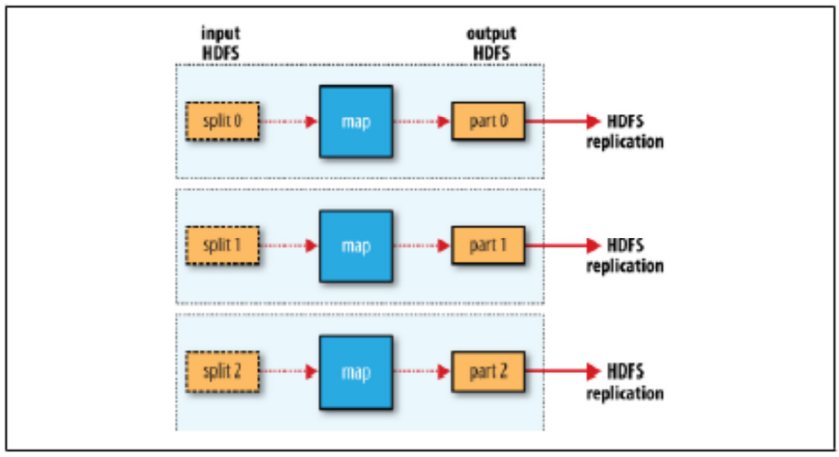


Figure 2-4. MapReduce data flow with no reduce tasks

Σχήμα 2.10: Λογική ροή των δεδομένων στο MapReduce χωρίς reduce tasks

2.3.5 Hadoop Distributed Filesystem (HDFS)

Ο σχεδιασμός του (HDFS)

Το HDFS είναι ένα σύστημα διαχείρισης αρχείων (filesystem) σχεδιασμένο για να αποθηκεύει μεγάλο αριθμό φακέλων με μοτίβα τύπου streaming data access, εκτελούμενο πάνω σε clusters πολλών κατασκευαστών υπολογιστικού εξοπλισμού (hardware).

Ας εξετάσουμε τα παραπάνω με μεγαλύτερη λεπτομέρεια:

Πολύ μεγάλα αρχεία (Very large files): «Πολύ μεγάλα» στο συγκεκριμένο πλαίσιο σημαίνει αρχεία μεγέθους εκατοντάδων megabytes, gigabytes, ή terabytes. Σήμερα, υπάρχουν Hadoop clusters που εκτελούνται και αποθηκεύουν petabytes δεδομένων.

Πρόσβαση συνεχούς ροής σε δεδομένα (Streaming data access): Το HDFS βασίζεται γύρω από την ιδέα ότι το πιο αποτελεσματικό μοτίβο επεξεργασίας δεδομένων είναι ένα μοτίβο μίας εγγραφής (write - once), πολλών αναγνώσεων (read-many-times). Ένα dataset συνήθως δημιουργείται ή αντιγράφεται από την πηγή και τότε πολλές και διάφορες αναλύσεις εκτελούνται σε αυτό το dataset στη διάρκεια του χρόνου. Κάθε ανάλυση θα περιλάβει ένα μεγάλο ποσοστό, αν όχι όλο το dataset, έτσι ο χρόνος να διαβαστεί ολόκληρο το dataset είναι πιο σημαντικός από τον λανθάνων χρόνο (latency) της ανάγνωσης της πρώτης εγγραφής.

Υπολογιστικός εξοπλισμός χαμηλού κόστους: Το Hadoop δεν απαιτεί ακριβό, υψηλής αξιοπιστίας hardware για να τρέξει. Είναι σχεδιασμένο για να τρέχει σε clusters από hardware του εμπορίου (hardware κοινά διαθέσιμο) από τους πολλαπλούς προμηθευτές για τους οποίους η πιθανότητα να αποτυγχάνουν οι κόμβοι του cluster είναι υψηλή, τουλάχιστον για τα μεγάλα clusters. Το HDFS όμως είναι σχεδιασμένο για να συνεχίσει να δουλεύει χωρίς τη σημαντική ανησυχία του χρήστη για μια τέτοια σοβαρή αποτυχία (failure).

Αξίζει επίσης να εξεταστούν οι περιπτώσεις στις οποίες όταν χρησιμοποιείται το HDFS δεν λειτουργούν τόσο καλά:

- **Καθυστέρηση μεταφοράς στην πρόσβαση δεδομένων (Low-latency data access)**

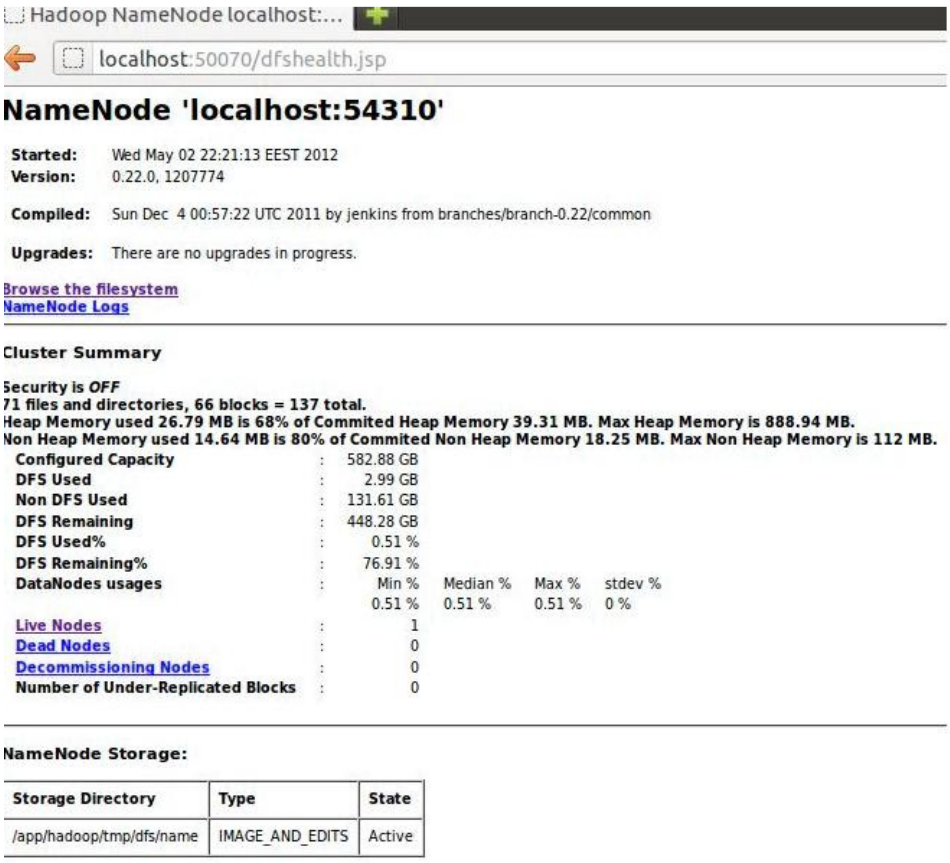
Εφαρμογές που απαιτούν καθυστέρηση μεταφοράς στην πρόσβαση δεδομένων, στο εύρος των δεκάτων του millisecond δεν θα λειτουργεί καλά με το HDFS. Το HDFS είναι βέλτιστο στη διαχείριση μεγάλου όγκου δεδομένων, και αυτό είναι σε βάρος της μεταφοράς.

- **Πολλά μικρά αρχεία**

Εφόσον το όνομα του κόμβου (namenode) κρατάει μεταδεδομένα του συστήματος αρχείων (filesystem) στη μνήμη, ο περιορισμός στον αριθμό των φακέλων σε ένα σύστημα αρχείων καθορίζεται από το μέγεθος της μνήμης στο namenode. Σαν έναν γενικό κανόνα, σε κάθε αρχείο (file), directory και block χρειάζεται περίπου 150. Έτσι, για παράδειγμα αν είχαμε ένα εκατομμύριο αρχεία, κάθε ένα σε ένα block, θα χρειαζόταν τουλάχιστον 300 MB μνήμης. Ενώ η αποθήκευση εκατομμυρίων φακέλων είναι εφικτή, τα δισεκατομμύρια είναι πέρα από την τρέχουσα ικανότητα του hardware.

- **Αρχεία πολλαπλών προελεύσεων και αυθαίρετη μορφοποίηση αρχείων**

Τα αρχεία του HDFS πρέπει να είναι δημιούργημα ενός χρήστη-συγγραφέα. Προς το παρόν, δεν παρέχεται υποστήριξη για πολλαπλούς χρήστες ή για αυθαίρετες παρεμβάσεις στη διαμόρφωση των αρχείων.



Εικόνα 2.11: Hadoop dfshealth

Έννοιες του HDFS

- **Blocks**

Κάθε δίσκος έχει ένα μέγεθος block, το οποίο είναι ο ελάχιστος αριθμός δεδομένων που μπορεί να διαβάσει ή να γράψει. Τα filesystems σε έναν απλό δίσκο χειρίζονται τα δεδομένα σε blocks, τα οποία είναι ακέραια πολλαπλάσια του μεγέθους ενός block. Τα blocks στα filesystem είναι συνήθως μερικά kilobytes σε μέγεθος, ενώ τα blocks των δίσκων είναι συνήθως 512 bytes. Αυτό είναι γενικά απαρατήρητο για τον χρήστη του filesystem που απλά γράφει ή διαβάζει έναν φάκελο οποιουδήποτε μεγέθους. Παρόλα αυτά υπάρχουνε εργαλεία για τη συντήρηση του filesystem, όπως το df και το fsck, που λειτουργούν σε επίπεδο filesystem block.

Επίσης το HDFS έχει την έννοια του block, αλλά σε πολύ μεγαλύτερο μέγεθος 64 MB εξ ορισμού. Όπως ένα filesystem για έναν απλό δίσκο, οι φάκελοι στο HDFS είναι διαιρεμένοι σε τμήματα μεγέθους block, όταν είναι αποθηκευμένοι σε ανεξάρτητες μονάδες. Όμως αντίθετα από ένα filesystem για έναν απλό δίσκο, το αρχείο στο HDFS που είναι μικρότερο από ένα block δεν καταλαμβάνει το χώρο ενός ολόκληρου block του μέσου αποθήκευσης. Στην παρούσα μεταπτυχιακή διατριβή όπου αναφερόμαστε σε "block" αναφερόμαστε στο σχετικό με το HDFS block.

Τα blocks του HDFS σε σχέση με blocks των δίσκων είναι μεγαλύτερα, διότι το ζητούμενο είναι η ελαχιστοποίηση του κόστους αναζήτησης. Ο χρόνος μεταφοράς δεδομένων από το δίσκο για ένα μεγάλο block μπορεί να γίνει αρκετά μεγαλύτερος από το χρόνο αναζήτησης από την αρχή του block. Συνεπώς, ο χρόνος που απαιτείται για να μεταφερθεί ένα αρχείο που αποτελείται από πολλά blocks, επηρεάζει το εύρος του χρόνου μεταφοράς του δίσκου. Πραγματοποιώντας ένα μικρό υπολογισμό, εάν ο χρόνος αναζήτησης είναι 10 ms και υπάρχει ένα εύρος μεταφοράς που είναι 100 MB/s, τότε για να πετύχουμε ο χρόνος αναζήτησης να είναι το 1% του χρόνου μεταφοράς, απαιτείται να κάνουμε το μέγεθος του block 100 MB. Το default είναι 64 MB, αν και πολλές εγκαταστάσεις HDFS, χρησιμοποιούν blocks των 128 MB.

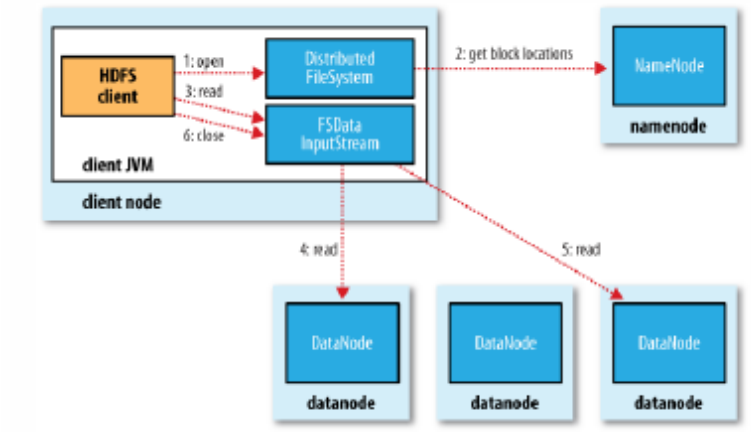
Ακόμη, χρησιμοποιώντας το block ως μονάδα αφαίρεσης αντί του αρχείου (file) απλοποιεί τη διαδικασία αποθήκευσης του υποσυστήματος. Η απλότητα είναι κάτι που επιδιώκεται για όλα τα συστήματα, αλλά είναι ιδιαίτερα σημαντική για ένα διασκορπισμένο σύστημα στο οποίο ποικίλουν τόσο πολύ οι αιτίες αστοχίας (failure modes). Το αποθηκευτικό σύστημα που χρησιμοποιεί τα blocks απλοποιεί τη διαχείριση της αποθήκευσης (εφόσον τα blocks έχουν έναν συγκεκριμένο μέγεθος είναι εύκολο να υπολογιστεί πόσα μπορούν να αποθηκευτούν σε έναν

συγκεκριμένο δίσκο) και να περιορίσει ότι αφορά τα μεταδεδομένα (τα blocks είναι ακριβώς ένα τμήμα δεδομένων για αποθήκευση στα μεταδεδομένα (metadata) των φακέλων όπως οι πληροφορίες για τις άδειες οι οποίες δεν χρειάζονται να είναι αποθηκευμένες στα blocks, έτσι ένα άλλο σύστημα μπορεί να χειρίζεται τα μεταδεδομένα χωριστά).

Ακόμη, τα blocks έχουν τη δυνατότητα να αντιγράφονται (replication), και ως εκ τούτου είναι πιο ανεκτικά σε λάθη και περισσότερο διαθέσιμα. Προκειμένου να εξασφαλιστεί το σύστημα από καταστροφή σε blocks, δίσκο, ή από αποτυχία της μηχανής, κάθε ένα block αντιγράφεται σε διαφορετικές μηχανές (συνήθως τρεις). Εφόσον κάποιο block δεν είναι διαθέσιμο, ένα από τα αντίγραφά του σε άλλο σημείο μπορεί να διαβαστεί έτσι ώστε να γίνει διαθέσιμο στον client.

- **Ανάγνωση αρχείου**

Στο παρακάτω Σχήμα 2.3.6 παρουσιάζεται ο τρόπος που ένα αρχείο διαβάζεται στο HDFS:



Σχήμα 2.12: Διαδικασία ανάγνωσης αρχείου στο HDFS

Βήμα 1: Ο client ανοίγει το αρχείο προς ανάγνωση.

Βήμα 2: Καλείται ο NameNode να ορίσει τις τοποθεσίες που βρίσκονται τα πρώτα blocks του αρχείου. Για κάθε ένα block, ο NameNode επιστρέφει τις διευθύνσεις των DataNodes που έχουν ένα αντίγραφο αυτού του block. Οι DataNodes ταξινομούνται βάσει της εγγύτητάς τους στον client. Εάν και ο ίδιος ο client είναι ένας DataNode, θα ανατρέξει και στον εαυτό του, να διαπιστώσει αν φιλοξενεί κάποιο αντίγραφο του block.

Βήμα 3: Έπειτα ο client καλεί εντολή ανάγνωσης, read.

Βήμα4: Γίνεται σύνδεση με τον πρώτο (και εγγύτερο) DataNode και για το αρχικό block του αρχείου. Τα δεδομένα διοχετεύονται από τον DataNode στον client, ο οποίος εκτελεί την εντολή ανάγνωσης (read) επαναληπτικά.

Βήμα 5: Όταν τελειώσει με το block κλείνει τη σύνδεση με το συγκεκριμένο DataNode και ψάχνει τον βέλτιστο DataNode για το επόμενο block.

Βήμα 6: Αυτή η διαδικασία πραγματοποιείται από τον client καθώς γίνεται διαρκής διοχέτευση δεδομένων σε αυτόν. Τα blocks διαβάζονται με τη σειρά και ανοίγουν σταδιακά τις νέες συνδέσεις με τους DataNodes, ενώ ταυτόχρονα στον client διοχετεύονται δεδομένα. Εκτελείται επίσης και το NameNode, προκειμένου να αναζητηθούν τα επόμενα blocks προς ανάγνωση. Τέλος, όταν ο client ολοκληρώσει την ανάγνωση, καλεί μια διαδικασία κλεισίματος (close).

2.3.6 Διεπαφές (interface) συστήματος αρχείων του Hadoop

Το Hadoop είναι γραμμένο σε Java και για όλες τις αλληλεπιδράσεις του filesystem του Hadoop μεσολαβεί το API της Java. Το κέλυφος του filesystem, για παράδειγμα είναι μια εφαρμογή της Java που χρησιμοποιεί το FileSystem class για να προμηθεύσει τις λειτουργίες του filesystem. Οι διεπαφές που συνήθως χρησιμοποιούνται με το HDFS, καθώς και το υπόλοιπο filesystem στο Hadoop συνήθως χρησιμοποιούν υπάρχοντα εργαλεία για την πρόσβαση στο underlying filesystem (FTP clients for FTP, S3 εργαλεία για S3, etc.), αλλά πολλά από αυτά θα λειτουργήσουν και με κάθε filesystem του Hadoop.

Thrift: Λειτουργώντας τη διεπαφή του filesystem στο API της Java, το Hadoop δυσκολεύει την περίπτωση των μη Java εφαρμογών για να αποκτήσουν πρόσβαση στα filesystems του. Το Thrift API στο "thriftfs" contrib module συμβάλλει στη θεραπεία αυτής της ενότητας με την έκθεση των Hadoop filesystems σαν υπηρεσία του Apache Thrift (service), διευκολύνοντας σε κάθε γλώσσα που έχει Thrift συνδέσεις να αλληλεπιδρά με ένα Hadoop filesystem, όπως είναι το HDFS.

C: Το Hadoop παρέχει μια βιβλιοθήκη της C που ονομάζεται libhdfs που αντικατοπτρίζει τη διεπαφή της Java FileSystem (ήταν γραμμένο σαν βιβλιοθήκη της C για την πρόσβαση στο HDFS, αλλά παρά το όνομά της μπορεί να χρησιμοποιηθεί για την πρόσβαση σε κάθε Hadoop filesystem). Λειτουργεί χρησιμοποιώντας τη διεπαφή Java Native Interface (JNI) για να καλέσει κάθε Java filesystem client.

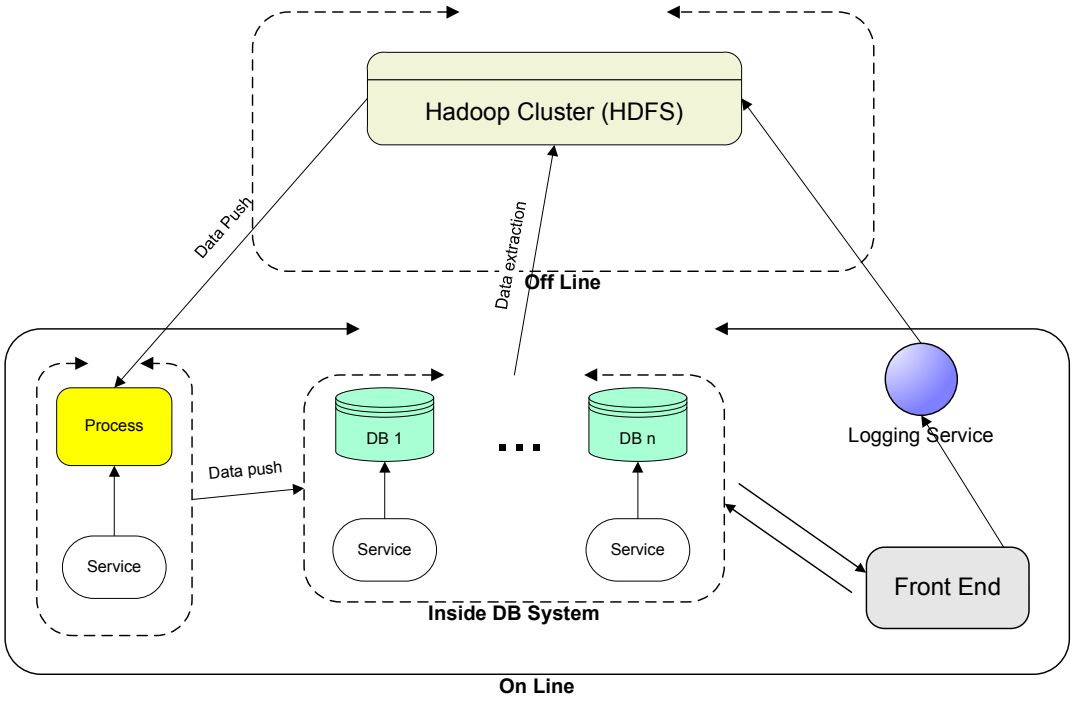
FUSE: Το filesystem in userspace επιτρέπει στα filesystems που εφαρμόζονται στο χώρο του χρήστη να ολοκληρώνονται σαν ένα Unix filesystem. Το Hadoop's Fuse-DFS contrib module επιτρέπει σε κάθε Hadoop filesystem (αλλά συνήθως στο HDFS) να τοποθετηθεί σαν ένα πρότυπο (standard) filesystem.

WebDAV: είναι ένα σύνολο επεκτάσεων στο HTTP για να υποστηρίξει την επεξεργασία και την ενημέρωση των αρχείων. Τέλος, υπάρχουν και τα http & ftp .

2.3.7 Κοινωνικά Δίκτυα & Hadoop: Μία προσέγγιση ενός off line συστήματος

Τα τελευταία χρόνια έχει παρατηρηθεί μια γεωμετρική αύξηση του όγκου των δεδομένων σε sites κοινωνικής δικτύωσης[2]. Οι μέχρι τότε μέθοδοι και εργαλεία έδειξαν την αντοχή τους στην διαχείριση μεγάλου όγκου δεδομένων. Για να βελτιωθούν οι υπηρεσίες ξεκίνησε η ανάπτυξη νέων μεθόδων και εργαλείων και μερικά από αυτά περιγράφονται στην παρούσα διατριβή.

Για την βελτίωση της απόδοσης του συστήματος αλλά και για την κάλυψη και μελλοντικών αλλαγών, ήρθε η ενσωμάτωση του Hadoop στην αρχιτεκτονική των μέχρι τώρα συστημάτων επεξεργασίας των δεδομένων των κοινωνικών δικτύων.



Εικόνα 2.13: Hadoop off line system

[2] Big Data Analytics Architecture Putting All Your Eggs in Three Baskets" by Neil Raden, Sponsored by: Hired Brains, Inc.

Μελετώντας τόσο της υπάρχουσες εγκαταστάσεις αλλά και ερευνητικές προτάσεις για το θέμα, η προσέγγιση που υιοθετήθηκε για την λύση του αποτυπώνεται στο παραπάνω σχήμα. Η προσθήκη του Hadoop σε online συστήματα δεν ήταν κάτι για το οποίο είχε σχεδιασθεί και συνεπώς δεν μπορούσε να χρησιμοποιηθεί ως τέτοιο. Έτσι χρησιμοποιείται σε offline κατάσταση, όπου λαμβάνει τα προς επεξεργασία δεδομένα και με την χρήση των κατάλληλων εργαλείων, τα επεξεργάζεται και στην συνέχεια με την βοήθεια κατάλληλων διαδικασιών, αυτά μεταφέρονται στο on line σύστημα προς διάθεση. Στο σχήμα δίνεται μια προσέγγιση για το πώς το συγκεκριμένο εργαλείο ενσωματώνεται και πως βοηθά την λειτουργία των sites της κοινωνικής δικτύωσης.

Στις επόμενες γραμμές θα δοθεί μια απλοποιημένη περιγραφή του πως χρησιμοποιείται το Hadoop στο σύστημα του Twitter. Οι χρήστες μέσω του front end συστήματος του portal καταχωρούν τα tweets ή διαβάζουν tweets των χρηστών που ακολουθούν.

Για την αποφυγή προβλημάτων επεξεργασίας στο online σύστημα τα δεδομένα καταγράφονται σε έναν logging σύστημα. Η μορφή των δεδομένων που αποθηκεύονται συνήθως είναι σε text μορφή. Στην συνέχεια αυτά τα δεδομένα σε συνδυασμό με τα δεδομένα από το online σύστημα επεξεργάζονται και μέσω συγκεκριμένων process (διαδικασιών), μεταφέρονται οι ανανεωμένες πληροφορίες στο online σύστημα. Με αυτό τον τρόπο επεξεργάζεται μεγάλος όγκος δεδομένων χωρίς να επηρεάζεται η αποδοτικότητα του online συστήματος.

2.3.8 Αστοχίες του Map Reduce

Οι αστοχίες (failures) που είναι πιθανόν να εμφανιστούν κατά την φάση διεργασιών στο Hadoop, οφείλονται στην πραγματικότητα στον κώδικα του χρήστη, όπου μπορεί να έχει λάθη και έχει ως συνέπεια να καταρρέουν οι διαδικασίες (crash). Ένα από τα μεγαλύτερα πλεονεκτήματα χρησιμοποιώντας το Hadoop είναι η ικανότητά του να διαχειρίζεται αυτές τις αστοχίες και να επιτρέπει στην εργασία που του έχει ανατεθεί να ολοκληρώνεται.

Μία ακόμα αστοχία είναι αυτή της ίδιας της διεργασίας (task failure). Αυτό μπορεί να συμβεί όταν στον κώδικα, είτε στο map ή στο reduce task υπάρχει runtime exception.

Αν ένας tasktracker αποτύχει, ή τρέχει πολύ αργά, θα σταματήσει να στέλνει εισροές (heartbeats) στον jobtracker (ή θα τις στέλνει ασύγχρονα). Ο jobtracker θα σημειώσει ότι ο tasktracker έχει σταματήσει να στέλνει εισροές (εάν δεν έχει λάβει μία για δέκα λεπτά, διαμορφωμένα μέσω του

mapred.task tracker.expiry.interval property, σε milliseconds) και θα αφαιρέσει τη δυνατότητα από την δεξαμενή των tasktrackers να προγραμματίσουν tasks σε αυτόν.

Η αστοχία ενός jobtracker είναι η πιο σοβαρή αστοχία που μπορεί να συμβεί. Προς το παρόν το Hadoop δεν έχει μηχανισμό για να διαχειριστή αυτού του είδους την αστοχία, και έτσι σε αυτήν την περίπτωση η εργασία αποτυγχάνει.

Με τον χρονοπρογραμματισμό εργασιών (job scheduling) που προστέθηκε στις πρόσφατες εκδόσεις του Hadoop υπάρχει μια πολύ απλή προσέγγιση για να προγραμματιστούν οι εργασίες από τον χρήστη. Τρέχουν με τη σειρά αποστολής, χρησιμοποιώντας έναν FIFO scheduler. Με αυτόν τον τρόπο ενώ στις προηγούμενες εκδόσεις κάθε εργασία χρησιμοποιούσε όλον το cluster για να τρέξει τις διεργασίες με αποτέλεσμα οι εργασίες να περιμένουν τη σειρά τους για να τρέξουν, με αυτή την βελτίωση, αυτό αποφεύγεται.

2.4 Χαρακτηριστικά των χρησιμοποιούμενων Datasets

2.4.1 Το BerkStan dataset (Berkeley-Stanford web graph)

Αυτό το dataset συλλέχθηκε το 2002 και αποτελείται από κόμβους (nodes) οι οποίοι αντιπροσωπεύουν σελίδες από τα domains των berkely.edu και stanford.edu με κατευθυνόμενες σχέσεις που αντιπροσωπεύουν hyperlinks ανάμεσά τους.

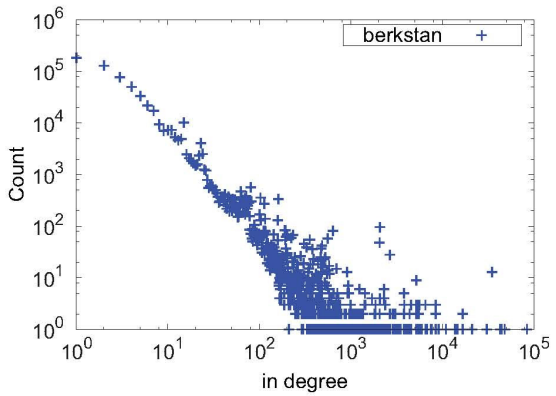
Στατιστικά του Dataset	
Κόμβοι (Nodes)	685.230
Σχέσεις (Edges)	7.600.595
Κόμβοι στο μεγαλύτερο WCC	654.782 (0,956)
Συνδέσεις στο μεγαλύτερο WCC	7.499.425 (0,987)
Κόμβοι στο μεγαλύτερο SCC	334.857 (0,489)
Συνδέσεις στο μεγαλύτερο SCC	4.523.232 (0,595)
Μέσος συντελεστής ομαδοποίησης (Average clustering coefficient)	0,6149

Αριθμός των τριγώνων	64.690.980
Ποσοστό των κλειστών τριγώνων	0,08769
Διάμετρος (longest shortest path)	669
90% αποτελεσματική διάμετρος-(90 percentile effective diameter)	10
Αρχείο (File)	Περιγραφή
http://snap.stanford.edu/data/web-BerkStan.html web-BerkStan.txt.gz	Berkely-Stanford web graph, 2002

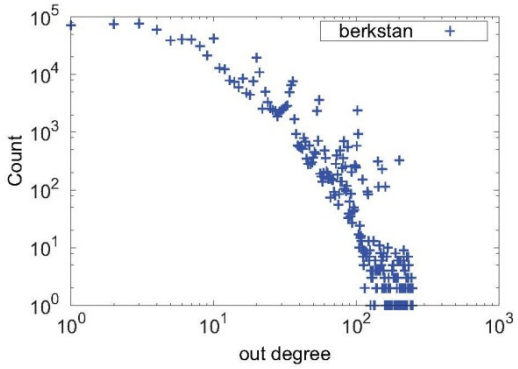
Πηγή: J. Leskovec et al, [17]

BerkStan (Berkeley-Stanford) web graph

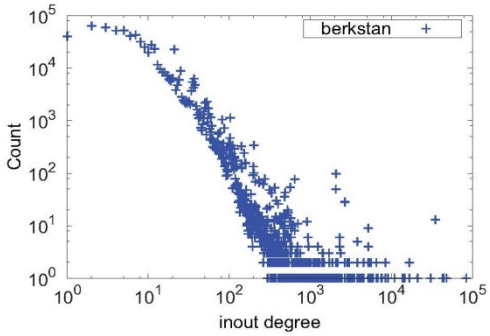
Για την γραφική αποτύπωση των συγκεκριμένων δεδομένων χρησιμοποιήθηκε το Pegasus. Για την εξαγωγή των γραφημάτων πρώτα ξεκινά η διαδικασία «Compute» όπου έγιναν οι απαραίτητοι υπολογισμοί με τις ανάλογες παραμέτρους (in degree, out degree, in-out degree) και στην συνέχεια ακολουθεί η διαδικασία «plot», όπου και παίρνουμε τα παρακάτω αποτελέσματα. Δίνονται τρείς παραστάσεις, μία για κάθε παράμετρο.



Εικόνα 2.14: BerkStan (in Degree) plot

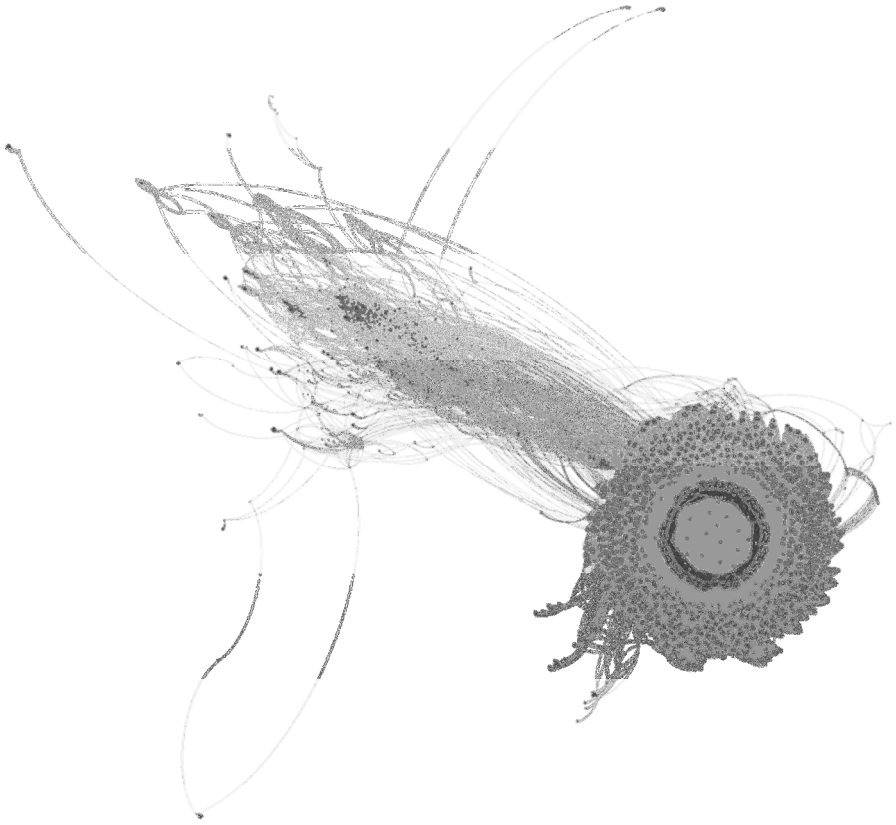


Εικόνα 2.15: BerkStan (out Degree) plot



Εικόνα 2.16: BerkStan (inout Degree) plot

Στην συνέχεια το ίδιο data set αποτυπώθηκε γραφικά με το εργαλείο Gephi. Το αποτέλεσμα που εξήχθη, δίνεται στο παρακάτω σχήμα. Είναι φανερό ότι μετά από 10 μέρες εκτέλεσης του συγκεκριμένου εργαλείου απεικόνισης το αποτέλεσμα δεν είναι καθόλου ικανοποιητικό. Το συμπέρασμά μας είναι ότι δεν επαρκεί το εργαλείο Gephi για την οπτική απεικόνιση μεγάλου όγκου δεδομένων σε λογικό χρόνο.



Εικόνα 2.17: Γραφική αποτύπωση berkstan dataset μέσω του εργαλείου Gephi

2.4.2 Το Twitter dataset

Το Twitter είναι μια online υπηρεσία κοινωνικής δικτύωσης που επιτρέπει στους χρήστες που είναι εγγεγραμμένοι σε αυτό να στέλνουν και να διαβάζουν μηνύματα , γνωστά και ως «tweets». Ιδρύθηκε το Μάρτιο του 2006 από τον Jack Dorsey. Είναι από τα δημοφιλέστερα κοινωνικά δίκτυα, με πάνω από 300 εκατομμύρια χρήστες έως το 2011. Παράγει πάνω από 300 εκατομμύρια tweets και διαχειρίζεται πάνω από 1,6 δισ. ερωτήματα αναζήτησης ανά ημέρα.

Αντίθετα με ότι συμβαίνει με το Facebook ή άλλα δίκτυα κοινωνικής δικτύωσης η σχέση που ισχύει στο Twitter, δηλαδή το να ακολουθείς ή να σε ακολουθούν, δεν απαιτεί ανταπόδοση. Ένας χρήστης μπορεί να ακολουθήσει οποιονδήποτε άλλο χρήστη και ο χρήστης που τον ακολουθούν δεν πρέπει να ακολουθήσει και αυτός. Ένας follower (αυτός που ακολουθεί) στο Twitter λαμβάνει όλα τα μηνύματα (που ονομάζονται tweets) από αυτούς που ακολουθεί (followings).

Υπάρχει επίσης συγκεκριμένη ορολογία για την λειτουργία του Twiiter: RT σημαίνει retweet, ένα @ ακολουθούμενο με την διεύθυνση ταυτότητας είναι ο χρήστης, # ακολουθούμενο από μια λέξη αντιπροσωπεύει ένα hashtag. Το hashtag είναι μια σύμβαση ανάμεσα στους χρήστες του Twitter για να δημιουργήσουν ή να ακολουθήσουν ένα νήμα συζητήσεων. Αυτό το συγκεκριμένο λεξιλόγιο περιορίζεται με ένα αυστηρό όριο των 140 χαρακτήρων ανά ανάρτηση (posting) διευκολύνοντας τους χρήστες να είναι σύντομοι και περιεκτικοί στην διατύπωση τους. Επιπλέον, ο μηχανισμός retweet επιτρέπει στους χρήστες να διασπείρουν πληροφορία της επιλογής τους που όμως δεν είναι αρχικά δική τους.

Το δίκτυο Twitter μπορεί να αναπαρασταθεί ως κατευθυνόμενος γράφος του οποίου τα χαρακτηριστικά[3] δεν έχουν μελετηθεί πλήρως ακόμη. Μια πρώτη εκτίμηση έχει γίνει από τους [Haewoon Kwak et. al., 13]. Τα βασικά συμπεράσματα των παραπάνω ερευνητών συνοψίζονται στα εξής:

1. Το δίκτυο αυτό περιμένουμε να έχει χαμηλό συντελεστή ομαδοποίησης (cluster coefficient), καθώς δεν απαιτεί αμοιβαίες σχέσεις και η δημιουργία των κλικών – ομάδων δεν ευνοείται. Η ανταποδοτικότητα δεν είναι στα χαρακτηριστικά του Twitter: 77,9% των χρηστών έχουν μία μονοδιάστατη σχέση following-follower και μόνο 22,1% έχουν

[3] Το δίκτυο Twitter αναλύθηκε διεξοδικά και για πρώτη φορά από τους Haewoon et al [13], των οποίων και τα συμπεράσματα συνοψίζουμε

ανταποδοτική σχέση μεταξύ τους. Παρόλα αυτά ο βαθμός διαχωρισμού (Degree of Separation) που μετρά πόσοι άνθρωποι μεσολαβούν ανάμεσα σε δύο άγνωστους προκειμένου να έρθουν σε επαφή, είναι ιδιαίτερα μικρός (4,2), σε κάθε περίπτωση μικρότερος από ότι αναμενόταν.

2. Θα πρέπει να υπάρχουν δύο συσσωρεύσεις (picks) στις περιγραφικές κατανομές που θα εμφανίζονται στα διαγράμματα και ειδικότερα στο indegree μέτρο το οποίο αφορά τους followings: Το Twitter προτείνει ένα σύνολο από 20 ανθρώπους που ο νεοεισερχόμενος μπορεί να ακολουθήσει με ένα απλό κλικ κι έτσι πολλοί άνθρωποι χρησιμοποιούν αυτή τη δυνατότητα. Οπότε αναμένουμε ένα pick στους 20 followings. Επιπλέον, πριν το 2009 υπήρχε ένα ανώτατο όριο στους ανθρώπους που κάποιος μπορούσε να ακολουθήσει (http://blog.hubspot.com/Portals/249/sotwitter09.pdf), Το Twitter περιόριζε τους followings σε ένα μέγιστο των 2000 μέχρι ο καθένας να ξεπεράσει τους 2000 followers. Έτσι συσσωρεύονταν για κάποιο χρονικό διάστημα 2000 followings περιμένοντας ο καθένας τους να ξεπεράσει τους 2000 followers. Σήμερα αυτό το όριο δεν υπάρχει.

3. Οι κατανομές των περιγραφικών μέτρων (degree, indegree, outdegree και inoutdegree) περιμένουμε να ακολουθούν την αρνητική εκθετική κατανομή με μακριά ουρά (a power-law or "long tail" curve).

4. Η «ουρά» της κατανομής είναι μεγαλύτερη από ότι τα μαθηματικά προβλέπουν, καθώς πάρα πολλοί followers συγκεντρώνονται σε λίγους followings όπως πολιτικούς, μέσα μαζικής ενημέρωσης, και διάσημους. Υπάρχουν μόλις 40 χρήστες με πάνω από ένα εκατομμύριο followers και αυτοί είναι διάσημοι ή μέσα μαζικής ενημέρωσης.

5. Όσον αφορά τη μορφή του δικτύου, αυτή σε μεγάλο βαθμό καθορίζεται από τα re-tweets (προώθηση των αρχικών tweets). Αποδεικνύεται ότι ανεξάρτητα από το πόσους αρχικούς followers έχει ένα tweet, μέσω των retweets φθάνει τους 1000 χρήστες, με ταχύτατα βήματα έως και 4 φορές εφόσον ξεκινήσει η διαδικασία retweet. Τα δίκτυα των retweets αποτελούν υποσύνολα του δικτύου του Twitter.

Ποια είναι τα χαρακτηριστικά του Twitter dataset

Το Twitter dataset, συλλέχθηκε το 2010 από τον Hawoon et. al [13] και είναι έκτοτε διαθέσιμο στο http://an.kaist.ac.kr/traces/WWW2010.html.

Περιγραφή των δεδομένων.

http://snap.stanford.edu/data/twitter7.html

Τα δεδομένα συλλέχθηκαν από την ομάδα των Hawoon et. al [13] και έχουν τα εξής χαρακτηριστικά: Για 467 εκατομμύρια αναρτήσεις Twitter από 20 εκατομμύρια χρήστες σε μια επτάμηνη περίοδο από την 1η Ιουνίου του 2009 μέχρι το 31 Δεκεμβρίου του 2009. Εκτιμάται ότι καταγράφηκε περί το 20-30% όλων των δημόσιων αναρτήσεων (public tweets) που δημοσιεύθηκαν στο Twitter στη διάρκεια του συγκεκριμένου χρονικού διαστήματος.

Για κάθε public tweet είναι διαθέσιμες στους αρχικούς ερευνητές οι παρακάτω ιδιότητες:

1. Συγγραφέας
2. Χρόνος
3. Περιεχόμενο

Στατιστικά του αρχικού dataset

Αριθμός χρηστών (users)	17.069.982
Αριθμός tweets	476.553.560
Αριθμός URLs	181.611.080
Αριθμός Hashtags	49.293.684
Αριθμός re-tweets	71.835.017

Έπειτα από απαίτηση του Twitter [Andrey Watters, [3]] απαγορεύτηκε η διαθεσιμότητα αυτών των δεδομένων δημόσια και περιορίστηκε στα στοιχεία του κοινωνικού γράφου. Ο κοινωνικός γράφος (ποιος ακολουθεί ποιόν) του Twitter αποτελεί το dataset της παρούσης διατριβής και η πρώτη του ανάλυση είναι διαθέσιμη από τους Haewoon Kwak, et al. [13]. Έτσι τα χαρακτηριστικά του dataset που χρησιμοποιείται στην παρούσα μεταπτυχιακή διατριβή περιορίζονται στον αριθμό των χρηστών και τη συσχέτισή τους (ποιος ακολουθεί ποιόν). Είναι διαθέσιμα στο

http://an.kaist.ac.kr/traces/WWW2010.html

και τα χαρακτηριστικά τους είναι τα εξής:

Format

USER \t FOLLOWER \n

* ο USER και FOLLOWER αναπαριστώνται με ακέραιους αριθμούς ID (integer).
* Αυτά τα IDs είναι τα ίδια με αυτά που χρησιμοποιεί το Twitter.
* Έτσι θα μπορούσε κανείς να βρει το προφίλ του χρήστη 12 μέσω:
http://api.twitter.com/1/users/show.xml?user_id=12.
* Για λεπτομέρειες : Twitter API Page

Παράδειγμα δομής δεδομένων:

12 13
12 14
12 15
16 17

* Οι χρήστες 13, 14 και 15 (followers) ακολουθούν τον χρήστη 12 (user).
* Ο χρήστης 17 είναι follower του χρήστη 16.

Στατιστικά του αρχικού dataset
Αριθμός χρηστών (users) 17,069,982
Αριθμός tweets 476,553,560

2.4.3 Παράδειγμα οπτικής αναπαράστασης των δεδομένων.

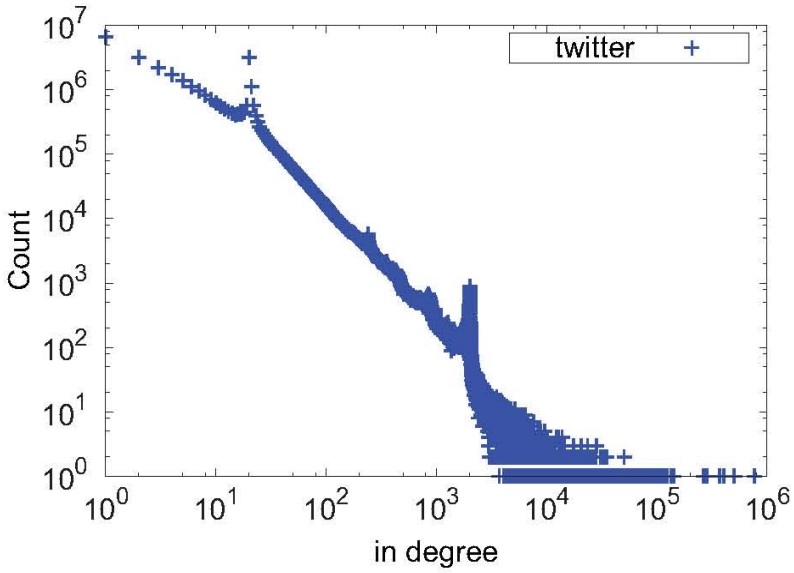
Παράδειγμα εφαρμογής μέτρου degree μέσω της εφαρμογής του Pegasus στο Twitter data set

Στις παρακάτω γραμμές δίνεται η γραφική απεικόνιση των δεδομένων με το Pegasus. Το Pegasus είναι ένα περιβάλλον γραφικής απεικόνισης για μεγάλης κλίμακας δεδομένων (a Peta-scale graph mining system). Τρέχει μέσα από το περιβάλλον του Hadoop.

Για τις ανάγκες του παραδείγματος χρησιμοποιήθηκε το μέτρο degree όπως δίνεται από το ίδιο το εργαλείο. Αφού φορτώθηκε τα αρχείο επιλέχθηκε το degree και έγινε υπολογισμός σε 3 φάσεις (in, out και in-out). Καθώς το Twitter αποτελεί κατευθυνόμενο γράφο και είναι απαραίτητα και τα τρία αυτά μέτρα για να έχουμε ένα διαυγές αποτέλεσμα.

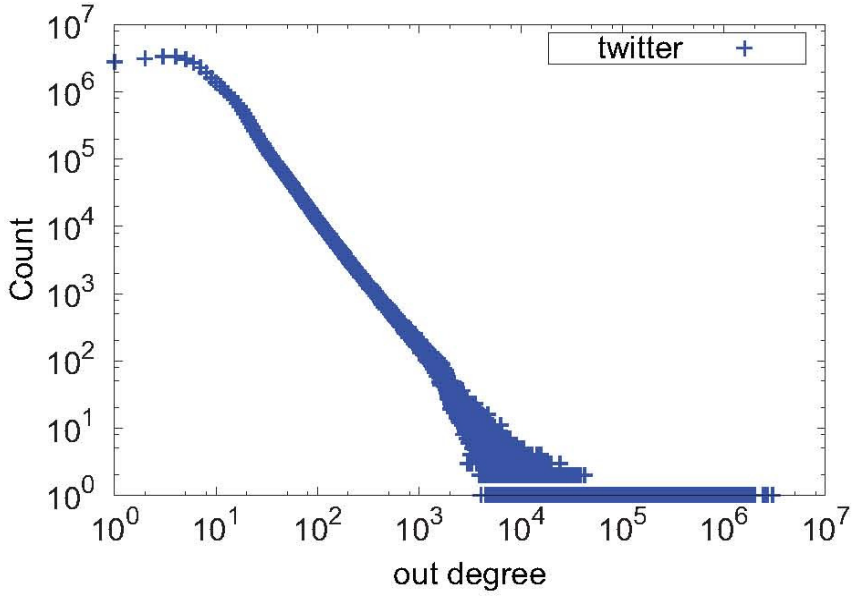
Στην συνέχεια έγινε γραφική απεικόνιση (plot) και στις παρακάτω απεικονίσεις δίνονται τα αποτελέσματα.

Degree (in)



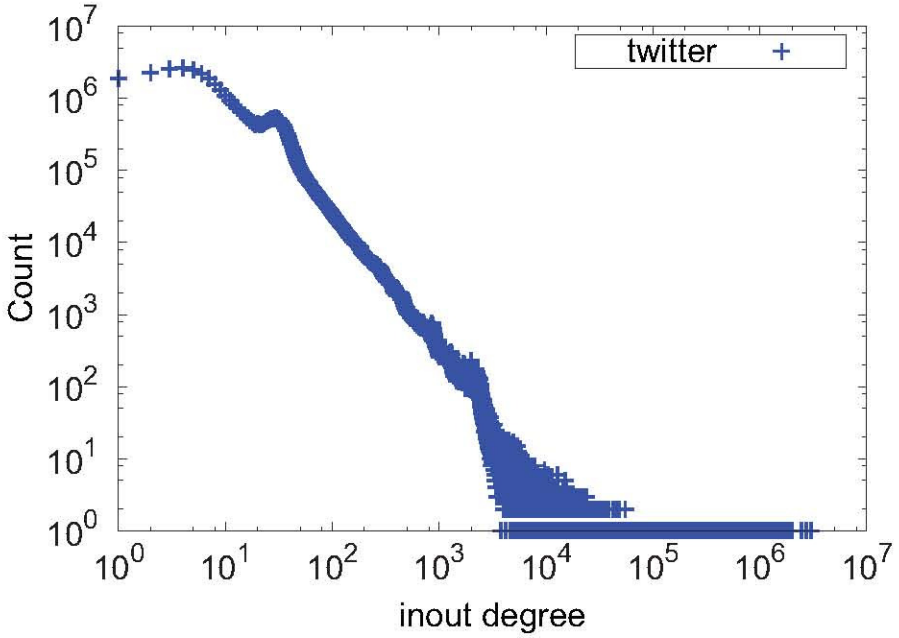
Εικόνα 2.18: Twitter (in Degree) plot

Degree (out)



Εικόνα 2.19: Twitter (out Degree) plot

Degree (inout)



Εικόνα 2.20: Twitter (inout Degree) plot

Παρατηρούμε ότι οι προβλέψεις μας για την κατανομή του in-degree (δηλαδή τον αριθμό των σχέσεων που έχει κάθε κόμβος με κατεύθυνση προς αυτόν - following) επαληθεύονται. Εμφανίζονται δύο σημεία στο διάγραμμα του in-degree τα οποία μεταφέρονται και στο συνολικό in-out degree όπου υπάρχει συσσώρευση των δεδομένων. Τα σημεία αυτά αντιστοιχούν αφενός στη χρήση της δυνατότητας που δίνει το Twitter στον νεοεισερχόμενο follower να επιλέξει τον αριθμό των 20 followings και αφετέρου, πριν το 2009 υπήρχε ένα ανώτατο όριο στους ανθρώπους που κάποιος μπορούσε να ακολουθήσει, οι followings είχαν μέγιστο όριο τους 2000 μέχρι ο καθένας να ξεπεράσει τους 2000 followers. Έτσι συσσωρεύονταν για κάποιο χρονικό διάστημα 2000 followings περιμένοντας ο καθένας τους να ξεπεράσει τους 2000 followers.

Ακόμη, βλέπουμε την κατανομή power-law να οπτικοποιείται αλλά με μεγάλη απόκλιση στην ουρά: Υπάρχει πολύ μεγάλη συσσώρευση followers σε μικρό αριθμό followings (μόλις 40 followings με πάνω από 1.000.000 followers). Αυτό αντικατοπτρίζεται τόσο στην απεικόνιση του in-degree, του out-degree αλλά και στο συσωρευτικό in-out degree.

Κεφάλαιο 3
Αλγόριθμοι & μετρικές σε γράφους

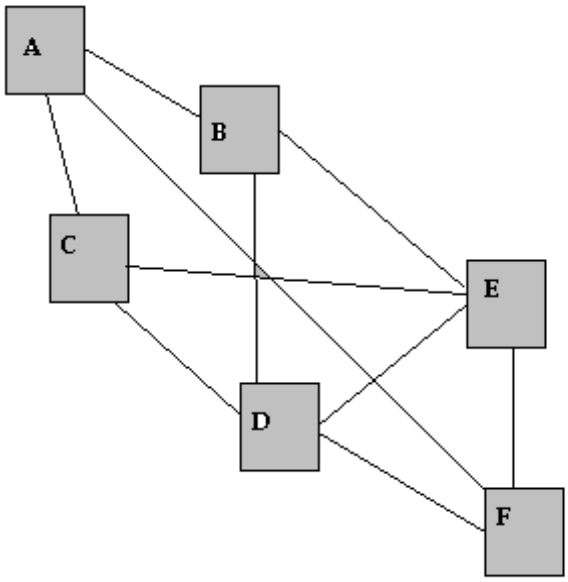
Στο κεφάλαιο αυτό περιγράφεται ο τρόπος που μοντελοποιούνται τα κοινωνικά δίκτυα ώστε να γίνουν διαχειρίσιμα από τις υπολογιστικές μηχανές, και στη συνέχεια αναφέρονται οι βασικοί αλγόριθμοι ομαδοποίησης των γράφων που εφαρμόζονται στα κοινωνικά δίκτυα. Οι αλγόριθμοι που χρησιμοποιούνται είναι οι σειριακοί και αυτοί του υπολογισμού των τριγώνων. Τελικά θα παρουσιάσουμε αναλυτικά αυτόν που επιλέξαμε στην παρούσα μεταπτυχιακή διατριβή και θα τεκμηριώσουμε την επιλογή του.

3.1 Κοινωνικοί γράφοι

Τα κοινωνικά δίκτυο συνήθως μοντελοποιούνται με διαγράμματα (ή γράφους). Αποτελούνται από κόμβους και τις γραμμές που τους ενώνουν και οι οποίες ορίζουν τις σχέσεις που

χαρακτηρίζουν το δίκτυο. Οι σχέσεις αυτές μπορούν να είναι χωρίς κατεύθυνση, κατευθυνόμενες όταν αντικατοπτρίζουν κατευθυνόμενα κοινωνικά δίκτυα (πχ Twitter), διμερείς, με βάρη, κλπ.

Τα κοινωνικά δίκτυα μοντελοποιούνται ως γράφοι. Οι οντότητες που απαρτίζουν τον γράφο είναι οι κόμβοι και οι ακμές που συνδέουν τους κόμβους μεταξύ τους με μια κατηγορία σχέσεων που χαρακτηρίζει το δίκτυο. Με τη μαθηματική έννοια του όρου ένας γράφος είναι ένα διατεταγμένο σύνολο G = (V, E) που περιλαμβάνει ένα σύνολο από V κορυφές ή κόμβους μαζί με ένα σύνολο από E σχέσεις ή ακμές, οι οποίες αποτελούν υποσύνολα διπλών στοιχείων του συνόλου V (μία σχέση συνδέει δύο κόμβους και αναπαριστάται σαν το μη-κατευθυνόμενο ζευγάρι αυτών των κόμβων). Αυτός ο τύπος γράφου περιγράφεται σαν μη-κατευθυνόμενος και απλός (βλ. επόμενο σχήμα 3.1).

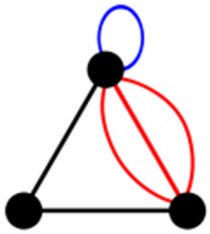


Σχήμα 3.1: Παράδειγμα γράφου με 6 κόμβους και 10 συνδέσεις

Ο προηγούμενος γράφος σύμφωνα με την μαθηματική απεικόνιση αποτελείται από τα εξής σύνολα:
V = {A, B, C, D, E, F}
E = {{A, B}, {A, C}, {A, F}, {B, A}, {B, D}, {B, E}, {C, A}, {C, D},{C, E},
{D, B},{D, C},{D, E},{D, F},{E, B},{E, C}, {E, D}, {E, F}, {F, A}, {F, D}, {F, E}}.

Άλλες έννοιες του γράφου προέρχονται από διαφορετικές αντιλήψεις για το σύνολο των σχέσεων E. Σε έναν άλλο πιο γενικό ορισμό το E είναι ένα πολύ-σύνολο (multiset) από μη ταξινομημένα ζευγάρια από (όχι απαραίτητα μοναδικούς) κόμβους. Πολλοί συγγραφείς

ονομάζουν αυτό τον τύπο του αντικειμένου σαν πολύ-γράφο ή ψευδο-γράφο. Υπάρχουν πολύ πιο πολύπλοκες μορφές γράφων όπως στο παράδειγμα του σχήματος 3.2.



Σχήμα 3.2: Γενικό παράδειγμα γράφου (στην πραγματικότητα ψευδογράφου) με τρεις κόμβους και έξι συνδέσεις

Ένας κόμβος μπορεί να ανήκει σε έναν γράφο και να μην ανήκει σε σχέση. Τα σύνολα V και E συνήθως είναι πεπερασμένα, Η τάξη (order) ενός γράφου είναι $|V|$ (ο αριθμός των κόμβων). Το μέγεθος ενός γράφου είναι $|E|$, ο αριθμός των σχέσεων. Ο βαθμός (degree) ενός κόμβου είναι ο αριθμός των σχέσεων που είναι συνδεδεμένες σε αυτόν, όπου μια σχέση που συνδέει τον κόμβο με τον εαυτό του ονομάζεται βρόχος (loop) και μετριέται δύο φορές. Για μια σχέση {u, v}, από τους θεωρητικούς χρησιμοποιείται η συντομογραφία uv. Συνήθως, στη σύγχρονη ορολογία της θεωρίας των γράφων, εκτός αν οριστεί διαφορετικά «γράφος» σημαίνει "μη-κατευθυνόμενος απλός πεπερασμένος γράφος " .

- **Σχέση γειτνίασης**

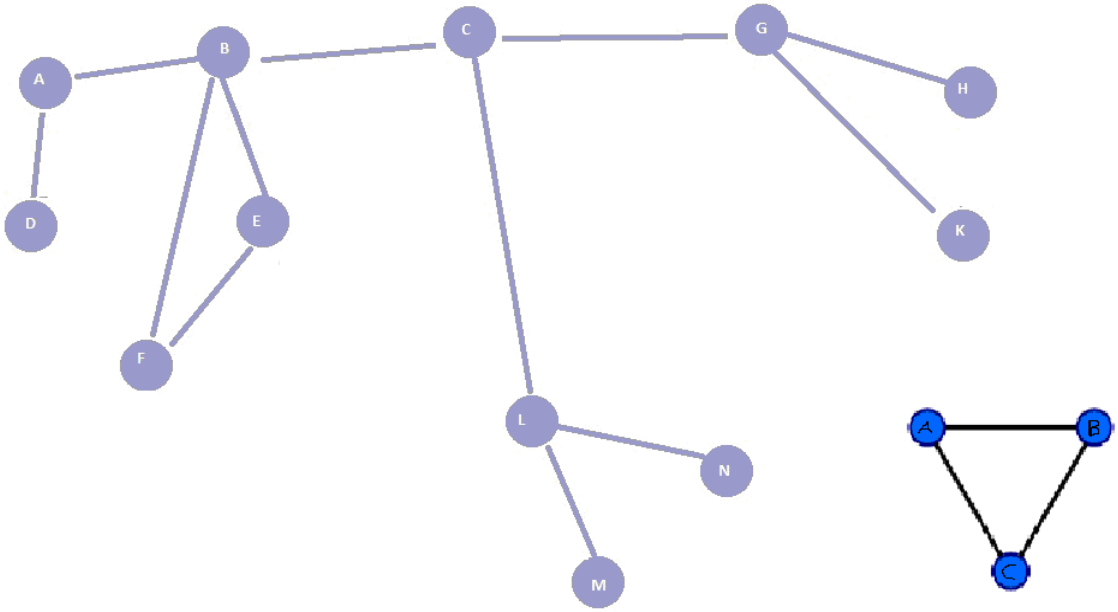
Οι σχέσεις E ενός μη κατευθυνόμενου γράφου G προκαλούν μια συμμετρική δυαδική σχέση ~ στο V που ονομάζεται σχέση γειτνίασης στο G. Ειδικότερα, για κάθε σχέση {u, v} υπάρχουν κόμβοι u και v που ονομάζονται γείτονες ο ένας στον άλλον και συμβολίζεται με u ~ v.

- **Κατηγορίες γράφων**

Οι γράφοι μοντελοποιούν με επιτυχία τα κοινωνικά δίκτυα και ακολουθούν την κατηγοριοποίησή τους.

1. Οι Μη-κατευθυνόμενοι γράφοι αναπαριστούν μη κατευθυνόμενα κοινωνικά δίκτυα (Undirected): Οι μη κατευθυνόμενοι γράφοι είναι η πιο απλή περίπτωση. Κοινωνικά

δίκτυα όπως το Facebook μπορούν να αναπαρασταθούν με ακρίβεια από τους μη κατευθυνόμενους γράφους. Στους γράφους αυτούς οι σχέσεις ανάμεσα στους κόμβους είναι μοναδικές, απλές συνδέσεις χωρίς κατεύθυνση ή άλλο χαρακτηριστικό (βλ. και Σχήμα 3.3). Στους μη κατευθυνόμενους γράφους, η σχέση (a, b) είναι όμοια με την σχέση (b, a), δηλαδή δεν είναι κατευθυνόμενα τα ζευγάρια, αλλά σύνολα {u, v} (ή πολύ-σύνολα των 2) των κόμβων.



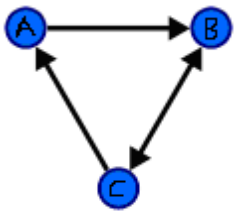
Σχήμα 3.3: Παραδείγματα μη κατευθυνόμενων γράφων

Παρατηρώντας τον πρώτο παραπάνω γράφο, αποτελείται από 12 κόμβους, 12 σχέσεις και μπορούμε να διακρίνουμε τρεις κοινωνικές ομάδες. Ο συνδετικός «κρίκος» μεταξύ αυτών των ομάδων είναι ο κόμβος "C". Ο μικρότερος τριγωνικός γράφος αποτελείται από τρεις κόμβους και τρεις σχέσεις, κάθε κόμβος έχει βαθμό 2.

Σε έναν μη κατευθυνόμενο γράφο, G, δύο κόμβοι u και v ονομάζονται συνδεδεμένοι αν το G περιέχει ένα μονοπάτι από το u προς το v. Διαφορετικά ονομάζονται μη-συνδεδεμένοι. Ένας γράφος ονομάζεται συνδεδεμένος αν κάθε ζευγάρι διακριτών κόμβων στο γράφο είναι συνδεδεμένο. Ένας γράφος ονομάζεται συνδεδεμένος σε k-κορυφή ή k-σχέση αν δεν υπάρχει σύνολο από k-1 κόμβους (ή αντίστοιχα σχέσεις) το οποίο όταν αφαιρεθεί διαλύει τον γράφο.

2. Οι κατευθυνόμενοι (Directed) γράφοι αναπαριστούν κατευθυνόμενα δίκτυα όπως το Twitter όπου οι σχέσεις ανάμεσα στους κόμβους έχουν διεύθυνση. Στο Twitter υπάρχουν

οι followers οι οποίοι ακολουθούν τους followings, με τη σχέση «ακολουθώ» ορίζεται η κατεύθυνση του δικτύου και του γράφου. Η κατεύθυνση ανάμεσα σε δύο κόμβους μπορεί να είναι προς τον ένα ή τον άλλο κόμβο ή ανάμεσα και στους δύο κόμβους (βλ. και Σχήμα 3.4)



Σχήμα 3.4: Παράδειγμα κατευθυνόμενου γράφου

Ένας κατευθυνόμενος γράφος (digraph) είναι ένα διατεταγμένο σύνολο D = (V, A) όπου V το σύνολο των κόμβων ή κορυφών και A σύνολο διατεταγμένων ζευγαριών κόμβων που ονομάζονται κατευθυνόμενες σχέσεις ή τόξα .Ένα τόξο a = (x, y) θεωρείται ως κατευθυνόμενο από το x στο y; το x ονομάζεται το κεφάλι και το y ονομάζεται η ουρά του τόξου; Το y θωρείται άμεσος διάδοχος του x, και το x θεωρείται άμεσος προκάτοχος y. Αν ένα μονοπάτι οδηγεί από το x στο y, τότε ο y θεωρείται διάδοχος του x και προσπελάσιμος από τον x, και ο x θεωρείται προκάτοχος του y. Το τόξο (y, x) θεωρείται το αντίστροφο του (x, y). Ένας κατευθυνόμενος γράφος D ονομάζεται συμμετρικός για κάθε τόξο στο D υπάρχει επίσης στο D το αντίστροφό του.

Ένας κατευθυνόμενος γράφος ονομάζεται αδύναμα συνδεδεμένος (weakly connected) αν αντικαθιστώντας όλες από τις κατευθυνόμενες σχέσεις με μη-κατευθυνόμενες σχέσεις παράγει έναν συνδεδεμένο μη-κατευθυνόμενο γράφο. Ένας κατευθυνόμενος γράφος ονομάζεται ισχυρά συνδεδεμένος όταν περιέχει ένα κατευθυνόμενο μονοπάτι από το u στο v και ένα κατευθυνόμενο μονοπάτι από το v στο u για κάθε ζευγάρι κόμβων u, v.

3. Υπάρχει και η περίπτωση του μεικτού γράφου G ο οποίος είναι ένας γράφος όπου ορισμένες σχέσεις μπορούν να είναι κατευθυνόμενες και μερικές όχι. Συμβολίζεται με μια διατεταγμένη τριάδα G = (V, E, A) στην οποία V, E, και A ορίζονται σύμφωνα με τα παραπάνω.

4. Ένας πολύ-γράφος (multi-graph) χρησιμοποιείται για να απεικονίσει κοινωνικά δίκτυα πολλαπλών συνδέσεων ανάμεσα σε δύο κόμβους. Ένας βρόχος (loop) είναι μια σχέση που

(κατευθυνόμενη ή μη) η οποία ξεκινά και τελειώνει στον ίδιο κόμβο. Αυτό μπορεί να επιτρέπεται ή όχι ανάλογα την περίπτωση. Ο όρος "multigraph" γενικά εννοεί ότι υπάρχουν εντός του πολλαπλά ήδη σχέσεων και μερικές φορές επιτρέπονται και οι βρόχοι.

5. Σαν αντίθετο του πολύ-γράφου υπάρχει ο απλός γράφος ο οποίος είναι ένας μη – κατευθυνόμενος γράφος που δεν έχει βρόχους και έχει μοναδικές σχέσεις ανάμεσα στους κόμβους. Σε έναν απλό γράφο οι σχέσεις ορίζουν ένα σύνολο και κάθε σχέση αποτελεί ένα ξεχωριστό ζευγάρι από κόμβους. Σε έναν απλό γράφο με n κόμβους, κάθε κόμβος έχει βαθμό (degree) ίσο ή μικρότερο με το n.

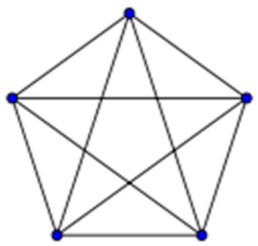
6. Υπάρχουν επιπλέον οι σταθμισμένοι γράφοι αν μια στάθμιση (βάρος) αποδοθεί σε κάθε σχέση. Τέτοιοι γράφοι για παράδειγμα μπορούν να αναπαριστούν κόστη ή/και χρονοδιαγράμματα ανάλογα με το είδος του έργου που αντιπροσωπεύουν. Αναπαριστούν διαβαθμισμένα (Labeled) κοινωνικά δίκτυα και ονομάζονται δίκτυα.

7. Τέλος, σε εξαιρετικές περιπτώσεις μπορούν οι σχέσεις να αφορούν μόνο μία κορυφή (κόμβο) ή καμία κορυφή (signed graphs) και (biased graphs).

Άλλες κατηγορίες γράφων ανάλογα με τις ιδιότητές τους:

1. Κανονικός γράφος: ένας κανονικός γράφος είναι ο γράφος όπου κάθε κόμβος έχει τον ίδιο αριθμό γειτόνων, δηλαδή κάθε κόμβος έχει τον ίδιο βαθμό (degree). Ένας κανονικός γράφος με κόμβους βαθμού k ονομάζεται k-κανονικός γράφος ή κανονικός γράφος βαθμού k.

2. Ολοκληρωμένος γράφος: Οι ολοκληρωμένοι γράφοι έχουν το χαρακτηριστικό ότι υπάρχει σχέση ανάμεσα σε κάθε πιθανό ζευγάρι κόμβων.



3. Πεπερασμένοι και άπειροι γράφοι: Σε έναν πεπερασμένο γράφο G = (V, E) τα σύνολα V και E είναι πεπερασμένα σύνολα. Ένας άπειρος γράφος είναι αυτός με το σύνολο τον σχέσεων

Ε ή/και των κόμβων V να είναι άπειρα. Συνήθως οι γράφοι είναι πεπερασμένοι εκτός αν ορίζεται διαφορετικά εξ αρχής.

Εκτός από τη μέθοδο των γράφων, την οποία πραγματεύεται η παρούσα μεταπτυχιακή διατριβή, τα κοινωνικά δίκτυα μπορούν να αναπαρασταθούν και με έναν άλλο τρόπο μέσω πινάκων (μήτρα γειτνίασης). Η μήτρα γειτνίασης ενός γράφου (κατευθυνόμενου με βρόχους και πολλαπλές συνδέσεις) είναι ένας πίνακας ακεραίων αριθμών με γραμμές και στήλες που αντιστοιχούν στους κόμβους του γράφου όπου κάθε εγγραφή a_{ij} εκτός της διαγωνίου είναι ο αριθμός των συνδέσεων από τον κόμβο i προς τον j. Οι εγγραφές της διαγωνίου a_{ii} είναι ο αριθμός των βρόχων για κάθε κόμβο i. Η μήτρα γειτνίασης ενός γράφου είναι μοναδική.

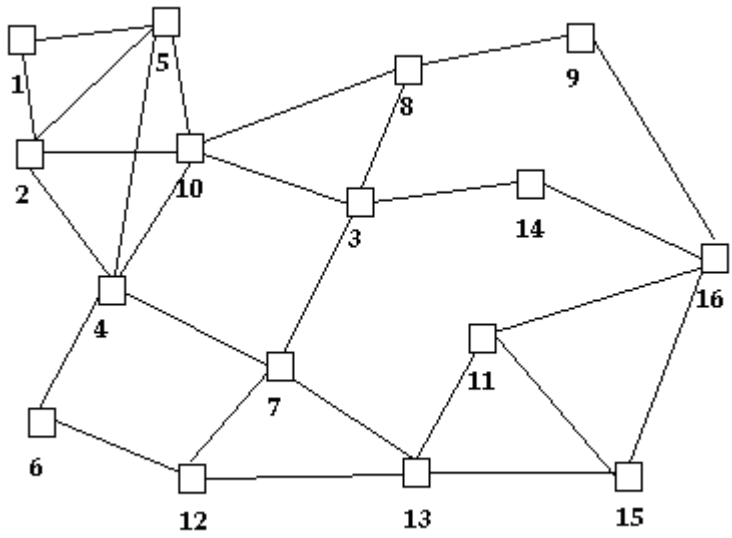
	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**K**	**L**	**M**	**N**
A	0	1	0	1	0	0	0	0	0	0	0	0
B	1	0	1	0	1	1	0	0	0	0	0	0
C	0	1	0	0	0	0	1	0	0	1	0	0
D	1	0	0	0	0	0	0	0	0	0	0	0
E	0	1	0	0	0	1	0	0	0	0	0	0
F	0	1	0	0	1	0	0	0	0	0	0	0
G	0	0	1	0	0	0	0	1	1	0	0	0
H	0	0	0	0	0	0	1	0	0	0	0	0
K	0	0	0	0	0	0	1	0	0	0	0	0
L	0	0	1	0	0	0	0	0	0	0	1	1
M	0	0	0	0	0	0	0	0	0	1	0	0
N	0	0	0	0	0	0	0	0	0	1	0	0

Πίνακας 3.5: Παράδειγμα Πίνακα γειτνίασης

Στην παρούσα μεταπτυχιακή διατριβή δεν θα αναπτυχθεί η συγκεκριμένη μέθοδος παρά θα περιοριστεί στη θεωρία της απεικόνισης των δικτύων μέσω γράφων.

3.2 Μετρικές για γράφους

Προκειμένου να αναλυθεί η βασική δομή ενός γράφου, υπάρχουν ορισμένες βασικές μετρικές ανάλυσης των γράφων των κοινωνικών δικτύων. Χρησιμοποιώντας το παράδειγμα του επόμενου σχήματος θα τις παρουσιάσουμε στη συνέχεια (Ζήνων Ζένιου, [38]).



Σχήμα 3.6: Παράδειγμα μη κατευθυνόμενου γράφου G(16, 27)

3.2.1 Πυκνότητα (Density)

Η πυκνότητα (density) αποτελεί ένα βασικό μέτρο για την κατανόηση της δομής ενός κοινωνικού γράφου. Είναι ο αριθμός των σχέσεων που υπάρχουν σε ένα κοινωνικό γράφο αναφορικά με τον μέγιστο αριθμό των πιθανών σχέσεων που θα μπορούσαν να υπάρξουν. Τυπικά, η πυκνότητα ενός γράφου G {V, E}:

$$D_G = \frac{2 * E}{V(V-1)}$$

Μεγάλη πυκνότητα σημαίνει και μεγάλη συνοχή στον κοινωνικό γράφο (σημαντικός αριθμός σχέσεων). Στο παράδειγμα μας $D_G = \frac{2 * 27}{16(16-1)} = 0.23$

3.2.2 Βαθμός ή Degree (Indegree & outdegree, inout degree)

Ο βαθμός ή degree ενός κόμβου ορίζεται ως ο αριθμός των σχέσεων του με άλλους κόμβους, με άλλα λόγια, ο αριθμός των συνδέσεων που καταλήγουν ή ξεκινούν από αυτόν. Τυπικά, σε έναν γράφο η πυκνότητα ενός γράφου G {V, E}, έστω ότι Nv_i είναι το σύνολο των γειτόνων του

κόμβου v_i, τότε ο βαθμός του είναι: Deg v_i = Nv_i. Μεγάλος βαθμός στους κόμβους υποδεικνύει ένα συνεκτικό δίκτυο καθώς σημαίνει ότι οι κόμβοι έχουν μεγάλο αριθμό σχέσεων και συνεπώς μεγάλη δικτύωση. Ωστόσο μόνο με τον μέσο βαθμό (average degree) ενός γράφου μπορούμε να πάρουμε μια εικόνα της συνεκτικότητας του κοινωνικού γράφου.

Στο δικό μας παράδειγμα Deg v_5 = 3, ενώ ό μέσος βαθμός (degree) του γράφου είναι:

Κόμβοι Vi	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**Μέσος βαθμός (average degree)**
Βαθμός (degree)	**2**	**4**	**4**	**5**	**4**	**2**	**4**	**3**	**2**	**5**	**3**	**3**	**4**	**2**	**3**	**4**	**3.4**

Πίνακας 3.7: Υπολογισμός του βαθμού (degree) των κόμβων του παραδείγματος του σχήματος 3.6

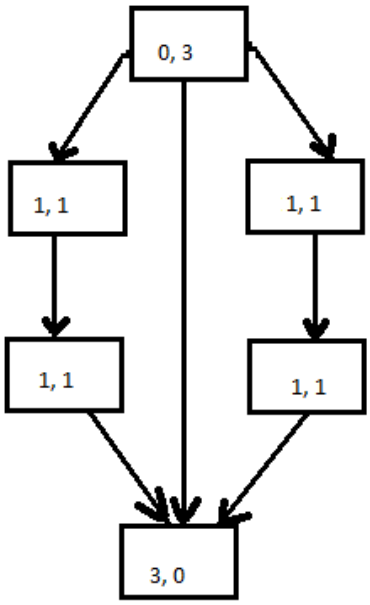
Πέρα από το μέσο βαθμό των κόμβων ενός γράφου (average degree), ορίζεται και η κατανομή των βαθμών (degree distribution). Καθώς μέσα σε ένα κοινωνικό γράφο δεν έχουν όλοι οι κόμβοι τον ίδιο βαθμό, η κατανομή των βαθμών (degree distribution) ως μέτρο της διασποράς των βαθμών των κόμβων ορίζεται ως το ποσοστό των κόμβων του δικτύου με degree k.

Οπότε η κατανομή του εύρους του βαθμού Deg=k ενός κοινωνικού δικτύου είναι το ποσοστό των κόμβων με τον βαθμό αυτό. Στο δικό μας παράδειγμα:

Deg = k	**n (Συχνότητα εμφάνισης βαθμού k)**	**Degree Distribution %**
2	**4**	**25,0**
3	**4**	**25,0**
4	**6**	**37,5**
5	**2**	**12,5**
Σύνολο κόμβων	**16**	**100,0**

Πίνακας 3.8: Υπολογισμός της κατανομής των βαθμών (degree distribution) των κόμβων του παραδείγματος του σχήματος 3.6

Για τον υπολογισμό της δομής των κατευθυνόμενων γράφων χρησιμοποιούνται γενικευμένες έννοιες των μετρικών των μη κατευθυνόμενων γράφων όπως του συντελεστή ομαδοποίησης (cluster coefficient). Βασικά μέτρα για την κατανόηση της δομής των μη κατευθυνόμενων γράφων είναι τα Indegree, Outdegree, Inout degree.



Σχήμα 3.9: Παράδειγμα κατευθυνόμενου γράφου με τα μέτρα ως ταμπέλες στους κόμβους του (indegree, outdegree)

Για έναν κόμβο, ο αριθμός των συνδέσεων που καταλήγουν σε αυτόν ονομάζεται indegree και ο αριθμός των συνδέσεων που φεύγουν από αυτόν ονομάζεται outdegree.

Το μέτρο indegree ορίζεται ως $\deg^-(v)$ και το outdegree ως $\deg^+(v)$. Ένας κόμβος με $\deg^-(v) = 0$ ονομάζεται πηγή (source), καθώς είναι μόνο πηγή κάθε μίας από τις υπάρχουσες συνδέσεις. Ομοίως ένας κόμβος με $\deg^+(v) = 0$ ονομάζεται δεξαμενή (sink) καθώς είναι μόνο αποδέκτης συνδέσεων. Αν για κάθε κόμβο v ∈ V, $\deg^+(v) = \deg^-(v)$, ο γράφος ονομάζεται ισορροπημένος (balanced digraph).

Ο inout degree για έναν κατευθυνόμενο γράφο είναι το άθροισμα των συνδέσεων που «λαμβάνει» ή «διώχνει» ένας κόμβος:

$$\sum_{v \in V} \deg^+(v) = \sum_{v \in V} \deg^-(v) = |A|.$$

3.2.3 Διάμετρος (Diameter) & Αποτελεσματική Διάμετρος (Effective Diameter)

Η διάμετρος (Diameter) ενός κοινωνικού γράφου είναι η μέγιστη απόσταση ανάμεσα σε κάθε ζευγάρι κόμβων, όπου ως απόσταση ορίζεται το μέγεθος της συντομότερης διαδρομής μεταξύ των κόμβων. Η διάμετρος ως μέτρο της δομής ενός κοινωνικού γράφου δίνει σημαντικές πληροφορίες για τη λειτουργία του, όπως πόσο ισχυρά συνδεδεμένος είναι ο γράφος και με ποια ταχύτητα φτάνει μια πληροφορία από τον ένα κόμβο στον άλλο. Όταν η διάμετρος είναι μικρή, αυτό σημαίνει ότι η προσβασιμότητα ανάμεσα στους κόμβους είναι άμεση.

Στο παράδειγμά μας (Σχήμα 3.6) η διάμετρος του γράφου μας είναι 6, με το μεγαλύτερο κοντινό μονοπάτι να είναι από τον κόμβο 1 στο κόμβο 15.

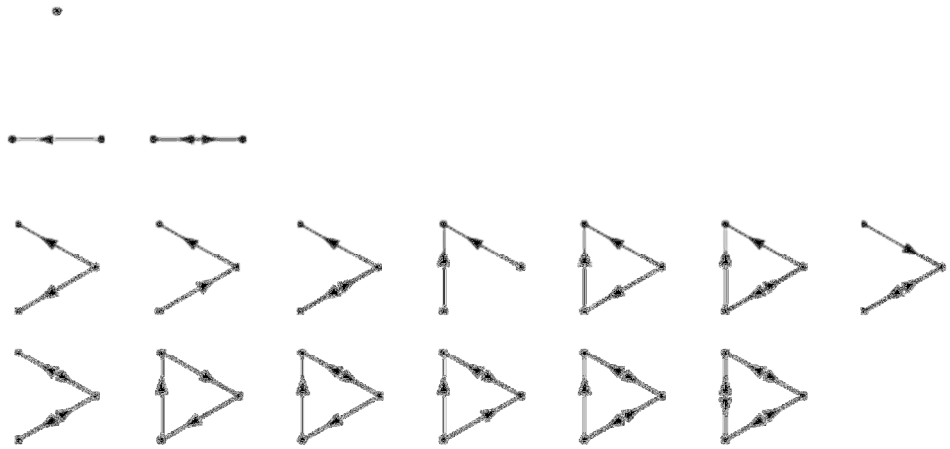
Καθώς όμως ένας μικρός αριθμός κόμβων είναι δυνατόν να σχηματίσει μια "αλυσίδα" εκτός του κυρίως όγκου του δικτύου, με αποτέλεσμα να δίνει λάθος τιμή στην διάμετρος, χρησιμοποιείται μια εξομαλυμένη μορφή της διαμέτρου, η «αποτελεσματική διάμετρος» (effective diameter). Ως «αποτελεσματική διάμετρος» ορίζεται η ελάχιστη απόσταση με την οποία μπορεί να υπάρξει πρόσβαση ανάμεσα στο 90% όλων των συνδεδεμένων ζευγών κόμβων. Στη δική μας περίπτωση είναι πάλι 6, καθώς δεν υπάρχει «αλυσίδα» κόμβων ως παρακλάδι του δικτύου που να διαμορφώνει σχετική «παραφωνία» στη μετρική.

3.2.4 Συνεκτικότητα γράφου (Digraph connectivity)

Η συνεκτικότητα είναι μια βασική έννοια της θεωρίας των γράφων: αναζητά τον ελάχιστο αριθμό στοιχείων (κόμβων ή συνδέσεων) που πρέπει να αφαιρεθούν για να διαλυθεί ένας γράφος. Σχετίζεται με τη θεωρία της ροής των δικτύων (theory of network flow). Η συνεκτικότητα ενός γράφου (connectivity) είναι μέτρο της ευρωστίας ενός δικτύου.

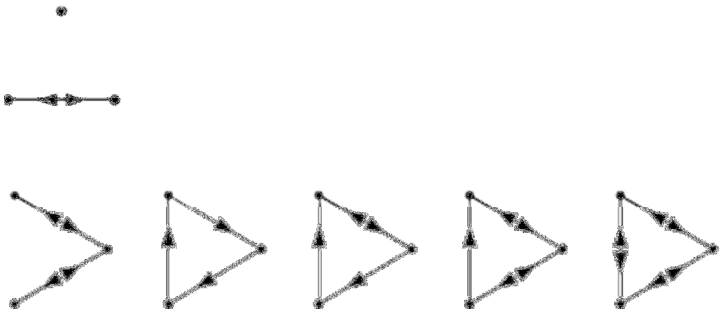
Ένας μη κατευθυνόμενος γράφος G ονομάζεται αδύναμα συνδεδεμένος (weakly connected) όταν ο μη κατευθυνόμενος γράφος που προκύπτει από την αφαίρεση των κατευθύνσεων του γράφου είναι επίσης ένας αδύναμα συνδεδεμένος γράφος. Αντίστοιχα, ένας ισχυρά συνδεδεμένος γράφος (strongly connected) αν περιέχει ένα κατευθυνόμενο μονοπάτι από το u στο v και ένα κατευθυνόμενο μονοπάτι από το v στο u για κάθε ζευγάρι κόμβων (u,v). Τα ισχυρά στοιχεία των γράφων είναι οι μέγιστοι ισχυρά συνδεδεμένοι υπο-γράφοι (the maximal strongly connected subgraphs).

Παράδειγμα αδύναμα συνδεδεμένου γράφου



Σχήμα 3.10[4]: Παράδειγμα αδύναμα συνδεδεμένου γράφου

Ένας κατευθυνόμενος γράφος στον οποίο είναι δυνατόν να συνδεθεί κάθε κόμβος με οποιονδήποτε άλλον μέσω ενός μονοπατιού, ανεξάρτητα από την κατεύθυνση του μονοπατιού, λέγεται αδύναμα συνδεδεμένος γράφος. Συνεπώς οι κόμβοι σε έναν αδύναμα συνδεδεμένο γράφο πρέπει όλοι να έχουν ή outdegree ή indegree τουλάχιστον.



Σχήμα 3.11: Παράδειγμα ισχυρά συνδεδεμένου γράφου

Ένας κατευθυνόμενος γράφος στον οποίο είναι δυνατόν να φτάσει κάθε κόμβος ξεκινώντας από έναν άλλο κινούμενος πάνω στις κατευθύνσεις τις οποίες δείχνουν οι σχέσεις του λέγεται ισχυρά κατευθυνόμενος. Οι κόμβοι σε έναν ισχυρά συνδεδεμένο γράφο πρέπει να έχουν indegree τουλάχιστον 1.

[4] Πηγή για τα **σχήματα**: Weisstein, Eric W. "Strongly Connected Digraph." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/StronglyConnectedDigraph.html

3.2.5 Συντελεστής Ομαδοποίησης - clustering coefficient

Στη συγκεκριμένη μεταπτυχιακή διατριβή επικεντρώνουμε στις έννοιες της ομαδοποίησης, της γειτονιάς και της ανάλυσής τους, και συνεπώς το μέτρο αυτό είναι ιδιαίτερα σημαντικό.

Ο Συντελεστής Ομαδοποίησης[5] ενός κόμβου είναι η πιθανότητα δυο γειτόνων ενός κόμβου να είναι συνδεδεμένοι μεταξύ τους. Ή διαφορετικά αποτελεί την αναλογία των ζευγαριών των γειτονικών κόμβων που είναι συνδεδεμένοι μεταξύ τους. Είναι πολύ σημαντικός συντελεστής καθώς μετρά την «κοινωνικότητα» ενός γράφου [28].

Έστω ότι G = (V,E) είναι ένας γράφος, που αποτελείται από ένα σύνολο κόμβων V και ένα σύνολο σχέσεων E ανάμεσά τους.

Μια σχέση e_{ij} συνδέει έναν κόμβο i με έναν κόμβο j. Η γειτονιά (neighbourhood) N_i για έναν κόμβο v_i ορίζεται σαν τους άμεσα συσχετιζόμενους γείτονές του ως εξής:

$$N_i = \{v_j : e_{ij} \in E \wedge e_{ji} \in E\}.$$

Ορίζουμε τον βαθμό ή degree k_i σαν τον αριθμό των κόμβων $|N_i|$, στη γειτονιά N_i, ενός κόμβου v_i.

Ο τοπικός συντελεστής ομαδοποίησης C_i για έναν κόμβο v_i δίνεται από την αναλογία των συνδέσεων ανάμεσα στους γείτονες-κόμβους ενός κόμβου διά τον αριθμό των πιθανών συνδέσεων ανάμεσά τους.

Για έναν κατευθυνόμενο γράφο το e_{ij} είναι διαφορετικό από το e_{ji}, και συνεπώς για κάθε γειτονιά N_i υπάρχουν $k_i(k_i - 1)$ συνδέσεις που θα μπορούσαν να δημιουργηθούν ανάμεσα στους γείτονες – κόμβους της γειτονιάς (k_i είναι το σύνολο (in + out) του βαθμού (degree) του κόμβου).

Έτσι, ο τοπικός συντελεστής ομαδοποίησης (Local Clustering Coefficient, LCC) για κατευθυνόμενο γράφο δίνεται ως εξής:

---

[5] http://en.wikipedia.org/wiki/Clustering_coefficient

$$C_i = \frac{|\{e_{jk}\}|}{k_i(k_i-1)} : v_j, v_k \in N_i, e_{jk} \in E.$$

Ένας μη-κατευθυνόμενος γράφος έχει την ιδιότητα $e_{ij} \equiv e_{ji}$. Έτσι, αν ένας κόμβος v_i έχει k_i γείτονες, $\frac{k_i(k_i-1)}{2}$ πιθανές συνδέσεις θα μπορούσαν να υπάρχουν στους κόμβους της γειτονιάς του. ΄

Συνεπώς, ο τοπικός συντελεστής ομαδοποίησης (Local Clustering Coefficient, LCC) για μη-κατευθυνόμενο γράφο δίνεται ως εξής:

$$C_i = \frac{2|\{e_{jk}\}|}{k_i(k_i-1)} : v_j, v_k \in N_i, e_{jk} \in E.$$

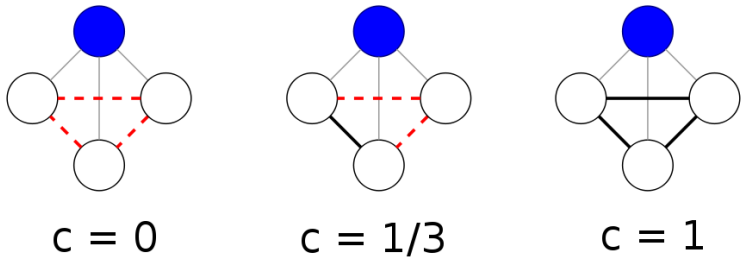
Ο αριθμητής της παραπάνω εξίσωσης είναι ο αριθμός των σχέσεων ανάμεσα στους γείτονες του v_i. Ο παρονομαστής είναι ο αριθμός των πιθανών σχέσεων ανάμεσα στους γείτονες του v_i.

Αυτό το μέτρο κυμαίνεται ανάμεσα στο 0 και το 1. Είναι 1 όταν όλοι οι γείτονες ενός κόμβου είναι συνδεδεμένοι μεταξύ τους και 0 όταν δεν είναι κανείς.

Πιο συγκεκριμένα, στο παράδειγμα του σχήματος 3.12, ο τοπικός συντελεστής ομαδοποίησης του μπλε κόμβου υπολογίζεται σαν το ποσοστό των συνδέσεων ανάμεσα στους γείτονές του σε σχέση όλες με τις πιθανές συνδέσεις. Στο σχήμα 3.12 ο μπλε κόμβος έχει τρεις γείτονες που μπορούν να έχουν το μέγιστο 3 συνδέσεις μεταξύ τους. Στην πρώτη εικόνα και οι τρεις πιθανές συνδέσεις φαίνονται (με μαύρο χρώμα, δίνοντας έναν τοπικό συντελεστή ομαδοποίησης = 1.

Στη μέση, μόνο μια σύνδεση υπάρχει και οι υπόλοιπες (με διακεκομμένη κόκκινη γραμμή) λείπουν. Σε αυτήν την περίπτωση ο τοπικός συντελεστής ομαδοποίησης είναι = 1/3.

Τέλος όταν καμία πιθανή σύνδεση δεν υλοποιείται, ο τοπικός συντελεστής ομαδοποίησης = 0.



Σχήμα 3.12: Παράδειγμα τοπικού συντελεστή ομαδοποίησης σε μη κατευθυνόμενου γράφου G(4, 6)

Ο τοπικός συντελεστής ομαδοποίησης ενός κόμβου σε έναν γράφο ορίζει πόσο κοντά είναι οι γείτονές του από το να αποτελούν μια κλίκα (δηλαδή έναν ολοκληρωμένο γράφο). Οι Duncan J. Watts και Steven Strogatz [07] εισήγαγαν το μέτρο το 1998 για να καθορίσουν αν ένας γράφος αποτελεί ένα «δίκτυο μικρόκοσμου» (small-world network[6]).

Ο συνολικός συντελεστής ομαδοποίησης (Global Cluster Coefficient, GCC) για το σύνολο του γράφου ορίζεται ως εξής:

Έστω $\lambda_G(v)$ ο αριθμός των τριγώνων στα οποία ανήκει ο κόμβος $v \in V(G)$ για έναν μη-κατευθυνόμενο γράφο G. Συνεπώς $\lambda_G(v)$ είναι ο αριθμός των υπο-γράφων (sub-graphs) του G με 3 σχέσεις και 3 κόμβους, ένας από τους οποίους είναι ο v. Έστω $\tau_G(v)$ ο αριθμός των τριάδων στις οποίες ανήκει ο κόμβος $v \in G$. Συνεπώς, $\tau_G(v)$ είναι ο αριθμός των πιθανών υπο-γράφων (sub-graphs) όχι απαραίτητα ολοκληρωμένων.

Τότε μπορούμε επίσης να ορίσουμε τον συντελεστή ομαδοποίησης (Ci) ως

$$C_i = \frac{\lambda_G(v)}{\tau_G(v)}.$$

[6] Ένα «δίκτυο μικρόκοσμου» (small-world network) είναι ένας τύπος γράφου στον οποίο οι περισσότεροι κόμβοι δεν είναι γείτονες μεταξύ τους αλλά μπορεί ο ένας να φτάσει τον άλλο με μικρό αριθμό βημάτων. Επίσης είναι γνωστό ως "six degrees of separation" εφόσον, στο κοινωνικό δίκτυο κάθε άτομο για να συνδεθεί με κάποιο άλλο δίκτυο από περίπου έξι συνδέσεις. Ο «μικρόκοσμος» (small-world network) τείνει να έχει κλίκες δηλαδή υπο-δίκτυα που έχουν συνδέσεις σχεδόν ανάμεσα σε κάθε δύο κόμβους μεταξύ τους.

Αυτό συνεπάγεται υψηλό συντελεστή ομαδοποίησης: (βλ. http://en.wikipedia.org/wiki/Small-world_network και http://mathworld.wolfram.com/SmallWorldNetwork.html)

Είναι απλό να δείξουμε ότι οι προηγούμενοι ορισμοί ταυτίζονται καθώς:

$$\tau_G(v) = C(k_i, 2) = \frac{1}{2}k_i(k_i - 1).$$

Ο GCC βασίζεται σε τριάδες (triplets) από κόμβους. Μια τριάδα αποτελείται από τρεις κόμβους που συνδέονται είτε με ανά δύο (ανοικτή τριάδα) είτα με ανά τρεις (κλειστή τριάδα) μη κατευθυνόμενες συνδέσεις. Ένα τρίγωνο αποτελείται από τρεις κλειστές τριάδες, κάθε μια να κατευθύνεται σε κάθε έναν από τους κόμβους. Ο GCC είναι ο αριθμός των κλειστών τριάδων (ή 3 x τρίγωνα) διά το συνολικό αριθμό των τριάδων (ανοικτών και κλειστών). Η πρώτη προσπάθεια να υπολογιστεί έγινε από τους Luce and Perry (1949). Αυτό το μέτρο δίνει μια ένδειξη της κατηγοριοποίησης του συνολικού δικτύου (global), και μπορεί να εφαρμοστεί και σε μη-κατευθυνόμενα και σε κατευθυνόμενα δίκτυα (συχνά καλείται transitivity, βλ. Wasserman and Faust, σ. 243 [29]).

Μπορεί να υπολογιστεί ως εξής:

$$C_i = \frac{3 * αριθμός\_των\_τριγώνων}{αρ.συνδεδεμένων\_τριάδων\_των\_κόμβων}$$

Μια γενίκευση του δείκτη στα σταθμισμένα δίκτυα προτάθηκε από τους Opsahl and Panzarasa [33], και μια επέκταση και στα δυαδικά δίκτυα (binary and weighted) από τον Opsahl [32].

Ο συντελεστής ομαδοποίησης για το σύνολο του δικτύου δόθηκε από τους Watts and Strogatz [07] σαν τον μέσο όρο όλων των τοπικών συντελεστών ομαδοποίησης όλων των κόμβων n :

$$\bar{C} = \frac{1}{n}\sum_{i=1}^{n} C_i.$$

Στο δικό μας το παράδειγμα οι διάφοροι συντελεστές ομαδοποίησης για τους 16 κόμβους του μη-κατευθυνόμενου γράφου του σχήματος 3.6, παρουσιάζονται στον παρακάτω πίνακα:

Κόμβοι Vi	**Βαθμός =k (degree)**	**Μέγιστος αριθμός πιθανών σχέσεων k *(k-1)**	**Αριθμός συνδέσεων (n) ανάμεσα στους γείτονες του Vi**	**Συντελεστής Ομαδοποίησης cc (Vi)=2*n/(k*(k-1))**
1	2	2	1	1.00
2	4	12	4	0.67
3	4	12	1	0.17
4	5	20	2	0.20
5	4	12	4	0.67
6	2	2	0	0.00
7	4	12	1	0.17
8	3	6	1	0.33
9	2	2	0	0.00
10	5	20	4	0.40
11	3	6	2	0.67
12	3	6	1	0.33
13	4	12	2	0.33
14	2	2	0	0.00
15	3	6	2	0.67
16	4	12	1	0.17
Μέσος βαθμός (average degree)	**3.4**		**Μέσος Συντελεστής Ομαδοποίησης (Average Cluster Coefficient)**	**0.36**
Σύνολο		**144**	**Συνολικός Συντελεστής Ομαδοποίησης (Global Cluster Coefficient)**	**3*8/72=0,33**

Πίνακας 3.13: Συντελεστές ομαδοποίησης

3.2.6 Betweenness Centrality

Το μέτρο Betweenness Centrality αποτελεί ένα μέτρο κεντρικότητας ενός κόμβου μέσα σε ένα γράφο. Το Betweenness Centrality ενός κόμβου αποδίδει την ένταση του ελέγχου που ασκεί ο κόμβος ως ενδιάμεσος των άλλων κόμβων μέσα στο συγκεκριμένο κοινωνικό δίκτυο. Πρωτοπαρουσιάστηκε από τον Linton Freeman [11] για να ποσοτικοποιηθεί ο έλεγχος ενός ατόμου στην ροή της επικοινωνίας ανάμεσα σε άλλα άτομα σε ένα κοινωνικό δίκτυο. Όσο

συχνότερα ένας κόμβος βρίσκεται σε ενδιάμεση θέση στα κοντινότερα μονοπάτια ανάμεσα σε τυχαίους κόμβους τόσο πιο κεντρικά είναι μέσα στο κοινωνικό δίκτυο. Έχει δηλαδή την δύναμη του ελέγχου των συνδέσεων και αυτό συλλαμβάνει το Betweenness Centrality.

Η Betweenness Centrality ενός κόμβου v σε ένα γράφο $G := (V, E)$ με V κόμβους και E σχέσεις υπολογίζεται ως εξής:

1. Για κάθε ζευγάρι κόμβων (s,t), υπολογίζονται τα κοντινότερα μονοπάτια μεταξύ τους.

2. Για κάθε ζευγάρι κόμβων (s,t), αποφασίζεται το κλάσμα του κοντινότερου μονοπατιού που περνά διαμέσου του κόμβου που ελέγχουμε (εδώ του κόμβου v).

3. Αθροίζονται όλα τα κλάσματα που περνούν από το (s,t).

Πιο τυπικά:

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Όπου σ_{st} είναι ο συνολικός αριθμός των κοντινότερων μονοπατιών από τον κόμβο s στον κόμβο t και $\sigma_{st}(v)$ είναι ο συνολικός αριθμός των μονοπατιών που περνάνε διαμέσου του v. The Betweenness Centrality μπορεί να κανονικοποιηθεί (normalized) διαιρώντας με τον αριθμό των ζευγαριών των κόμβων που δεν περιλαμβάνουν τον v, το οποίο είναι για τους κατευθυνόμενους γράφους: $(n-1)(n-2)$ και για τους μη-κατευθυνόμενους $(n-1)(n-2)/2$. Για παράδειγμα σε έναν μη – κατευθυνόμενο γράφο, ο κεντρικός κόμβος (ο οποίος περιέχεται σε οποιοδήποτε κοντινό μονοπάτι) θα είχε μια betweenness $(n-1)(n-2)/2$ (1, αν το κανονικοποιούσαμε) ενώ τα «φύλα» (που δεν περιέχονται σε κοντινότερα μονοπάτια) θα είχαν betweenness=0.

3.2.7 Closeness Centrality

Στους γράφους υπάρχει η μέτρηση απόστασης ανάμεσα σε όλα τα ζευγάρια των κόμβων που αποτελεί το κοντινότερο μονοπάτι. Η απόσταση ενός κόμβου s ορίζεται σαν το σύνολο όλων των αποστάσεων του από όλους τους άλλους κόμβους και η εγγύτητά του ορίζεται ως το

αντίστροφο από την απόσταση. Έτσι, ένας κόμβος είναι πιο κεντρικός όσο μικρότερη είναι η συνολική απόσταση του από όλους τους άλλους κόμβους. Η εγγύτητα (closeness) μπορεί να θεωρηθεί ως μέτρο του πόσο θα χρειαστεί για να διασπαρθεί η πληροφορία από το s σε όλους τους άλλους κόμβους με σειριακή αλληλουχία.

Εξ' ορισμού από τη θεωρία των γράφων η κλασσική closeness centrality όλων των κόμβων ενός μη συνδεδεμένου γράφου θα ήταν 0. Σε μια εργασία των Dangalchev [08] που σχετίζεται με το πόσο ευάλωτα είναι τα δίκτυα ο ορισμός της closeness διαμορφώθηκε έτσι ώστε να μπορεί να υπολογιστεί πιο εύκολα αλλά και να εφαρμοστεί σε γράφους χωρίς συνδεσιμότητα:

$$C_C(v) = \sum_{t \in V \setminus v} 2^{-d_G(v,t)}.$$

Το closeness centrality ορίζεται ως ο κανονικοποιημένος αριθμός "βημάτων" που χρειάζονται για ένα οποιοδήποτε κόμβο για να έχει πρόσβαση σε όλους τους υπόλοιπους κόμβους μέσα στο κοινωνικό δίκτυο.

Από άποψη υπολογισμού και η betweenness και η closeness centrality όλων των κόμβων απαιτούν τον υπολογισμό των κοντινότερων μονοπατιών. Για τον υπολογισμό τους μπορούν να χρησιμοποιηθούν ανάλογα με τον τύπο του δικτύου οι αλγόριθμοι Floyd–Warshall, Johnson's και Brandes'.

3.2.8 Network Community Profile Plot

Η Network community profile plot χαρακτηρίζει την καλύτερη δυνατή κοινότητα σε ένα εύρος κλίμακας κοινοτήτων. Χρησιμοποιούμε την ευρέως διαδεδομένη έννοια του μέτρου που ονομάζεται conductance, το οποίο μπορεί να θεωρηθεί ως ο αριθμός των σχέσεων που απευθύνονται προς το εξωτερικό της κοινότητας διά του αριθμού των σχέσεων προς το εσωτερικό της κοινότητας. Όσο μικρότερη είναι η τιμή conductance, τόσο καλύτερη θεωρείται η κοινότητα. Βάση αυτού του μέτρου, μια καλή κοινότητα είναι αυτή που περιέχει πολλές σχέσεις μεταξύ των κόμβων της και συνδέεται με το υπόλοιπο δίκτυο μέσω όσο το δυνατόν λιγότερων σχέσεων.

Σε αντίθεση με τα μικρά δίκτυα που συχνά εμφανίζουν αμφίβολη κοινωνική δομή, τα μεγάλα δίκτυα έχουν έναν ένθετο πυρήνα και περιφερειακή δομή. Δηλαδή αποτελούνται από έναν

σχετικά μεγάλο και διασυνδεδεμένο πυρήνα και έναν μεγάλο αριθμό μικρών αλλά πολύ καλά συνδεδεμένων κοινοτήτων με μικρή σύνδεση με τον πυρήνα τους. Οι κοινότητας στα δίκτυα τείνουν να υπάρχουν μόνο σε μικρές κλίμακες (μέχρι 100 κόμβους) ενώ σε μεγαλύτερης κλίμακας αριθμούς κόμβων οι δικτυώσεις χάνουν τον χαρακτήρα της κοινότητας.

3.2.9 Modularity

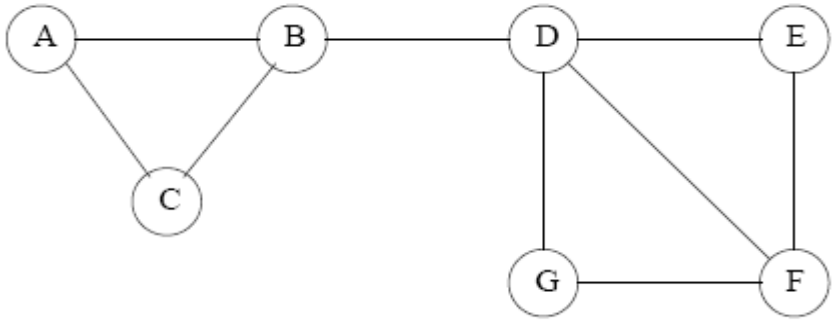
Η Modularity μετρά τη δομή των δικτύων καθώς σχεδιάστηκε για να μετρά το διαχωρισμό ενός δικτύου σε κοινότητες. Τα δίκτυα με μεγάλη modularity έχουν έντονη συσχέτιση ανάμεσα στους κόμβους των κοινοτήτων και μικρή συσχέτιση ανάμεσα σε κόμβους διαφορετικών κοινοτήτων. Η modularity συχνά χρησιμοποιείται σε μεθόδους βελτιστοποίησης για την ανίχνευση της δομής των κοινοτήτων στο δίκτυο. Ωστόσο, έχει έναν περιορισμό στη μικρή ανάλυση, και συνεπώς δεν είναι ικανή να ανιχνεύσει με επιτυχία μικρές κοινότητες.

Το modularity ενός κοινωνικού δικτύου υπολογίζεται ως την αναλογία των συνδέσεων που συμβαίνουν ανάμεσα σε δεδομένες ομάδες μείον τις αναμενόμενες συνδέσεις αν αυτές κατανέμονταν τυχαίο τρόπο. Η τιμή της modularity κυμαίνεται ανάμεσα [−1/2,1]. Είναι θετική αν ο αριθμός των σχέσεων εντός των ομάδων υπερέχει του αριθμού που αναμένεται στη βάση της τυχαιότητας. Για μια δεδομένη δομή δικτύου σε κοινότητες, η modularity αντικατοπτρίζει την συγκέντρωση των κόμβων εντός των κοινοτήτων σε σύγκριση με την τυχαία κατανομή των συνδέσεων ανάμεσα στους κόμβους ανεξάρτητα από τις κοινότητες.

Στην συνέχεια θα εξετάσουμε τους σημαντικότερους αλγόριθμους που χρησιμοποιούνται για τον υπολογισμό των παραμέτρων – μετρικών των κοινωνικών γράφων.

3.3 Αλγόριθμοι Υπολογισμού Μετρικών για την συσταδοποίηση (clustering) των γράφων

Όσον αφορά τη μεθοδολογία που αφορά την κατηγοριοποίηση στους γράφους φαίνεται ότι η βασική μεθοδολογία ταξινόμησης (clustering methods) δεν επαρκεί. Υπάρχουν δύο προσεγγίσεις για την ταξινόμηση συνόλων: η ιεραρχική (agglomerative) και η σημειακού βάρους (point-assignment). Αυτές οι παραδοσιακές μέθοδοι ταξινόμησης και ομαδοποίησης δεν εφαρμόζονται στα κοινωνικά δίκτυα: η μέτρηση της απόστασης αλλά και οι προαναφερόμενες μέθοδοι δεν εφαρμόζονται σωστά σε έναν κοινωνικό γράφο. Αυτό οφείλεται στην λογική δόμησης του κοινωνικού δικτύου. Για παράδειγμα υπάρχει δυσκολία εξ ’αρχής στον ορισμό της έννοιας της απόστασης (αν οι σχέσεις ανάμεσα στους κόμβους δεν είναι διαβαθμισμένες καταφεύγει κανείς στην binary λύση του 0 και 1) και συνεπώς και στις εφαρμογές αυτής.



Σχήμα 3.14: Παράδειγμα μικρού κοινωνικού δικτύου

Έτσι, από την εφαρμογή των κλασσικών μεθόδων που χρησιμοποιούν την απόσταση ως μέσο για την συσταδοποίηση (βλ. Σχήμα 3.14) προκύπτει ο κίνδυνος να θεωρήσουμε ως ομάδα κόμβους που απλώς συνδέονται όπως ο B και ο D οι οποίοι ανήκουν σαφώς σε διαφορετικές ομάδες. Έτσι αναπτύχθηκαν διάφορες εξειδικευμένες τεχνικές για τον εντοπισμό των κοινοτήτων στα κοινωνικά δίκτυα.

3.3.1 Σειριακοί αλγόριθμοι

Με την έννοια σειριακός αλγόριθμός εννοούμε όλους αυτούς τους αλγόριθμους οι οποίοι ακολουθούν σειριακή εκτέλεση σε αντίθεση με τους map reduce αλγόριθμους οι οποίοι μπορούν να τρέξουν παράλληλα.

Betweenness

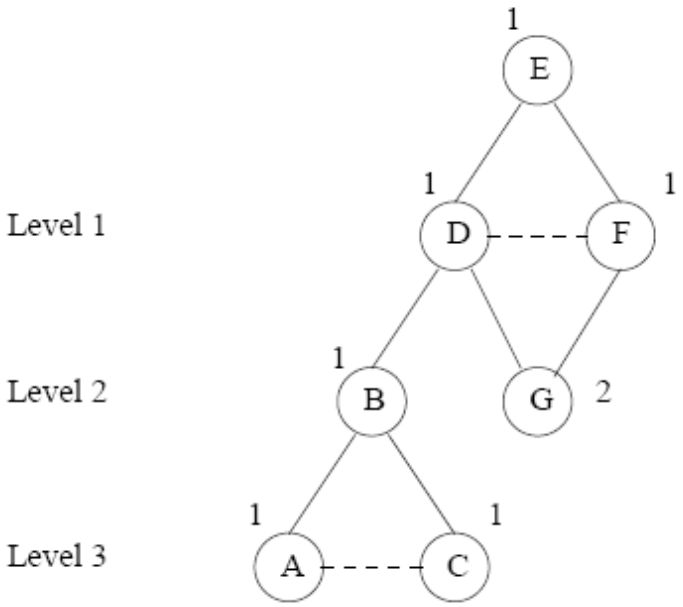
Ο αλγόριθμος αυτός βασίζεται στον εντοπισμό των συνδέσεων που είναι λιγότερο πιθανόν να βρίσκονται μέσα στην κοινότητα. Ορίζουμε ως betweenness μιας σχέσης (a,b) σαν τον αριθμό των ζευγαριών των κόμβων x και y έτσι ώστε η σχέση (a,b) να βρίσκεται ανάμεσα στο κοντινότερο μονοπάτι ανάμεσα στο x και το y. Ειδικότερα, εφόσον μπορούν να υπάρχουν δεκάδες κοντινότερα μονοπάτια ανάμεσα στο x και το y η σχέση (a,b) βρίσκεται χρεώνεται με το ποσοστό αυτών των κοντινότερων μονοπατιών που περιλαμβάνουν τη σχέση (a,b). Η υψηλή τιμή για το μέτρο αυτό υποδεικνύει ότι η σχέση (a,b) υπάρχει ανάμεσα σε δύο διαφορετικές κοινότητες, δηλαδή ότι το a και το b δεν ανήκουν στην ίδια κοινότητα.

Στο σχήμα 3.14 του παραδείγματος, η σχέση (B,D) έχει την μεγαλύτερη betweenness. Βρίσκεται στο κοντινότερο μονοπάτι από τα A, B, C, προς D, E, F, και G. Η betwenness είναι συνεπώς: 3x4=12. Αντιθέτως η σχέση (D,F) έχει μόνο 4 κοντινότερα μονοπάτια: από τα A, B, C, D προς το F.

Αλγόριθμος Girvan-Newman

Οι Girvan και Newman [22] πρότειναν αλγόριθμο που δεν χρειαζόταν να υπάρχει προϋπάρχουσα γνώση για τις κοινότητες μέσα σε δίκτυο. Ο αλγόριθμος υπολογίζει την πιο σημαντική ακμή μέσα στο δίκτυο και την αφαιρεί. Αυτό επαναλαμβάνεται μέχρι ο γράφος να χωριστεί σε κομμάτια, και αυτά τα κομμάτια θεωρούνται κοινότητες. Ο υπολογισμός της πιο σημαντικής ακμής γίνεται με την μετρική betweenness η οποία τείνει να δίνει ψηλή τιμή σε ακμές που είναι γέφυρες μεταξύ κοινοτήτων. Επειδή σε δίκτυα μεγάλης κλίμακας ο αλγόριθμος είναι αργός, προτάθηκε ένας εναλλακτικός αλλά πιο γρήγορος αλγόριθμος με βάση το modularity. Ο αλγόριθμος αυτός αρχίζει με τον κάθε κόμβο να ανήκει σε μια ξεχωριστή κοινότητα και στη συνέχεια ενώνει ζεύγη κοινοτήτων, διαλέγοντας κάθε φορά το ζεύγος που θα δώσει την μεγαλύτερη αύξηση (ή μείωση) στο modularity. Νεώτερη προσέγγιση [2] λέει ότι πρότειναν μια παραλλαγή του αλγόριθμου που είναι πιο βελτιστοποιημένος και αποδοτικός και επιτρέπει την χρησιμοποίηση του σε τεράστιους γράφους με χιλιάδες ακμές.

Για να χρησιμοποιήσουμε την έννοια της betweenness πρέπει να υπολογίσουμε τον αριθμό των κοντινότερων μονοπατιών σε κάθε σχέση. Θα περιγράψουμε τη μέθοδο που χρησιμοποιεί ο αλγόριθμος Girvan – Newman [22] (GN) που επισκέπτεται κάθε κόμβο Χ μία φορά και υπολογίζει τον αριθμό των κοντινότερων μονοπατιών από τον Χ σε κάθε έναν από τους άλλους κόμβους που βρίσκονται σε κάθε μία σχέση. Ο αλγόριθμος ξεκινά εκτελώντας ένα αρχικό ψάξιμο του γράφου (BFS: Breath First Search) ξεκινώντας από τον κόμβο Χ. σημειώσετε ότι το επίπεδο κάθε κόμβου στην BFS παρουσίαση είναι αυτό του κοντινότερου μονοπατιού ανάμεσα στον Χ και σε αυτόν τον κόμβο. Έτσι οι σχέσεις που βρίσκονται σε κόμβους του ίδιου επιπέδου δεν μπορούν ποτέ να είναι κομμάτι του κοντινότερου μονοπατιού. Οι σχέσεις ανάμεσα στα επίπεδα ονομάζονται DAG σχέσεις (DAG: Directed Acyclic Graph). Κάθε DAG σχέση θα είναι τμήμα ενός τουλάχιστον κοντινότερου μονοπατιού από στην πορεία της ρίζας Χ. Αν υπάρχει μια DAG σχέση (Υ,Ζ) όπου το Υ είναι στο επίπεδο πάνω από το Ζ, δηλαδή κοντινότερα στην ρίζα Χ, τότε ονομάζουμε τον Υ γονέα (parent) του Ζ και τον Ζ παιδί (child) του Υ, παρόλο που οι γονείς δεν είναι απαραίτητο να είναι μοναδικοί στην DAG σχέση.



Σχήμα 3.15: BFS παρουσίαση, βήμα 1 του αλγόριθμου Girvan-Newman [22]

Το παραπάνω σχήμα, αποτελεί μια BFS παρουσίαση του σχήματος 3.14, ξεκινώντας από τον κόμβο Ε. Οι συνεχείς γραμμές είναι σχέσεις DAG και οι διακεκομμένες γραμμές συνδέουν τους κόμβους του ίδιου επιπέδου.

Το δεύτερο βήμα του αλγόριθμου GN είναι να τοποθετήσει μια ετικέτα σε κάθε κόμβο με τον αριθμό των κοντινότερων μονοπατιών από τη ρίζα με αριθμό 1. Ξεκινώντας από πάνω προς τα κάτω ονοματίζουμε κάθε κόμβο Υ με το σύνολο των ετικετών των γονέων του.

Παραδείγματος χάριν, στο σχήμα 3.15 έχουμε τις ταμπέλες για κάθε έναν από τους κόμβους. Πρώτα βάζουμε την ταμπέλα για τη ρίζα E = 1. Στο επίπεδο 1 έχουμε κόμβους D και F, οι οποίοι έχουν ως γονέα μόνο τον E και παίρνουν τον αριθμό 1. Οι κόμβοι B και G είναι στο επίπεδο 2. Ο B έχει μόνο τον D σαν γονέα οπότε κι αυτός παίρνει τον αριθμό 1. Όμως ο G έχει γονείς τον D και τον F και ο αριθμός του είναι το σύνολο των γονέων του δηλαδή 2. Τελικά στο επίπεδο 3, οι κόμβοι A και C ο καθένας έχει μόνο γονέα τον B κι έτσι οι αριθμοί του είναι οι ίδιοι με τον B ο οποίος είναι 1.

Το τρίτο και τελευταίο βήμα είναι να υπολογίσει για κάθε κόμβο το σύνολο για όλους τους κόμβους Υ την πιθανότητα ότι το κοντινότερο μονοπάτι από τη ρίζα X στην Υ να πηγαίνει διαμέσου αυτής της σχέσης. Αυτός ο υπολογισμός περιλαμβάνει τον υπολογισμό αυτού του αθροίσματος και για τους κόμβους και τις σχέσεις από τη βάση προς τα πάνω. Κάθε κόμβος άλλος από τη ρίζα του δίνεται ένας βαθμός 1 που αντιπροσωπεύει το κοντινότερο μονοπάτι σε αυτόν τον κόμβο. Ο βαθμός αυτός μπορεί να διαιρείται ανάμεσα στους κόμβους και τις σχέσεις τους, εφόσον θα μπορούσαν να υπάρχουν δεκάδες διαφορετικά κοντινότερα μονοπάτια προς τον κόμβο. Οι κανόνες των υπολογισμών ακολουθούν:

1 Κάθε «φύλο» του DAG (κόμβος χωρίς DAG σχέσεις σε κόμβους στα κατώτερα επίπεδα παίρνει τον αριθμό 1.

2 Κάθε κόμβος που δεν είναι «φύλο» παίρνει έναν βαθμό ίσο με το 1 συν το σύνολο των βαθμών από τις σχέσεις DAG από αυτόν τον κόμβο σε ένα επίπεδο παρακάτω.

3 Μια σχέση DAG ανάμεσα στον κόμβο Z από το παραπάνω επίπεδο δίνεται ένα τμήμα του βαθμού του Z ποσοστιαία προς την πιθανότητα ότι το κοντινότερο μονοπάτι από τη ρίζα Z πηγαίνει διαμέσου αυτής της σχέσης. Τυπικά, έστω ότι οι γονείς του Z είναι οι Υ1, Υ2, …, Υκ. Έστω ότι ρj είναι ο αριθμός των κοντινότερων μονοπατιών από τη ρίζα στο Υ1, ο αριθμός είχε υπολογιστεί στο βήμα 2 και εμφανίζεται με τις ταμπέλες στο σχήμα 2. στη συνέχεια ο βαθμός για τη σχέση (Υi,Z) είναι ο βαθμός του Z x ρi / $\sum_{j\ 1}^{k} \rho_j$.

4 Στο τέλος ο βαθμός κάθε DAG σχέσης είναι το τμήματα ων κοντινότερων μονοπατιών από τη ρίζα X προς αυτήν τη σχέση. Οι μη- DAG σχέσεις δεν έχουν κοντινότερα μονοπάτια από τον κόμβο X προς αυτές.

3.3.2 Αλγόριθμοι αναζήτησης τριγώνων

NodeIterator

Για τον υπολογισμό του CC χρησιμοποιούμε την παρακάτω μεθοδολογία: αρχικά θα ταξινομήσουμε τους κόμβους ως εξής (A. Rajaraman & J. Ullman, [02]): Κατ' αρχάς ταξινομούμε τους κόμβους σύμφωνα με το βαθμό τους (degree). Στη συνέχεια, αν ο υ και ο u έχουν τον ίδιο βαθμό, δεδομένου ότι και οι δύο είναι ακέραιοι, τότε τους ταξινομούμε αριθμητικά. Έτσι, ορίζουμε τη σχέση υ $\prec$ u αν και μόνο αν

(i) ο βαθμός του υ είναι μικρότερος από τον βαθμό του u, ή

(ii) ο βαθμός του υ είναι ίσος με τον βαθμό u, και υ < u

Ονομάζουμε τη διαδικασία της δημιουργίας μονοπατιών μήκους 2 ανάμεσα στους γείτονες του κόμβου υ «περιστροφή» ή "pivoting", γύρω από τον κόμβο υ. Ο αλγόριθμος λειτουργεί χρησιμοποιώντας αυτήν την περιστροφή γύρω από κάθε κόμβο και έπειτα ελέγχοντας αν υπάρχει σχέση η οποία να ολοκληρώνει οποιοδήποτε από τα εξαγόμενα μονοπάτια έτσι ώστε να δημιουργηθεί τρίγωνο. Δίνεται στη συνέχεια ο ψευδοκώδικας: Σημειώνουμε ότι κάθε τρίγωνο {u,υ,w} μετριέται έξι φορές στο σύνολο ως εξής: (κάθε φορά ως {u,υ,w}, {u,w,υ}, {υ,u,w}, {υ,w,u}, {w,u,υ}, {w,υ,u}).

Ο χρόνος που τρέχει αυτός ο αλγόριθμος υπολογίζεται ως : O($\sum_{\upsilon \in V} d_{\upsilon}^{2}$), ο οποίος σε γράφους σταθερού βαθμού (degree), όπου du=0(1) για όλα τα υ, αυτός είναι ένας αλγόριθμος γραμμικού χρόνου (linear time algorithm). Έτσι, ακόμη και ένας απλός κόμβος υψηλού βαθμού μπορεί να οδηγήσει σε εκθετικό χρόνο τρεξίματος (running time). Όμως τέτοιου τύπου κόμβοι στην πραγματικότητα υπάρχουν, αυτός ο αλγόριθμος δεν ενδείκνυται για τους υπολογισμούς σε πραγματικά, μαζικά δεδομένα.

Algorithm 1 NodeIterator(V,E)

```
1: T ← 0;
2: for v ∈ V do
3:    for u ∈ Γ(v) do
4:       for w ∈ Γ(v) do
5:          if ((u, w) ∈ E) then
6:             T ← T + 1/2;
7: return T / 3;
```

NodeIterator++

Ο αλγόριθμος NodeIterator++ αποτελεί τη βελτιωμένη έκδοση του αλγόριθμου NodeIterator και έχει ως αποτέλεσμα την ταχύτερη εύρεση των τριγώνων ενός γράφου.

Ο NodeIterator++ μειώνει τον χρόνο εκτέλεσης του παραδοσιακού αλγόριθμου NodeIterator – στον οποίο σημειώνεται ότι κάθε τρίγωνο μετριέται έξι φορές- δύο φορές στον άξονα κάθε κόμβου. Επιπλέον αυτοί οι άξονες γύρω από τους κόμβους με υψηλό βαθμό παράγουν πολύ περισσότερα από δύο μονοπάτια και είναι συνεπώς πολύ περισσότερο χρονοβόροι από ότι οι άξονες γύρω από τους κόμβους με χαμηλό βαθμού.

Για να βελτιώσει λοιπόν τον βασικό αλγόριθμο ο T. Schank [30] πρότεινε ότι ο κόμβος με τον χαμηλότερο βαθμό σε κάθε τρίγωνο θα είναι «υπεύθυνος» για την επιβεβαίωση της μέτρησης του τριγώνου. Με τον τρόπο αυτόν γίνεται ένας επιπλέον έλεγχος στον αλγόριθμο, ο οποίος περιορίζει το σύνολο των μονοπατιών που δημιουργούνται από τους γείτονες του u-κόμβου στα μονοπάτια ανάμεσα στον u και στους κόμβους που έχουν βαθμό μεγαλύτερο από αυτόν του u-κόμβου (τον εαυτό του δηλαδή).

Ο χρόνος εκτέλεσης του συγκεκριμένου αλγόριθμου είναι: O(m3/2).

Algorithm 2 NodeIterator++(V,E)

```
1: T ← 0;
2: for v ∈ V do
3:    for u ∈ Γ(v) and u ≻ v do
4:       for w ∈ Γ(v) and w ≻ u do
5:          if ((u, w) ∈ E) then
6:             T ← T + 1;
7: return T;
```

Επεξήγηση συμβόλων

E	Ακμή
w	Κόμβος
υ	Κορυφή (Vertex)
Γ	Γείτονας
T	Αριθμός τριγώνων
u	Κόμβος

Ο αλγόριθμος δέχεται ως όρισμα τα V & E. Για την εκτέλεση του αλγόριθμου θα πρέπει να προηγηθεί η εύρεση του βαθμού του κάθε κόμβου δηλ. τον αριθμό των ακμών που έχει με κάθε άλλο κόμβο.

Στην συνέχεια «τρέχουμε» τον αλγόριθμο για κάθε κόμβο με στόχο να εντοπίσουμε τα πιθανά τρίγωνα των γειτόνων , η ανάλυση του οποίου δίνεται παρακάτω:

1 Αρχική τιμή τριγώνων = 0

2 Στο πρώτο for: για κάθε κόμβο u που ανήκει στο σύνολο των κορυφών V

3 Στο δεύτερο for: για κάθε u που ανήκει στους γείτονες του υ (Γ(υ)) και ο βαθμός του είναι μεγαλύτερος από αυτόν του v (u>υ)

4 Στο τρίτο for: για κάθε w που ανήκει στους γείτονες του v (Γ(υ)) και ο βαθμός του είναι μεγαλύτερος του από αυτόν του v (w>u)

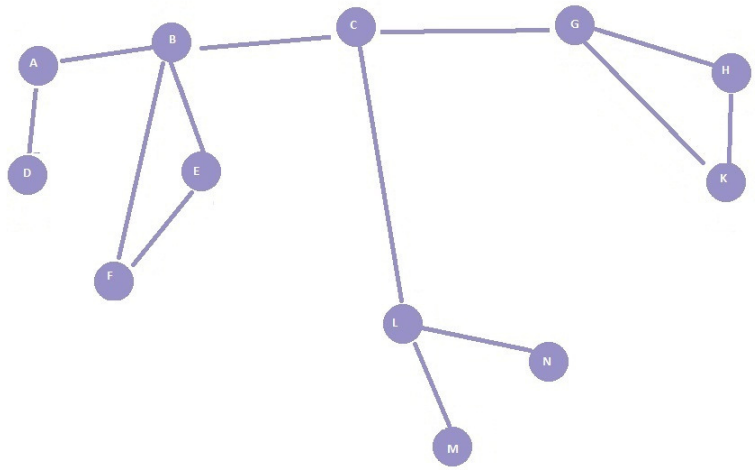
5 Αν οι κόμβοι u και w ανήκουν στο σύνολο των γειτόνων (ανήκει E) τότε

6 Υπάρχει τρίγωνο και δίνεται η αντίστοιχη τιμή

7 Ολοκλήρωση αλγορίθμου και επιστροφή τιμής

Στην συνέχεια θα τρέξουμε τον αλγόριθμο για ένα τμήμα του παρακάτω γράφου, ο οποίος αποτελεί ένα στιγμιότυπο από την γραφική αποτύπωση ενός κοινωνικού δικτύου.

Ο στόχος του αλγορίθμου είναι μετά την εφαρμογή του, να μας δώσει τον αριθμό των τριγώνων που σχηματίζονται, ο οποίος είναι και ο αριθμός των «σχέσεων».



Σχήμα 3.16: Γράφος

Κατά την εφαρμογή του αλγορίθμου για κάθε κόμβο που υπάρχει στον γράφο δίνεται ο βαθμός του.

Συνεπώς:

A B E C F D E F L M N G H K

2 4 2 3 2 1 2 2 3 1 1 3 2 2

Έπειτα, εντοπίζονται οι κόμβοι και οι γείτονες του κάθε ενός και ταξινομούνται με βάση το βαθμό τους.

Στη συνέχεια εφαρμόζεται ο αλγόριθμος για κάθε κόμβο της γειτονίας έως να εντοπιστεί το τρίγωνο μέσω του κόμβου με τον μικρότερο βαθμό.

Ξεκινάμε την εφαρμογή του αλγόριθμου για τον κόμβου Β, οι γείτονες (Γ) του κόμβου Β είναι οι A, E, C, F, άρα (Γ(B) = {A,E,C,F}) και οι οποίοι ταξινομούνται σε σειρά βαθμού:

A E F C

2 2 2 3

Για τον κόμβο Β έχουμε : Γ(Β) = {Α, Ε, F, C }

Όπου Β > C > A > E > F

Για τον κόμβο Ε έχουμε: Γ(Ε) = { Β!!, F!!}

Όπου Β > Ε > F

Για όσο Β: Ε (ισχύει)

Για όσο είναι το F > Β; (δεν ισχύει η σχέση)

Επόμενος κόμβος

Όπου F: E (δεν ισχύει η σχέση)

Επόμενος κόμβος F: {Β!!, Ε!!}

Όπου Β > Ε > F

Για όσο Β: F (ισχύει)

Ε > Β (δεν ισχύει)

Ε > F (ισχύει)

Ε: Β > F (ισχύει), συνεπώς Τ= Τ+1

Ομοίως προχωρούμε και για τους άλλους κόμβους του γράφου.

3.3.3 Αλγόριθμοι αναζήτησης τριγώνων στο Map Reduce

Οι προηγούμενοι αλγόριθμοι απλά υποθέτουν ότι η δομή των δεδομένων των γράφων απλά ταιριάζει στην μνήμη μιας απλής μηχανής. Για μαζικούς γράφους αυτό δεν ισχύει πια, και οι

ερευνητές έχουν στραφεί σε παράλληλους αλγόριθμους για να αντιμετωπίσουν αυτό το πρόβλημα. Παρά το γεγονός ότι οι παράλληλοι αλγόριθμοι επιτρέπουν τον υπολογισμό σε πολύ μεγαλύτερες κλίμακες, δεν αποτελούν πανάκεια και ο σχεδιασμός του κάθε αλγόριθμου έχει μεγάλη σημασία για την επιτυχή του εφαρμογή στον πραγματικό κόσμο. Ειδικότερα, οι απαιτήσεις σε χώρο των αλγορίθμων δεν εξαφανίζονται επειδή απλά είναι περισσότερες μηχανές διαθέσιμες. Για αρκετά μεγάλο n, $O(n^2)$ μέγεθος μνήμης αποτελεί ακόμη και με αυτήν την μέθοδο μια παράλογη απαίτηση για $n \approx 10^8$, ένας χώρος n^2, απαιτεί περίπου 10 petabyte αποθήκευσης.

Ο συντελεστής ομαδοποίησης (clustering coefficient) ενός κόμβου σε ένα κοινωνικό δίκτυο αποτελεί ουσιώδες μέτρο που ποσοτικοποιεί πόσο «δεμένη» είναι η κοινότητα γύρω από τον κόμβο. Ο υπολογισμός του μπορεί να περιοριστεί στη μέτρηση των υπαρχόντων τριγώνων στο συγκεκριμένο κόμβο του δικτύου. Στην περίπτωση που ο γράφος είναι πολύ μεγάλος για να χωράει στη μνήμη, υπάρχει μια όχι και τόσο αυτόματη διαδικασία και προηγούμενοι ερευνητές έδειξαν πώς μπορεί να υπολογιστεί ό συντελεστής ομαδοποίησης σε αυτό το σενάριο. Ένας διαφορετικός δρόμος έρευνας είναι να γίνουν οι υπολογισμοί παράλληλα, διασκορπίζοντάς τους σε πολλές μηχανές. Στα πρόσφατα έτη το MapReduce εμφανίστηκε ως de facto παράδειγμα προγραμματισμού για παράλληλο υπολογισμό σε μαζικά datasets. Στο κεφάλαιο που ακολουθεί παρουσιάζονται οι MapReduce αλγόριθμοι που χρησιμοποιούνται για να μετρηθούν τα τρίγωνα που χρησιμοποιούμε για να υπολογιστούν οι συντελεστές ομαδοποίησης (clustering coefficients).

NodeIterator ++ στο περιβάλλον MapReduce

Ξεκινάμε με τον αλγόριθμο NodeIterator. Ο αλγόριθμος εκτελείται σε δύο γύρους. Χρειάζονται δηλαδή δύο διαδοχικές Map Reduce διαδικασίες, κατά τις οποίες η εκτέλεση του αλγορίθμου, είναι η ακόλουθη:

Βήμα 1: Δημιουργία το πιθανό μήκος δύο μονοπατιών στο γράφο με την «περιστροφή» (pivoting) γύρω από κάθε κόμβο παράλληλα.

Ο σκοπός της εκτέλεσης της πρώτης διαδικασίας, είναι η δημιουργία σε ζεύγη, όλων των πιθανών μονοπατιών μήκους δύο – δηλαδή δευτέρου βαθμού και άνω – στο γράφο, με τον περιστροφικό έλεγχο σε κάθε κόμβο, παράλληλα.

```
1: Map 1: Input: ((u; v); 0)
2:          if v > u then
3:          emit (u; v)
4: Reduce 1: Input (v; S ⊆ Γ(v))
5:          for (u; w) : u; w ∈ S do
6:          emit (v; (u; w))
```

Βήμα 2: Στη δεύτερη αυτή διαδικασία, ελέγχονται όλα τα ζεύγη που παράχθηκαν από την πρώτη διαδικασία, εάν «κλείνουν» με ακμή στο γράφο και κατάλληλα υπολογίζει και τον αριθμό των τριγώνων που προκύπτουν. Σημειώνουμε ότι μαζί με την έξοδο της πρώτης διαδικασίας που λαμβάνεται ως είσοδος, στη δεύτερη διαδικασία μπαίνει στην είσοδο και η αρχική λίστα του γράφου. Όπως θα δούμε, ο αλγόριθμος διαφοροποιείται μεταξύ των δύο τύπων εισόδου, με τη χρήση του ειδικού χαρακτήρα «$», με την προϋπόθεση βέβαια ότι δεν εμφανίζεται αυτός σε κάποιο από τα set εισόδου.

```
7: Map 2:
8:          if Input of type (v; (u; w)) then
9:          emit ((u; w); v)
10:         if Input of type ((u; v); 0) then
11:         emit ((u; v); $)
12: Reduce 2: Input ((u; w); S ⊆ V ∪ {$})
13:         if $ ∈ S then
14:                 for v ∈ S ∩ V do
15:                 emit (v; 1)
```

Όπως προαναφέρθηκε στον NodeIterator ένας απλός κόμβος υψηλού βαθμού (degree) μπορεί να οδηγήσει σε υψηλά κόστη υπολογισμού $O(n^2)$. Έτσι παρουσιάζουμε τη βελτιωμένη του έκδοση, τον NodeIterator++ ο οποίος προσαρμόζεται ως εξής στο περιβάλλον Map Reduce. Σημειώσετε ότι στο Βήμα 2, παίρνει ως input τα αποτελέσματα του Βήματος 1, αλλά ταυτόχρονα παίρνει και ως input την αρχική λίστα με τις σχέσεις (edges). Ο αλγόριθμος διαφοροποιείται ανάλογα τους δύο διαφορετικούς τύπους του input χρησιμοποιώντας έναν ειδικό χαρακτήρα, τον $ ο οποίος υποθέτει ότι «δεν εμφανίζεται πουθενά αλλού στο input».

Algorithm 3 MR-NodeIterator++(V,E)

```
1: Map 1: Input: ⟨(u, v); ∅⟩
2:     if v ≻ u then
3:         emit ⟨u; v⟩
4: Reduce 1: Input ⟨v; S ⊆ Γ(v)⟩
5:     for (u, w) : u, w ∈ S do
6:         emit ⟨v; (u, w)⟩
7: Map 2:
8:     if Input of type ⟨v; (u, w)⟩ then
9:         emit ⟨(u, w); v⟩
10:    if Input of type ⟨(u, v); ∅⟩ then
11:        emit ⟨(u, v); $⟩
12: Reduce 2: Input ⟨(u, w); S ⊆ V ∪ {$}⟩
13:    if $ ∈ S then
14:        for v ∈ S ∩ V do
15:            emit ⟨v; 1⟩
```

Ανάλυση του αλγόριθμου

Αναλύουμε τη συνολική χρήση του χώρου και του χρόνου της έκδοσης του NodeIterator++ προσαρμοσμένο για το MapReduce. Η διόρθωση στο συνολικό χρόνο τρεξίματος προκύπτει από την προηγούμενη, σειριακή έκδοση. Η ακόλουθη πρόταση υπονοεί ότι η συνολική μνήμη που χρειάζεται ανά μηχανή είναι sublinear.

Αν αναλύσουμε το συνολικό χρόνο και χώρο που χρησιμοποιεί ο NodeIterator++ στο MapReduce αποδεικνύουμε ότι:

Α) Η διόρθωση και ο συνολικός χρόνος τρεξίματος βελτιώνονται.

Β) Η συνολική μνήμη που απαιτείται σε κάθε μηχανή είναι sublinear. Έτσι, κανένας reducer δεν υπερφορτώνεται με πολλά δεδομένα ακόμη και αν υπάρχει κλίση στην καμπύλη των εισροών (input), (δηλ. μεγάλος αριθμός συνδέσεων σε λίγους κόμβους)

Αποδεικνύεται μαθηματικά ότι:

1 Το input σε κάθε περίπτωση reduce του πρώτου βήματος έχει O($\sqrt{m}$) συνδέσεις (edges).

2 Ο συνολικός αριθμός των εγγραφών του output στο τέλος της πρώτης μείωσης (reduce) είναι O($\sqrt{m^{3/2}}$) το οποίο σύμφωνα με τον Schank [30] είναι το βέλτιστο.

Αλγόριθμος Graph Partition

Στη συνέχεια παρουσιάζουμε έναν διαφορετικό αλγόριθμο για την μέτρηση των τριγώνων. Ο αλγόριθμος λειτουργεί με τον τεμαχισμό των γράφων σε επικαλυπτόμενα υποσύνολα. Δεδομένου αυτού του τεμαχισμού (partitioning) μπορούμε να χρησιμοποιήσουμε κάθε σειριακό αλγόριθμο μέτρησης τριγώνων σαν black-box σε κάθε τμήμα και στη συνέχεια απλά θα συνδυάσουν τα αποτελέσματα. Αποδεικνύεται ότι ο αλγόριθμος με αυτόν τον τρόπο επιτυγχάνει να είναι πολύ αποτελεσματικός στην εργασία μέτρησης των τριγώνων. Καθώς τα τμήματα γίνονται όλο και μικρότερα, η συνολική ποσότητα εργασίας που ξοδεύεται στην εύρεση όλων των τριγώνων παραμένει στα O (m3/2) ο οποίος είναι ο βέλτιστος χρόνος εργασίας (Th. Schank, [30]), όμως υπάρχουν περισσότερες μηχανές που κάνουν η κάθε μία λιγότερη δουλειά. Εν ολίγοις, ο αλγόριθμος με αποτελεσματικότητα χρησιμοποιεί έναν αλγόριθμο μέτρησης τριγώνων

που δουλεύει σε μια απλή μηχανή και κατανέμει τους υπολογισμούς του χωρίς να εκτινάξει τη συνολική ποσότητα της δουλειάς που γίνεται. Στη συνέχεια αναλύεται η διαδικασία που ακολουθεί ο αλγόριθμος.

Ανάλυση του αλγόριθμου:

Αρχικά οι κόμβοι (nodes) διαχωρίζονται σε ρ ίσα τμήματα:

- $V = V_1 \cup V_2 \cup \ldots \cup V_\rho$, όπου ρ>0, $V_i \cap V_j = 0$ για κάθε $i \neq j$.
- Ορίζουμε με το $V_{ijk} = V_i \cup V_j \cup V_k$ και
- Έστω $E_{ijk} = \{(u,w) \in E : u, w \in V_{ijk}\}$ είναι το σύνολο των συνδέσεων (edges) ανάμεσα στις κορυφές (nodes, vertices) V_i, V_j, και V_k. και
- Έστω $G_{ijk} = \{V_i \cup V_j \cup V_k, E_{ijk}\}$ είναι ο προκύπτων γράφος-υποσύνολο στο V_{ijk}
- Έστω $g = \cup_{i<j<k} G_{ijk}$ είναι το σύνολο των γράφων
- Ένας γράφος G_{ijk} περιέχει 3/ρ ποσοστό των κορυφών του αρχικού γράφου και κατά επέκταση ένα ποσοστό O(1/ρ^2) των συνδέσεων
- Εφόσον κάθε κόμβος του τριγώνου πρέπει να είναι μέρος του διαχωρισμού του V και το g περιέχει όλους τους συνδυασμούς των τμημάτων του V, κάθε τρίγωνο στον γράφο G εμφανίζεται τουλάχιστον σε έναν υπο-γράφο g, πιθανόν σε περισσότερους, έτσι
- Εισάγονται «βάρη» για να διορθώσουν πιθανές επικαλύψεις στα τρίγωνα. Ο αλγόριθμος εφαρμόζει μια διαβαθμισμένη μέτρηση (weighted count) του αριθμού των τριγώνων σε κάθε υπογράφο g. Τα βάρη αυτά (weights) διορθώνουν το γεγονός ότι ένα τρίγωνο μπορεί να μετρηθεί σε πολλούς υπογράφους.

Ο κώδικας του MapReduce για τον διαχωρισμό παρουσιάζεται εδώ:

Algorithm 4 MR-GraphPartition(V,E, ρ)

1: **Map 1:** Input: $\langle (u,v);1\rangle$
2: {Let $h(\cdot)$ be a universal hash function into $[0,\rho-1]$}
3: $i \leftarrow h(u)$
4: $j \leftarrow h(v)$
5: **for** $a \in [0,\rho-1]$ **do**
6: **for** $b \in [a+1,\rho-1]$ **do**
7: **for** $c \in [b+1,\rho-1]$ **do**
8: **if** $\{i,j\} \subseteq \{a,b,c\}$ **then**
9: emit $\langle (a,b,c);(u,v)\rangle$.
10: **Reduce 1:** Input $\langle (i,j,k);E_{ijk} \subseteq E\rangle$
11: Count triangles on G_{ijk}
12: **for** every triangle (u,v,w) **do**
13: $z \leftarrow 1$
14: **if** $h(u)=h(v)=h(w)$ **then**
15: $z \leftarrow \binom{h(u)}{2} + h(u)(\rho-h(u)-1) + \binom{\rho-h(u)-1}{2}$
16: **elif** $h(u)=h(v) \mid h(v)=h(w) \mid h(u)=h(w)$ **then**
17: $z \leftarrow \rho-2$
18: Scale triangle (u,v,w) weight by $1/z$

Στην γραμμή 15 φαίνεται ο συνολικός αριθμός των τριγώνων που εντοπίζονται στους υπογράφους και στις επόμενες γραμμές αυτά ταξινομούνται μέσω των «βαρών» ώστε να δοθεί ως output ο πραγματικός αριθμός τους στο τέλος.

Όσον αφορά την παράμετρο ρ, η οποία είναι δυνατόν να μεταβάλλεται κατά περίπτωση, σε υψηλά επίπεδα των r διαχειρίζεται το συνολικό χώρο που χρησιμοποιείται από τον αλγόριθμο ως (όπως μετριέται στο τέλος της φάσης Map) με το μέγεθος του input για κάθε reduce. Αυτή η διαχείριση (trade off) ποσοτικοποιείται ως εξής:

1 Το αναμενόμενο συνολικό μέγεθος της εισροής (input) σε κάθε reduce είναι O(m/ρ^2).

2 Ο αναμενόμενος συνολικά χώρος που χρησιμοποιείται στη φάση map είναι O(ρm).

Αυτό που παρουσιάζεται στην πρόταση 2 είναι πολύ σημαντικό διότι σημαίνει ότι αυξάνοντας το συνολικό χώρο που χρησιμοποιείται από τον αλγόριθμο επί 2, αυτό έχει ως αποτέλεσμα έναν παράγοντα βελτίωσης επί 4 σε όρους βελτίωσης της μνήμης του κάθε reducer. Αυτό είναι ιδιαίτερα θετικό στην πράξη καθώς η RAM (απαιτήσεις μνήμης των reducers) είναι συνήθως πολύ πιο ακριβή από το χώρο σε δίσκους (τον χώρο που απαιτείται για την αποθήκευση του output των mappers).

Τελικά αποδεικνύεται ότι και ο συγκεκριμένος αλγόριθμος είναι αποτελεσματικός, δηλαδή εφόσον $\rho \leq \sqrt{m}$ η συνολική δουλειά που γίνεται σε αυτήν την περίπτωση είναι O(m^3/2)).

Κεφάλαιο 4
Σχεδίαση & Υλοποίηση

4.1 Αναλυτική Περιγραφή εργασιών

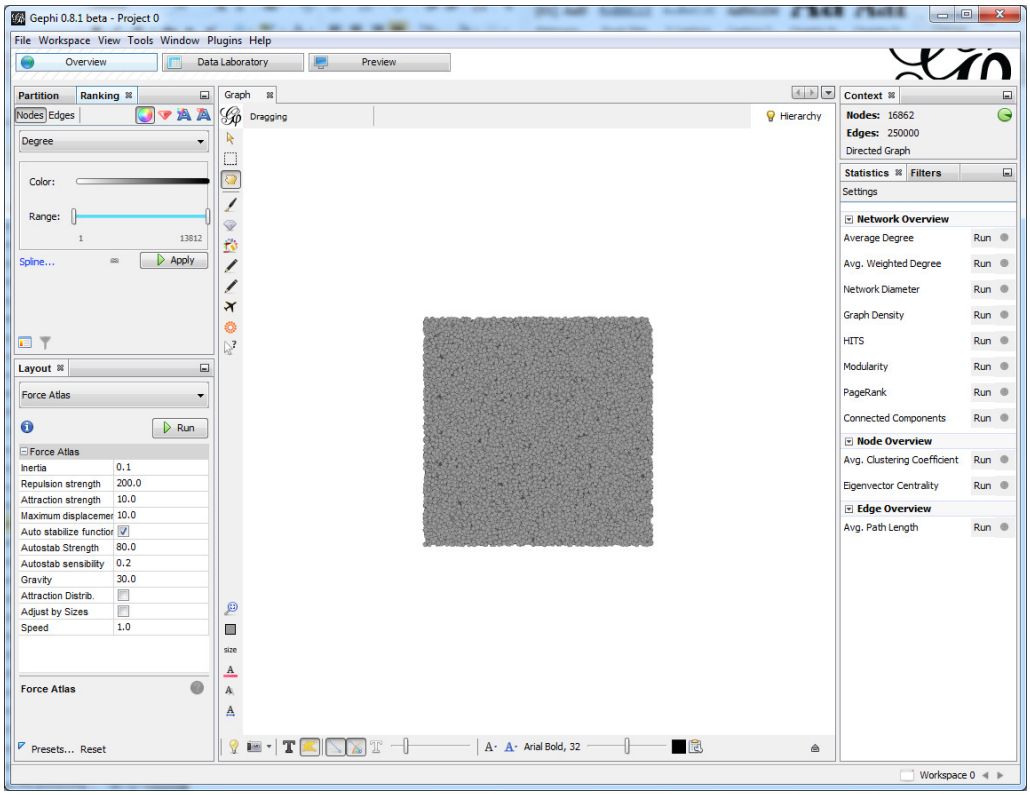
4.1.1 Εργαλεία οπτικοποίησης

Παράλληλα με την κατανόηση του θεωρητικού υπόβαθρου ξεκίνησαν και οι εργασίες για τις μετρήσεις που θα παρήγαγαν αποτελέσματα για την αξιολόγηση των αλγορίθμων. Παράλληλα έγινε έρευνα για εργαλεία γραφικής αποτύπωσης σχέσεων όπως αυτά δίνονται από τα κοινωνικά δίκτυα. Τα δεδομένα που χρησιμοποιήθηκαν ήταν αυτά του BerkStan και του Twitter, τα οποία είναι και τα πιο διαδεδομένα για την μελέτη κοινωνικών δικτύων.

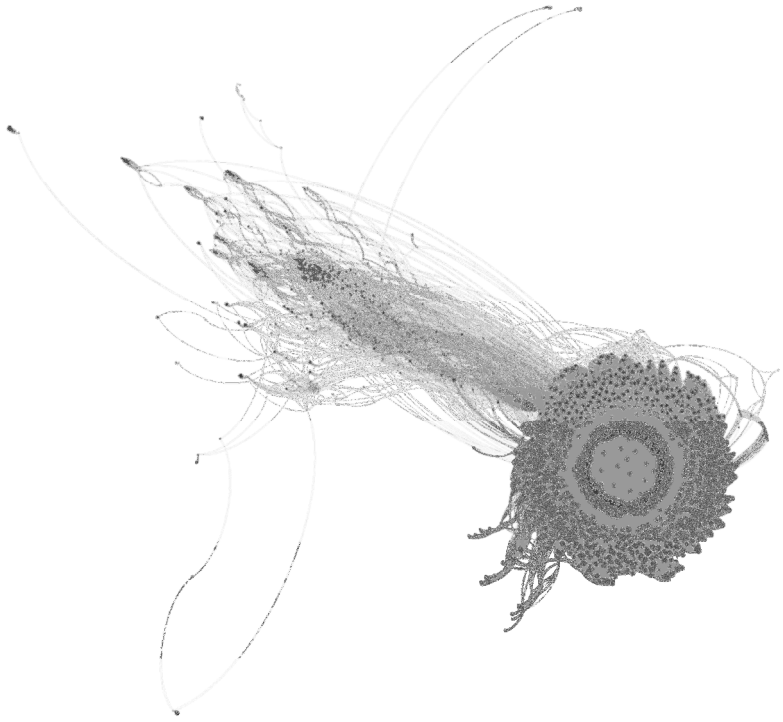
Στα πλαίσια της διατριβής και για την καλύτερη αποτύπωση και κατανόηση των δεδομένων έγινε και οπτικοποίηση αυτών. Η αρχική προσέγγιση ήταν με το εργαλείο SNAP, ίσως το πιο γνωστό από αυτά της κατηγορίας του. Όμως επειδή είχαν προηγηθεί αρκετές μελέτες και δοκιμές με το συγκριμένο εργαλείο, μελετήθηκαν και δοκιμάσθηκαν το Pegasus και το Gephi. . Μελετήθηκε και ένα τρίτο εργαλείο, το «ShiftR»[7], το οποίο φαίνεται να υποστηρίζει μεγάλου όγκου δεδομένων. Δεν δοκιμάσθηκε διότι δεν ήταν εφικτή η απόκτηση του εργαλείου για δοκιμές.

Για να έχουμε συγκρίσιμα γραφικά αποτελέσματα και στα δύο εργαλεία έγινε προσπάθεια για να φορτωθούν και τα δύο data sets. Αυτή η προσπάθεια δεν είχε επιτυχία για λόγους που θα εξηγηθούν στην παρακάτω ανάλυση των εργαλείων.

Τα αποτελέσματα από την γραφική απεικόνιση του Gephi, έχει αποτυπωθεί, εφαρμόζοντας το BerStan data set, δίνοντας ως μέτρο το Degree και ως παράμετρο layout το Atlas Force. Στο παρακάτω στιγμιότυπο δίνεται η εικόνα από το λογισμικό αφού έχει φορτωθεί το data set και έχουν δοθεί όλες οι παράμετροι εκτέλεσης.



[7] Duen Horng "Polo" Chau, Aniket Kittur, Jason I. Hong & Christos Faloutsos, «Supporting sensemaking in large network data», School of Computer Science - Carnegie Mellon University, April 2012



Εικόνα 4.1: Gephi screenshots

Από το παραπάνω εργαλείο μετά από 10 ημέρες λειτουργίας πήραμε το δεύτερο γράφημα της εικόνας 4.1 το οποίο έχει παρουσιαστεί ξανά στο Κεφάλαιο 2. Στην συνέχεια έγινε προσπάθεια να φορτωθεί και το Twitter data set, χωρίς επιτυχία διότι δεν μπορούσε ούτε να το ανοίξει λόγω μεγάλου μεγέθους.

Στην συνέχεια μελετήθηκε, εφαρμόσθηκε και δοκιμάσθηκε το Pegasus. Πρόκειται για ένα εργαλείο που περιγράφεται ως peta-scale graph mining system και λειτουργεί στο περιβάλλον του Hadoop. Έχει διαφορετική φιλοσοφία λειτουργίας από το Gephi και λόγω αυτής πάρθηκαν αποτελέσματα και το Twitter data set.

4.1.2 Περιβάλλον εργασίας και δοκιμών

Το περιβάλλον εργασίας που επιλέχθηκε ήταν αυτό του Ubuntu Linux (11.0). Σε πρώτη φάση για τις αρχικές έρευνες έγινε μέσω VMware Player (4.0.1) πάνω σε Windows 7. Αρχικά εγκαταστάθηκε η έκδοση 0.20.0 του Hadoop και στην συνέχεια η 0.22.0 για λόγους συμβατότητας με τον κώδικα δοκιμών.

Το προγραμματιστικό περιβάλλον που επιλέχθηκε είναι το Eclipse (Indigo), όπου εγκαταστάθηκε και το Karmasphere plug-in. Στην αρχή έγιναν δοκιμές από τα έτοιμα παραδείγματα που έδινε η εταιρεία με σκοπό την κατανόηση λειτουργίας τόσο του Hadoop αλλά και την εφαρμογής των map reduce αλγορίθμων.

Στην συνέχεια, στο παραπάνω υπολογιστικό περιβάλλον, συνεχίσθηκαν με το ίδιο εργαλείο και το τρέξιμο του MR-NodeIterator++ αλγορίθμου. Όμως καθ' όλη την διάρκεια των δοκιμών προέκυψαν προβλήματα σταθερότητας του λειτουργικού που έτρεχε σε VMware και έπρεπε να μεταφερθεί όλη η παραπάνω εγκατάσταση σε άλλο ισχυρότερο υπολογιστή, φορτώνοντας εξαρχής Ubuntu Linux. Ένας από τους βασικούς λόγους που οδήγησαν σε αυτή την απόφαση ήταν ότι η φόρτωση και η επεξεργασία μεγάλου όγκου δεδομένων σε περιβάλλον VMware δημιουργούσε προβλήματα στην ίδια την εγκατάσταση και υπήρχε αστοχία στην φόρτωση του λειτουργικού (καταστράφηκε η εγκατάσταση του vmware μετά από προσπάθεια να γίνει extend ο διαθέσιμος χώρος). Επίσης στην διάρκεια των δοκιμών υπήρξε και αστοχία σε hardware (καταστροφή HDD).

Μετά τις πρώτες δοκιμές και όταν άρχισε να σταθεροποιείται ο κώδικας, παραχωρήθηκε μια συστοιχία από τέσσερα nodes (Intel Core2Duo@2.13GHz, 4GB RAM, 250GB HDD 7200) που ήταν εγκατεστημένα στο Εθνικό Μετσόβιο Πολυτεχνείο και σ' αυτό το περιβάλλον έγιναν οι τελικές δοκιμές και πάρθηκαν τα αποτελέσματα που παρουσιάζονται στις επόμενες γραμμές.

4.2 Ανάπτυξη κώδικα

Κατά τη φάση ανάπτυξης κώδικα, όταν και χρησιμοποιήθηκαν τεχνικές Agile και iterative programming, οι αλγόριθμοι εφαρμόστηκαν σε μικρά τμήματα των datasets ώστε να ελεγχθεί σε εύλογο χρόνο η ορθή λειτουργία του λογισμικού.. Οι βασικές δυσκολίες που αντιμετωπίσθηκαν είχαν να κάνουν αφενός με την χρήση και κατανόηση των βιβλιοθηκών ώστε αυτές να μπορέσουν να ενσωματωθούν στον υπό συγγραφή κώδικα, αφετέρου με την μεταφορά του ίδιου του αλγόριθμου από τη θεωρία στην πράξη.

Η δομή του προγράμματος χωρίζεται σε τρείς βασικές φάσεις όπως αυτές αντικατοπτρίζονται στα software packages που αναπτύχθηκαν:

Η πρώτη είναι η preprocessing στην οποία γίνεται η μορφοποίηση των δεδομένων του data set και περιλαμβάνει τις κλάσεις: DegreeCalculator.java, DirectedToUndirectedGraph.java, DirectedToUndirectedGraph2.java, LongWritablePair.java, NodeWritable.java, SecondNodeTagger.java και TagEdgeList.java

Η δεύτερη είναι οι βοηθητικές κλάσεις (Utils) EdgeInputFormat.java, EdgeListRecordReader.java, OrderByDegree.java, SplittableCompressionCodec.java, TaggedInputSplit.java, TextPair.java.

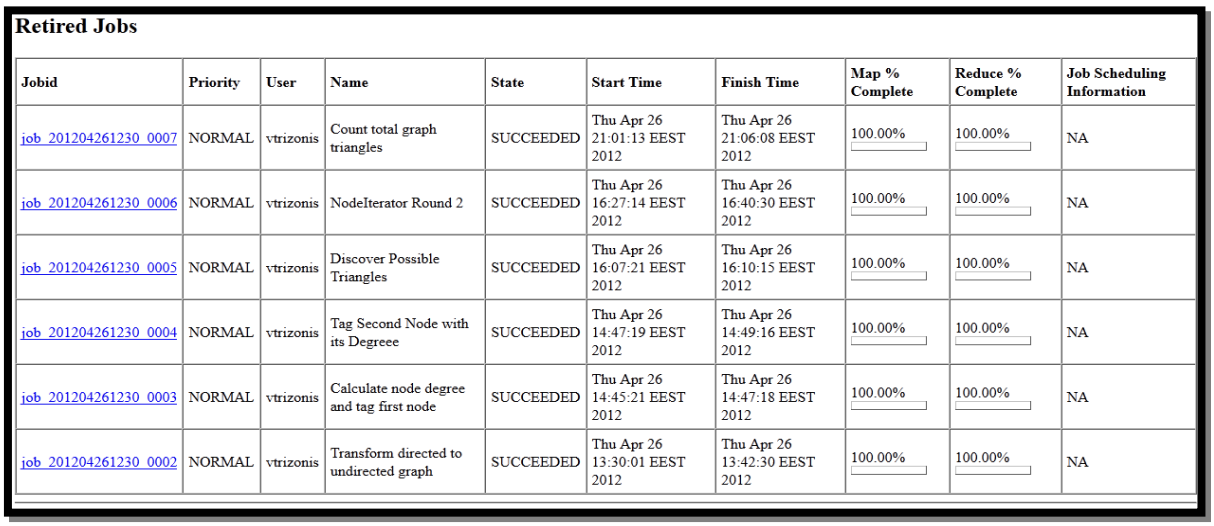
Και τέλος η NodeIterator όπου βρίσκονται οι βασικές κλάσεις του αλγορίθμου. Η πρώτη προς εκτέλεση είναι η NodeDegreesTagger κλάση, ακολουθεί η RoundOneNodeIterator, η RoundTwoNodeIterator και τέλος η κλάση TotalTriangleCounter στην οποία γίνεται ο υπολογισμός των τριγώνων.

Το σύνολο του κώδικα επισυνάπτεται ως παράρτημα στο τέλος της διατριβής με τον απαραίτητο σχολιασμό.

Κεφάλαιο 5
Πειράματα - Αποτελέσματα

Η εφαρμογή του αλγορίθμου Map-Reduce NodeIterator έγινε στο BerkStan data set για λόγους άμεσης απόδοσης αποτελεσμάτων αφενός και αφετέρου διότι η υπολογιστική ισχύς του συστήματός μας δεν ήταν αρκετή για να δώσει αποτελέσματα στα χρονικά περιθώρια που δόθηκαν για ένα μεγαλύτερο dataset όπως αυτό του twitter. Σε εκείνη την περίπτωση, η σημασία των αποτελεσμάτων στην επιτάχυνση επεξεργασίας δε θα ήταν δυνατό να διακριθεί στον άξονα του χρόνου.

Η εφαρμογή του MR-NodeIterator++ αλγορίθμου αποτυπώθηκε σε παραπάνω από μία java κλάσεις και η δοκιμή του έγινε σταδιακά σε 5 βήματα. Τα πρώτα δύο βήματα αφορούν την προετοιμασία των δεδομένων ώστε στην συνέχεια να προχωρήσει η επεξεργασία. Στην συνέχεια δίνεται μια ενδεικτική εικόνα (screenshot) στην οποία φαίνεται μία χαρακτηριστική δέσμη ενεργειών όπως αυτές αποτυπώνονται από τον jobtracker, κατά την πρώτη φάση εκτέλεσης του αλγορίθμου στον πρώτο κόμβο. Δεν επισυνάφθηκαν όλες οι εικόνες για λόγους οικονομίας.



Retired Jobs

Jobid	Priority	User	Name	State	Start Time	Finish Time	Map % Complete	Reduce % Complete	Job Scheduling Information
job_201204261230_0007	NORMAL	vtrizonis	Count total graph triangles	SUCCEEDED	Thu Apr 26 21:01:13 EEST 2012	Thu Apr 26 21:06:08 EEST 2012	100.00%	100.00%	NA
job_201204261230_0006	NORMAL	vtrizonis	NodeIterator Round 2	SUCCEEDED	Thu Apr 26 16:27:14 EEST 2012	Thu Apr 26 16:40:30 EEST 2012	100.00%	100.00%	NA
job_201204261230_0005	NORMAL	vtrizonis	Discover Possible Triangles	SUCCEEDED	Thu Apr 26 16:07:21 EEST 2012	Thu Apr 26 16:10:15 EEST 2012	100.00%	100.00%	NA
job_201204261230_0004	NORMAL	vtrizonis	Tag Second Node with its Degreee	SUCCEEDED	Thu Apr 26 14:47:19 EEST 2012	Thu Apr 26 14:49:16 EEST 2012	100.00%	100.00%	NA
job_201204261230_0003	NORMAL	vtrizonis	Calculate node degree and tag first node	SUCCEEDED	Thu Apr 26 14:45:21 EEST 2012	Thu Apr 26 14:47:18 EEST 2012	100.00%	100.00%	NA
job_201204261230_0002	NORMAL	vtrizonis	Transform directed to undirected graph	SUCCEEDED	Thu Apr 26 13:30:01 EEST 2012	Thu Apr 26 13:42:30 EEST 2012	100.00%	100.00%	NA

Εικόνα 5.1: Sample snapshot από jobtracker κατά την πρώτο τρέξιμο των πειραμάτων

5.1 Εκτέλεση πειραμάτων

Η εκτέλεση των πειραμάτων έγινε σε πέντε στάδια για κάθε κόμβο ξεχωριστά. Το καθένα από αυτά περιγράφεται αναλυτικά, συνοδευόμενο από ενδεικτικό screenshot (από την πρώτη εκτέλεση).

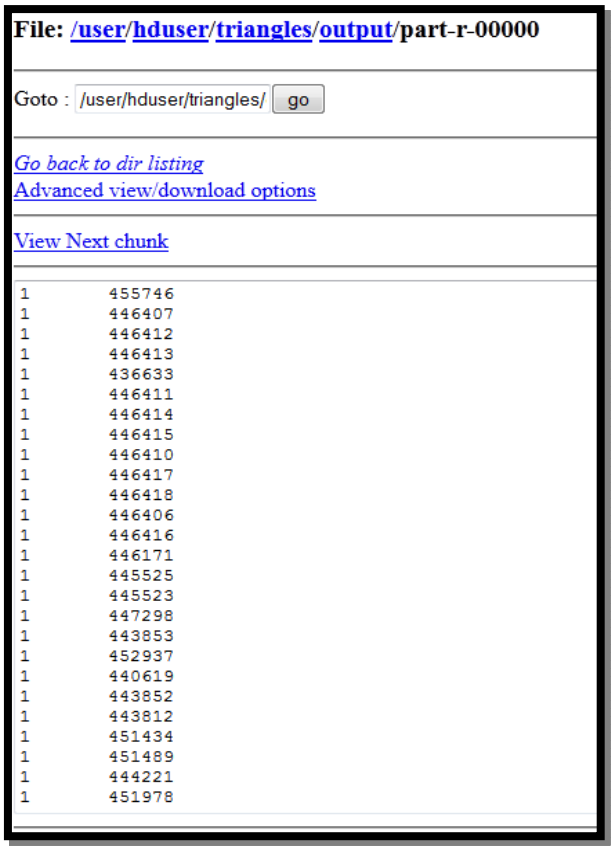
Το πρώτο βήμα «Transform directed to undirected graph» είναι η διαδικασία που μετατρέπουμε τον γράφο από κατευθυνόμενο σε μη κατευθυνόμενο. Αυτό είναι απαραίτητο καθώς ο αλγόριθμος απαιτεί την ύπαρξη μη-κατευθυνόμενου γράφου για να λειτουργήσει ορθά. Δηλαδή κάθε κατευθυνόμενη σχέση node1 -> node2, αναλύεται ως node1 – node2, node2 – node1. Ως παράδειγμα, για την εγγραφή 1 [tab] 455746 δημιουργείται και η 455746 [tab] 1.

Για να μεταφέρουμε το data set στο HDFS, δίνεται η Hadoop εντολή

```
hadoop dfs –copyFromLocal /usr/local/hadoop/dataset /user/hduser/triangles
```

και αφού μεταφερθεί επιτυχώς εκτελείται η κλάση ως: «`hadoop jar /usr/local/Hadoop/bin/triangles.jar preprocessing. DirectedToUndirectedGraph /user/hduser/triangles/ /user/hduser/triangles/output 1`». Ως παράμετροι δόθηκαν, το edgelist directory, το output directory και ο επιθυμητός αριθμός των reducers. Αφού ολοκληρωθεί

επιτυχώς (επιστρέφοντας μήνυμα «_success») το αποτέλεσμα το οποίο αποθηκεύεται στο αρχείο output θα είναι της παρακάτω μορφής:



Εικόνα 5.2: Sample snapshot από το output της 1ης φάσης

Και το 2ο βήμα αποτελεί φάση προετοιμασίας των δεδομένων, όπως είχε προαναφερθεί, και στο οποίο γίνεται ο υπολογισμός του degree για κάθε κόμβο « Calculate node degree and tag first node».

Σε αυτή τη φάση γίνονται διαδοχικά 2 διαδικασίες. Στην πρώτη υπολογίζεται το degree και γίνεται tag του πρώτου κόμβου (calculate node degree and tag first node). Η συγκεκριμένη διεργασία map reduce υπολογίζει το βαθμό (degree) για κάθε κόμβο στον γράφο. Μετά το τέλος αυτής της φάσης το αρχείο εξόδου (tagged) είναι μια αντεστραμμένη λίστα κόμβων του τύπου node2, node1 [tab] degree.

Στην επόμενη φάση του συγκεκριμένου βήματος (tag second node with its degree), ο reducer διαβάζει μέσα από τα εκπεμπόμενα μερικώς tagged pairs (προς επεξεργασία – επιλεγμένα

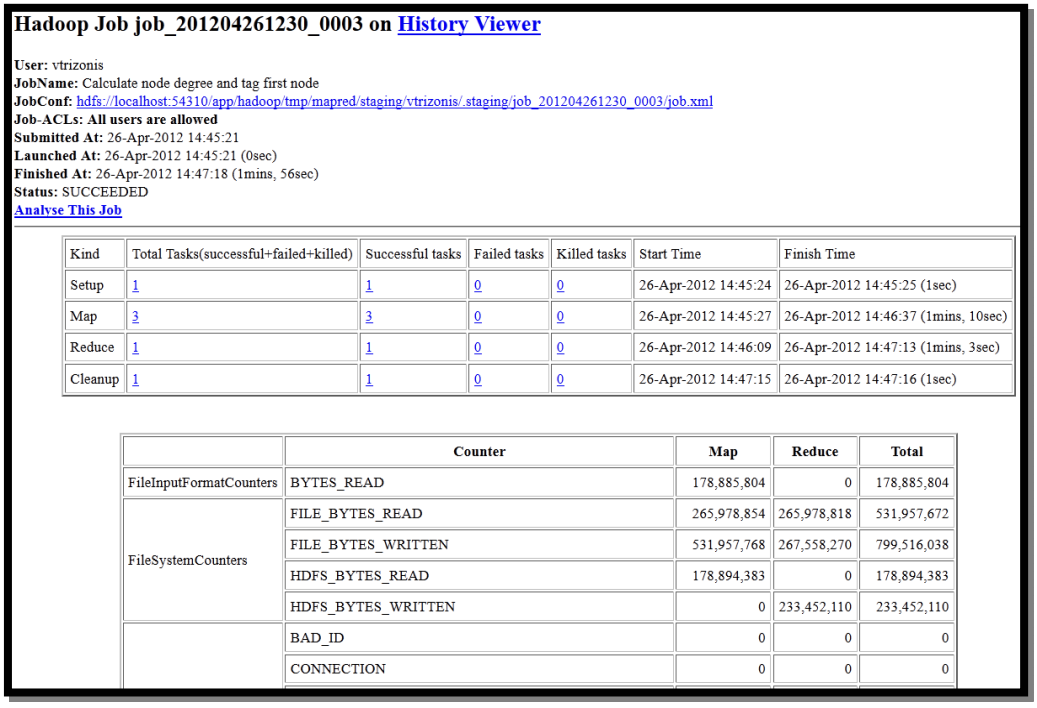
ζεύγη). Όταν ένα ζευγάρι σημειώνεται με "0", είναι πρώτο στην σειρά με το tag και το degree του. Τα επόμενα k,n ζεύγη θεωρούνται κόμβοι σχέσεων και ο πρώτος λαμβάνει το tag με το degree του.

Ως παραμέτρους παίρνουμε το αποτέλεσμα από την 1η φάση το οποίο αποθηκεύεται στο αρχείο tagged (HDFS). Το αρχείο θα είναι της μορφής: [node1 [tab] degree_of_node1 [tab] node2 [tab] degree_of_node2]

Η εντολή που δίνεται είναι:

```
hadoop jar /home/vtrizonis/workspace/triangles.jar
gr.ntua.stasino.triangles.nodeiterator.NodeDegreesTagger /user/hduser/triangles/output
/user/hduser/triangles/tagged/ 1
```

Στις παρακάτω εικόνες αποτυπώνονται τα στατιστικά στοιχεία εκτέλεσης των παραπάνω java κλάσεων όπως αυτά δίνονται από τον jobtracker. Μεταξύ άλλων δίνονται οι χρόνοι διάρκειας εκτέλεσης καθώς και όλες οι φάσεις εκτέλεσης της map reduce διαδικασίας.



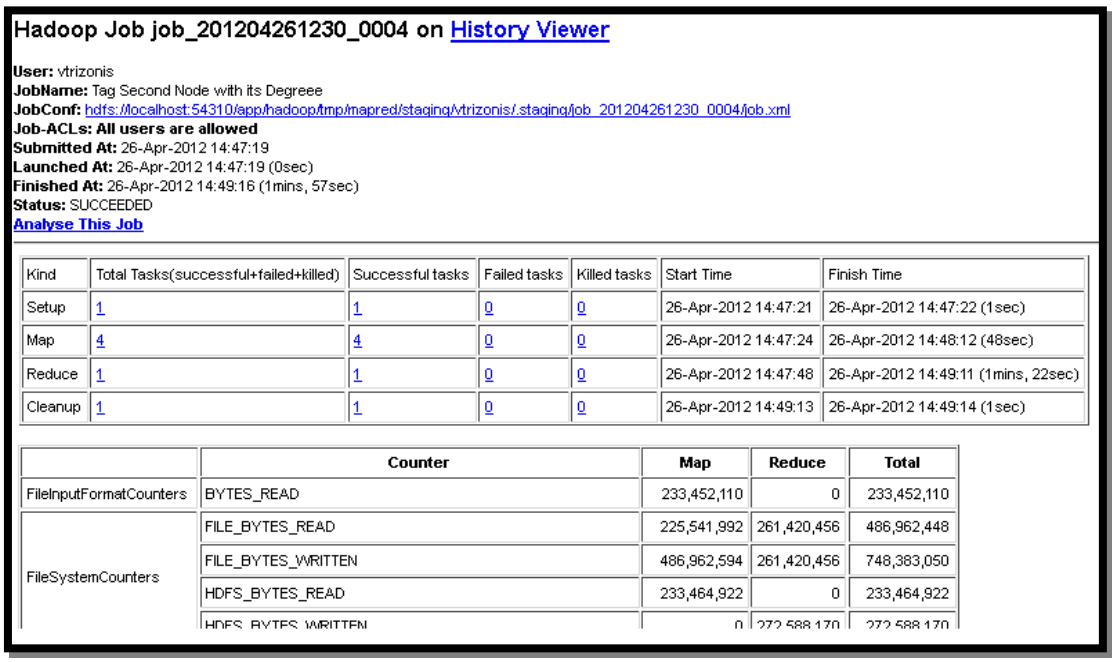
Hadoop Job job_201204261230_0003 on History Viewer

User: vtrizonis
JobName: Calculate node degree and tag first node
JobConf: hdfs://localhost:54310/app/hadoop/tmp/mapred/staging/vtrizonis/.staging/job_201204261230_0003/job.xml
Job-ACLs: All users are allowed
Submitted At: 26-Apr-2012 14:45:21
Launched At: 26-Apr-2012 14:45:21 (0sec)
Finished At: 26-Apr-2012 14:47:18 (1mins, 56sec)
Status: SUCCEEDED
Analyse This Job

Kind	Total Tasks(successful+failed+killed)	Successful tasks	Failed tasks	Killed tasks	Start Time	Finish Time
Setup	1	1	0	0	26-Apr-2012 14:45:24	26-Apr-2012 14:45:25 (1sec)
Map	3	3	0	0	26-Apr-2012 14:45:27	26-Apr-2012 14:46:37 (1mins, 10sec)
Reduce	1	1	0	0	26-Apr-2012 14:46:09	26-Apr-2012 14:47:13 (1mins, 3sec)
Cleanup	1	1	0	0	26-Apr-2012 14:47:15	26-Apr-2012 14:47:16 (1sec)

	Counter	Map	Reduce	Total
FileInputFormatCounters	BYTES_READ	178,885,804	0	178,885,804
FileSystemCounters	FILE_BYTES_READ	265,978,854	265,978,818	531,957,672
	FILE_BYTES_WRITTEN	531,957,768	267,558,270	799,516,038
	HDFS_BYTES_READ	178,894,383	0	178,894,383
	HDFS_BYTES_WRITTEN	0	233,452,110	233,452,110
	BAD_ID	0	0	0
	CONNECTION	0	0	0

Εικόνα 5.3: Sample snapshot κατά την πρώτο τρέξιμο των πειραμάτων.



Hadoop Job job_201204261230_0004 on History Viewer

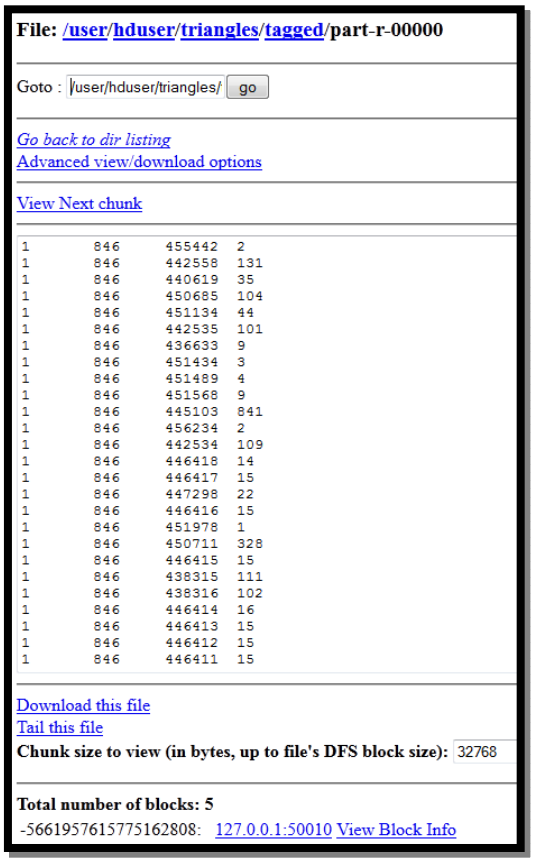
User: vtrizonis
JobName: Tag Second Node with its Degreee
JobConf: hdfs://localhost:54310/app/hadoop/tmp/mapred/staging/vtrizonis/.staging/job_201204261230_0004/job.xml
Job-ACLs: All users are allowed
Submitted At: 26-Apr-2012 14:47:19
Launched At: 26-Apr-2012 14:47:19 (0sec)
Finished At: 26-Apr-2012 14:49:16 (1mins, 57sec)
Status: SUCCEEDED
Analyse This Job

Kind	Total Tasks(successful+failed+killed)	Successful tasks	Failed tasks	Killed tasks	Start Time	Finish Time
Setup	1	1	0	0	26-Apr-2012 14:47:21	26-Apr-2012 14:47:22 (1sec)
Map	4	4	0	0	26-Apr-2012 14:47:24	26-Apr-2012 14:48:12 (48sec)
Reduce	1	1	0	0	26-Apr-2012 14:47:48	26-Apr-2012 14:49:11 (1mins, 22sec)
Cleanup	1	1	0	0	26-Apr-2012 14:49:13	26-Apr-2012 14:49:14 (1sec)

	Counter	Map	Reduce	Total
FileInputFormatCounters	BYTES_READ	233,452,110	0	233,452,110
FileSystemCounters	FILE_BYTES_READ	225,541,992	261,420,456	486,962,448
	FILE_BYTES_WRITTEN	486,962,594	261,420,456	748,383,050
	HDFS_BYTES_READ	233,464,922	0	233,464,922
	HDFS_BYTES_WRITTEN	0	272,588,170	272,588,170

Εικόνα 5.4: Sample snapshot από το output της 1ης φάσης

Επίσης παρατίθεται και ένα στιγμιότυπο από τις πρώτες εγγραφές από το tagged, όπως αυτό αποτυπώνεται μέσα από τον namenode φόρμα (http://localhost:50070) .



File: /user/hduser/triangles/tagged/part-r-00000

Goto : /user/hduser/triangles/ go

Go back to dir listing
Advanced view/download options

View Next chunk

```
1	846	455442	2
1	846	442558	131
1	846	440619	35
1	846	450685	104
1	846	451134	44
1	846	442535	101
1	846	436633	9
1	846	451434	3
1	846	451489	4
1	846	451568	9
1	846	445103	841
1	846	456234	2
1	846	442534	109
1	846	446418	14
1	846	446417	15
1	846	447298	22
1	846	446416	15
1	846	451978	1
1	846	450711	328
1	846	446415	15
1	846	438315	111
1	846	438316	102
1	846	446414	16
1	846	446413	15
1	846	446412	15
1	846	446411	15
```

Download this file
Tail this file
Chunk size to view (in bytes, up to file's DFS block size): 32768

Total number of blocks: 5
-5661957615775162808: 127.0.0.1:50010 View Block Info

Εικόνα 5.5: Sample snapshot από το output (tagged)

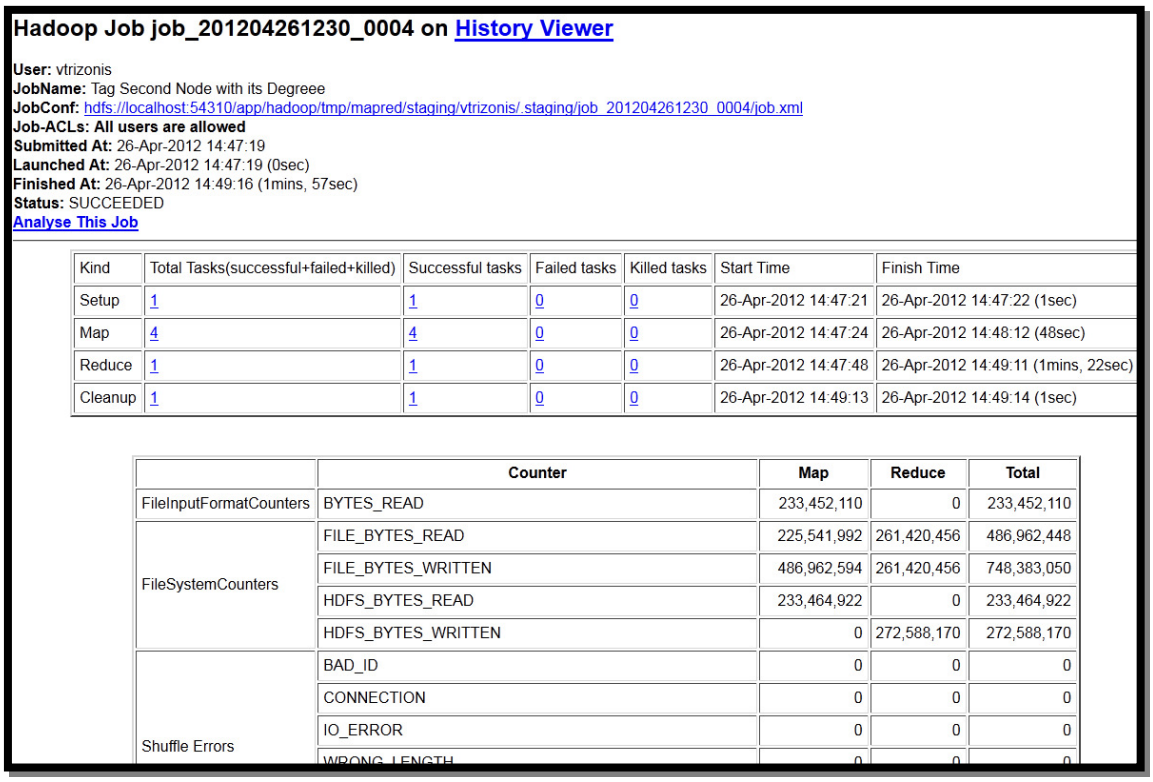
Στο 3ο βήμα «RoundOneNodeIterator» ξεκινά η εκτέλεση του πρώτο γύρου του αλγορίθμου. Σε αυτή την φάση υπολογίζονται τα πιθανά 2 μονοπάτια βαθμού 2 , δηλαδή ανιχνεύονται 3 κόμβοι που υπάρχει η πιθανότητα να δημιουργούν ένα τρίγωνο στο γράφο. Όπως έχει περιγραφεί και στην ανάλυση του αλγορίθμου στο 1ο βήμα δημιουργείται το πιθανό μήκος δύο μονοπατιών στο γράφο με την «περιστροφή» (pivoting) γύρω από κάθε κόμβο παράλληλα, όπου ο σκοπός της εκτέλεσης της πρώτης διαδικασίας, είναι η δημιουργία σε ζεύγη, όλων των πιθανών μονοπατιών μήκους δύο – δηλαδή δευτέρου βαθμού και άνω – στο γράφο, με τον περιστροφικό έλεγχο σε κάθε κόμβο, παράλληλα.

Τα δεδομένα εισόδου από την edgelist είναι αυτά από το προηγούμενο βήμα (tagged) με το βαθμό του κάθε κόμβου (degree) και τα αποτελέσματα αποθηκεύονται στο αρχείο roundone.

Η εντολή που δίνεται είναι:

```
hadoop jar /home/ouc/workspace/triangles.jar gr.ntua.stasino.triangles.nodeiterator
.RoundOneNodeIterator /user/hduser/triangles/tagged /user/hduser/triangles/roundone/ 1
```

Στην παρακάτω εικόνα αποτυπώνονται τα στατιστικά στοιχεία εκτέλεσης της παραπάνω java κλάσης όπως αυτά δίνονται από τον jobtracker:



Hadoop Job job_201204261230_0004 on History Viewer

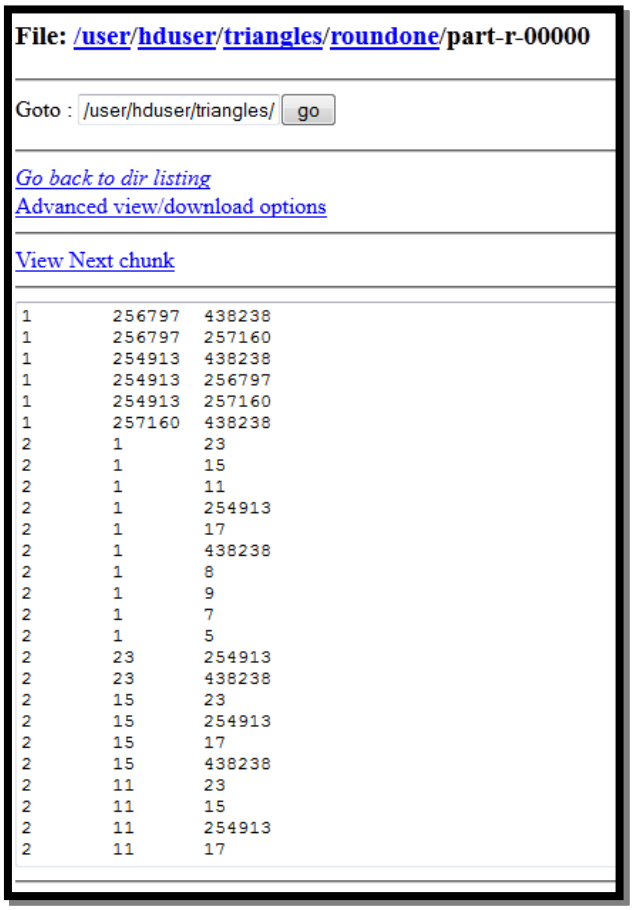
User: vtrizonis
JobName: Tag Second Node with its Degreee
JobConf: hdfs://localhost:54310/app/hadoop/tmp/mapred/staging/vtrizonis/.staging/job_201204261230_0004/job.xml
Job-ACLs: All users are allowed
Submitted At: 26-Apr-2012 14:47:19
Launched At: 26-Apr-2012 14:47:19 (0sec)
Finished At: 26-Apr-2012 14:49:16 (1mins, 57sec)
Status: SUCCEEDED
Analyse This Job

Kind	Total Tasks(successful+failed+killed)	Successful tasks	Failed tasks	Killed tasks	Start Time	Finish Time
Setup	1	1	0	0	26-Apr-2012 14:47:21	26-Apr-2012 14:47:22 (1sec)
Map	4	4	0	0	26-Apr-2012 14:47:24	26-Apr-2012 14:48:12 (48sec)
Reduce	1	1	0	0	26-Apr-2012 14:47:48	26-Apr-2012 14:49:11 (1mins, 22sec)
Cleanup	1	1	0	0	26-Apr-2012 14:49:13	26-Apr-2012 14:49:14 (1sec)

	Counter	Map	Reduce	Total
FileInputFormatCounters	BYTES_READ	233,452,110	0	233,452,110
FileSystemCounters	FILE_BYTES_READ	225,541,992	261,420,456	486,962,448
	FILE_BYTES_WRITTEN	486,962,594	261,420,456	748,383,050
	HDFS_BYTES_READ	233,464,922	0	233,464,922
	HDFS_BYTES_WRITTEN	0	272,588,170	272,588,170
Shuffle Errors	BAD_ID	0	0	0
	CONNECTION	0	0	0
	IO_ERROR	0	0	0
	WRONG_LENGTH	0	0	0

Εικόνα 5.6: Sample snapshot από το output της 3ης φάσης

Επίσης παρατίθεται και ένα στιγμιότυπο από τις πρώτες εγγραφές από το roundone, όπως αυτό αποτυπώνεται μέσα από τον namenode φόρμα (http://localhost:50070).



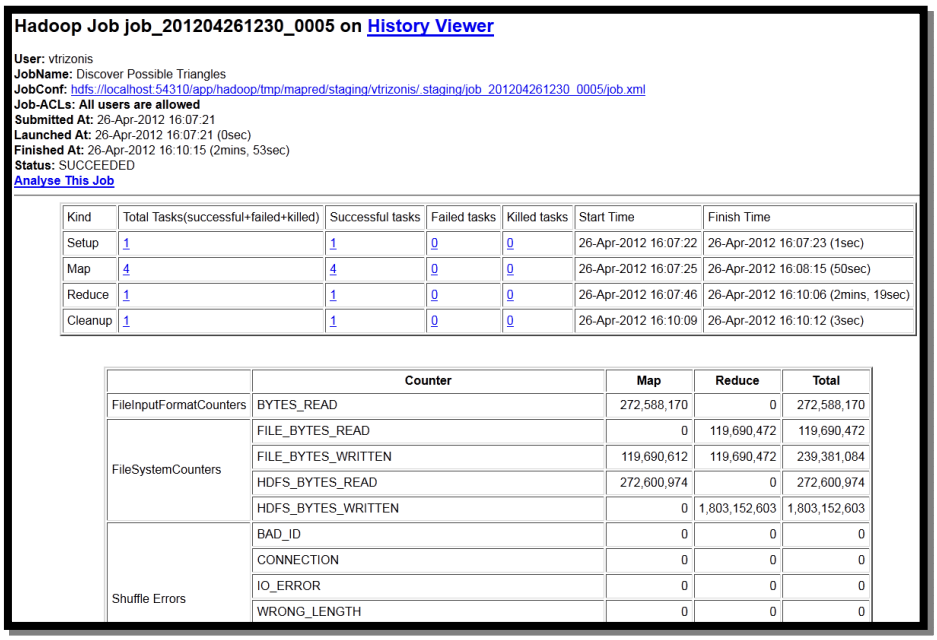
Εικόνα 5.7: Sample snapshot από το output της 3ης φάσης (roundone)

Στο 4ο το βήμα «RoundTwoNodeIterator» τρέχει η 2η φάση του αλγορίθμου, όπου ελέγχονται όλα τα ζεύγη που παράχθηκαν από την πρώτη διαδικασία, εάν «κλείνουν» με ακμή στο γράφο και κατάλληλα υπολογίζει και τον αριθμό των τριγώνων που προκύπτουν. Σημειώνουμε ότι μαζί με την έξοδο της πρώτης διαδικασίας που λαμβάνεται ως είσοδος, στη δεύτερη διαδικασία μπαίνει στην είσοδο και η αρχική λίστα του γράφου.

Η εντολή που δίνεται είναι:

```
hadoop jar /home/ouc/workspace/triangles.jar
gr.ntua.stasino.triangles.nodeiterator.RoundTwoNodeIterator /user/hduser/triangles/roundone
/user/hduser/triangles/output /user/hduser/triangles/roundtwo/ 1
```

Στην παρακάτω εικόνα αποτυπώνονται τα στατιστικά στοιχεία εκτέλεσης της παραπάνω java κλάσης όπως αυτά δίνονται από τον jobtracker:



Hadoop Job job_201204261230_0005 on History Viewer

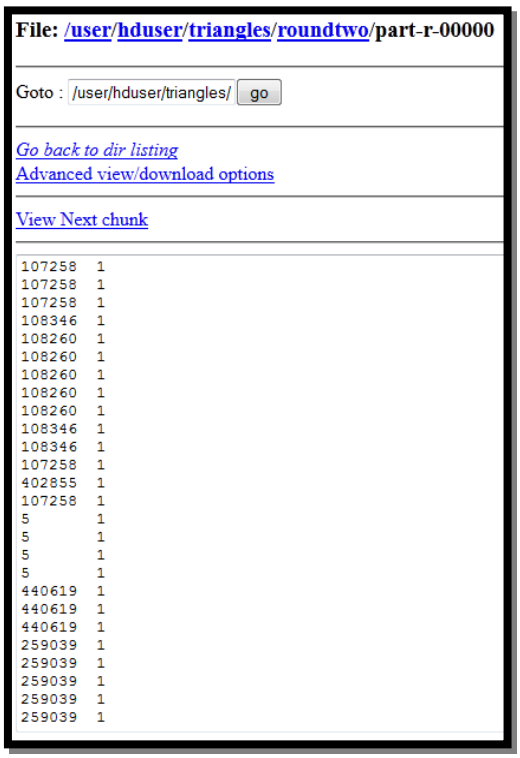
User: vtrizonis
JobName: Discover Possible Triangles
JobConf: hdfs://localhost:54310/app/hadoop/tmp/mapred/staging/vtrizonis/.staging/job_201204261230_0005/job.xml
Job-ACLs: All users are allowed
Submitted At: 26-Apr-2012 16:07:21
Launched At: 26-Apr-2012 16:07:21 (0sec)
Finished At: 26-Apr-2012 16:10:15 (2mins, 53sec)
Status: SUCCEEDED
Analyse This Job

Kind	Total Tasks(successful+failed+killed)	Successful tasks	Failed tasks	Killed tasks	Start Time	Finish Time
Setup	1	1	0	0	26-Apr-2012 16:07:22	26-Apr-2012 16:07:23 (1sec)
Map	4	4	0	0	26-Apr-2012 16:07:25	26-Apr-2012 16:08:15 (50sec)
Reduce	1	1	0	0	26-Apr-2012 16:07:46	26-Apr-2012 16:10:06 (2mins, 19sec)
Cleanup	1	1	0	0	26-Apr-2012 16:10:09	26-Apr-2012 16:10:12 (3sec)

	Counter	Map	Reduce	Total
FileInputFormatCounters	BYTES_READ	272,588,170	0	272,588,170
FileSystemCounters	FILE_BYTES_READ	0	119,690,472	119,690,472
	FILE_BYTES_WRITTEN	119,690,612	119,690,472	239,381,084
	HDFS_BYTES_READ	272,600,974	0	272,600,974
	HDFS_BYTES_WRITTEN	0	1,803,152,603	1,803,152,603
Shuffle Errors	BAD_ID	0	0	0
	CONNECTION	0	0	0
	IO_ERROR	0	0	0
	WRONG_LENGTH	0	0	0

Εικόνα 5.8: Sample snapshot από το output της 4ης φάσης (jobtracker)

Επίσης παρατίθεται και ένα στιγμιότυπο από τις πρώτες εγγραφές από το roundtwo, όπως αυτό αποτυπώνεται μέσα από τον namenode φόρμα (http://localhost:50070).



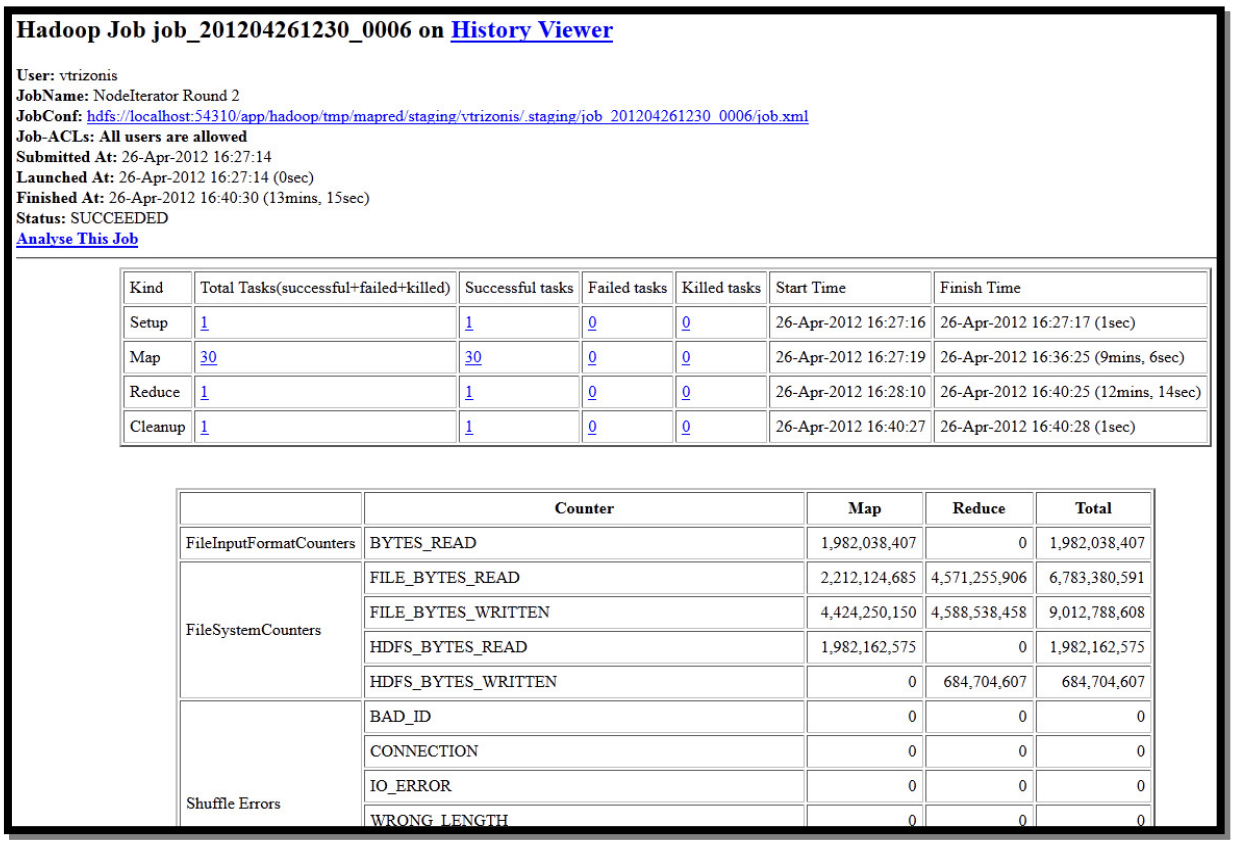
Εικόνα 5.9: Sample snapshot από το output της 4ης φάσης (roundtwo)

Τέλος, στο 5ο και τελικό βήμα γίνεται ο υπολογισμός των τριγώνων (Total Triangle Counter). Ως παράμετροι είναι το αρχείο εισόδου roundtwo, το output και το αποτέλεσμα αποθηκεύεται στο final.

Η εντολή που δίνεται είναι:

```
hadoop jar /home/ouc/workspace/triangles.jar
gr.ntua.stasino.triangles.nodeiterator.TotalTriangleCounter /user/hduser/triangles/roundtwo/
/user/hduser/triangles/final 1
```

Στην παρακάτω εικόνα αποτυπώνονται τα στατιστικά στοιχεία εκτέλεσης της παραπάνω java κλάσης όπως αυτά δίνονται από τον jobtracker:



Hadoop Job job_201204261230_0006 on History Viewer

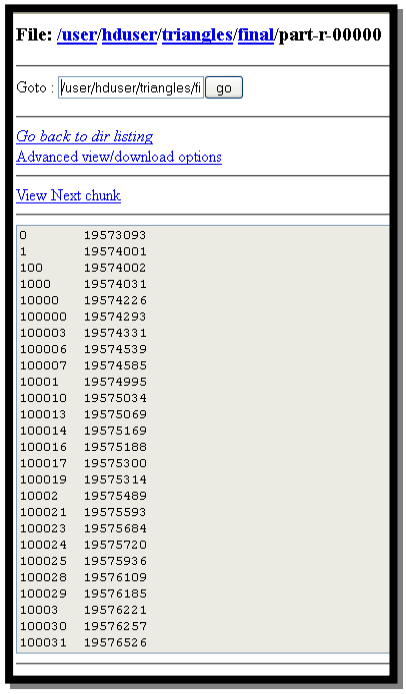
User: vtrizonis
JobName: NodeIterator Round 2
JobConf: hdfs://localhost:54310/app/hadoop/tmp/mapred/staging/vtrizonis/.staging/job_201204261230_0006/job.xml
Job-ACLs: All users are allowed
Submitted At: 26-Apr-2012 16:27:14
Launched At: 26-Apr-2012 16:27:14 (0sec)
Finished At: 26-Apr-2012 16:40:30 (13mins, 15sec)
Status: SUCCEEDED
Analyse This Job

Kind	Total Tasks(successful+failed+killed)	Successful tasks	Failed tasks	Killed tasks	Start Time	Finish Time
Setup	1	1	0	0	26-Apr-2012 16:27:16	26-Apr-2012 16:27:17 (1sec)
Map	30	30	0	0	26-Apr-2012 16:27:19	26-Apr-2012 16:36:25 (9mins, 6sec)
Reduce	1	1	0	0	26-Apr-2012 16:28:10	26-Apr-2012 16:40:25 (12mins, 14sec)
Cleanup	1	1	0	0	26-Apr-2012 16:40:27	26-Apr-2012 16:40:28 (1sec)

	Counter	Map	Reduce	Total
FileInputFormatCounters	BYTES_READ	1,982,038,407	0	1,982,038,407
FileSystemCounters	FILE_BYTES_READ	2,212,124,685	4,571,255,906	6,783,380,591
	FILE_BYTES_WRITTEN	4,424,250,150	4,588,538,458	9,012,788,608
	HDFS_BYTES_READ	1,982,162,575	0	1,982,162,575
	HDFS_BYTES_WRITTEN	0	684,704,607	684,704,607
Shuffle Errors	BAD_ID	0	0	0
	CONNECTION	0	0	0
	IO_ERROR	0	0	0
	WRONG_LENGTH	0	0	0

Εικόνα 5.10: Sample snapshot από το output της 4ης φάσης (jobtracker)

Επίσης παρατίθεται και ένα στιγμιότυπο από τις πρώτες εγγραφές από το final, όπως αυτό αποτυπώνεται μέσα από τον namenode φόρμα (http://localhost:50070).



File: /user/hduser/triangles/final/part-r-00000

Goto : /user/hduser/triangles/fi go

Go back to dir listing
Advanced view/download options

View Next chunk

```
0       19573093
1       19574001
100     19574002
1000    19574031
10000   19574226
100000  19574293
100003  19574331
100006  19574539
100007  19574585
10001   19574995
100010  19575034
100013  19575069
100014  19575169
100016  19575188
100017  19575300
100019  19575314
10002   19575489
100021  19575593
100023  19575684
100024  19575720
100025  19575936
100028  19576109
100029  19576185
10003   19576221
100030  19576257
100031  19576526
```

Εικόνα 5.11: Sample snapshot από το output της τελικής φάσης (final)

Στο σημείο αυτό ολοκληρώνεται η υλοποίηση του NodeIterator++ σε MapReduce όπου σαν αποτέλεσμα φέρνει τον κόμβο και τον αριθμό των τριγώνων στα οποία αυτός συμμετέχει. Στην συνέχεια τα τρίγωνα χρησιμοποιούνται για τον υπολογισμό του clustering coefficient (local, average & global), μέτρου της κοινωνικότητας του γράφου, όπως αυτός έχει παρουσιαστεί στο κεφάλαιο 3.

Τα παραπάνω βήματα εκτελέστηκαν μία φορά για κάθε κόμβο, σε σύνολο τεσσάρων, του cluster που χρησιμοποιήθηκε για τα πειράματα που έγιναν στα πλαίσια της παρούσης διπλωματικής διατριβής. Οι οθόνες που δίνονται είναι screenshots που πάρθηκαν κατά την διαδικασία αυτών των δοκιμών.

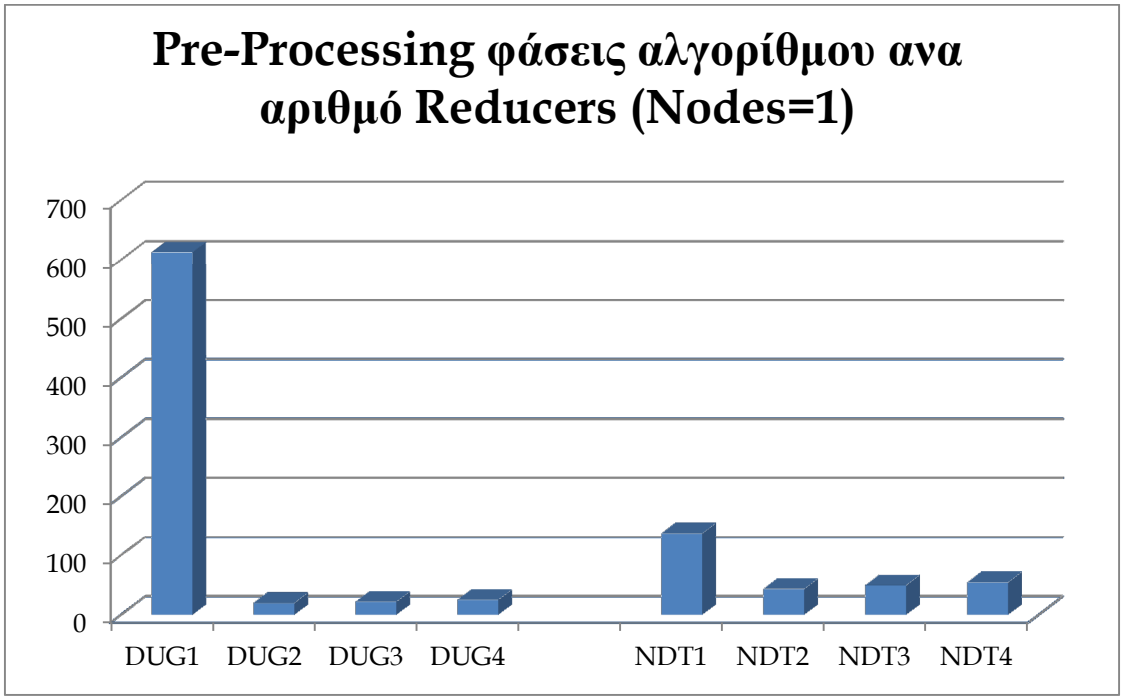
5.2 Παρουσίαση αποτελεσμάτων

Τα βήματα της παραπάνω διαδικασίας έτρεξαν σε κάθε κόμβο τέσσερις φορές, μία για κάθε reducer, ως παράμετρο. Κάθε κόμβος είχε 2 reduce tasks. Στην συνέχεια θα αναλυθεί η απόδοση του αλγορίθμου σε σχέση με τον αριθμό των reducers για κάθε run. Όπως προαναφέρθηκε το dataset που χρησιμοποιήθηκε για τις δοκιμές είναι το berkstan. Τα δεδομένα πριν την εκτέλεση του βασικού αλγορίθμου τρέχουν δύο κλάσεις προετοιμασίας των δεδομένων (Directed To Undirected Graph και Node Degree Tagger). Οι χρόνοι εκτέλεσης για τις παραπάνω κλάσεις αποτυπώνονται στους παρακάτω πίνακες, για κάθε κόμβο ξεχωριστά.

Pre-Processing φάσεις αλγορίθμου ανά αριθμό Reducers Node 1	**Χρόνος εκτέλεσης (Sec)**
TUG1 (Directed To Undirected Graph (Reducers=1))	613
TUG2 (Directed To Undirected Graph (Reducers=2))	20
TUG3 (Directed To Undirected Graph (Reducers=3))	22
TUG4 (Directed To Undirected Graph (Reducers=4))	25
NDT1 (Node Degree Tagger (Reducers=1))	138
NDT2 (Node Degree Tagger (Reducers=2))	43
NDT3 (Node Degree Tagger (Reducers=3))	49
NDT4 (Node Degree Tagger (Reducers=4))	54

Πίνακας 5.12: Pre-Processing φάσεις αλγορίθμου ανά αριθμό Reducers - Node 1

Στον παραπάνω πίνακα δίνονται τα αποτελέσματα εκτέλεσης των κλάσεων που μορφοποιούν τα δεδομένα για επεξεργασία από την κύρια φάση εκτέλεσης του αλγορίθμου, που ακολουθεί. Στην πρώτη φάση μετατρέπονται ο γράφος από directed σε undirected και στην δεύτερη φάση (Node Degree Tagger), όπου μια map reduce εργασία υπολογίζει το degree για κάθε κόμβο στον γράφο. Η ανάλυση των αποτελεσμάτων δίνεται παρακάτω μετά τον συγκριτικό πίνακα όλων των αποτελεσμάτων, το οποίο θα επαναλαμβάνεται για κάθε run σε κάθε κόμβο. Τα παραπάνω αποτελέσματα αποτυπώνονται γραφικά ως:



Εικόνα 5.13: Pre-Processing φάσεις αλγορίθμου ανά αριθμό Reducers - Node 1

Στην συνέχεια, μετά την προ-επεξεργασία, ακολουθεί η κύρια και βασική ροή του αλγορίθμου. Η πρώτη κλάση «Round One NodeIterator» είναι το πρώτο τμήμα υλοποίησης του αλγορίθμου MR-NodeIterator++, το οποίο υπολογίζει τα πιθανά 2 μονοπάτια (για παράδειγμα κόμβους που μπορεί να σχηματίζουν ένα τρίγωνο στο γράφο). Διαβάζει από την edge list η οποία δίνεται ως παράμετρος (τη λίστα με τις πιθανές συνδέσεις κάθε κόμβου) και ενισχύεται από τον βαθμό των κόμβων, όπως είχε υπολογισθεί πρωτύτερα.

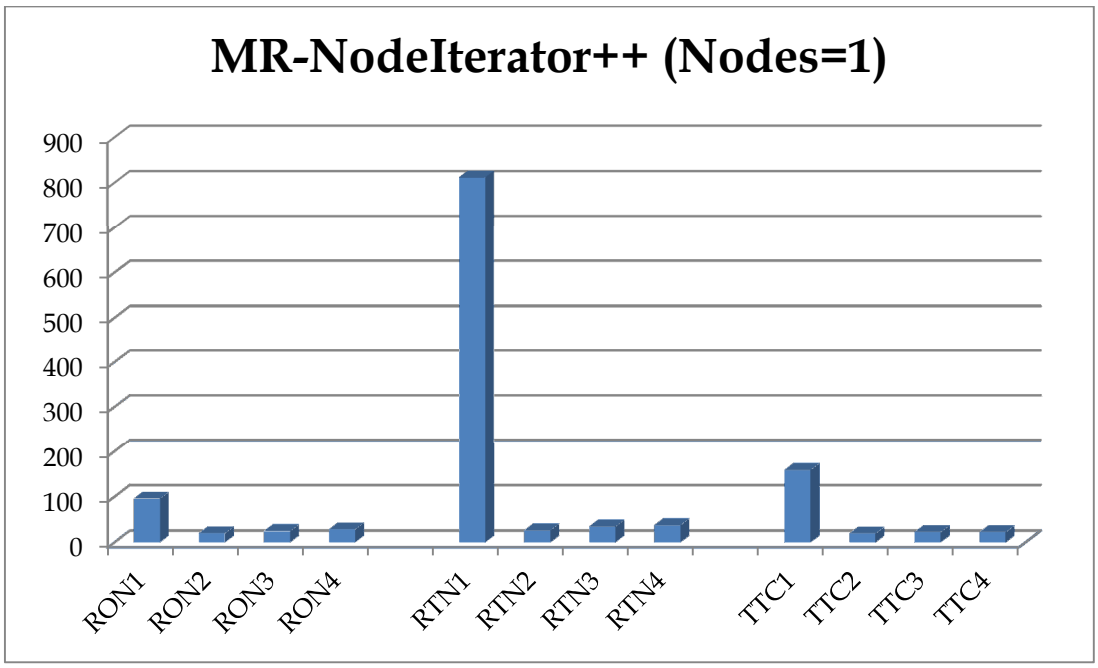
Στην συνέχεια τα αποτελέσματα της παραπάνω κλάσης δίνονται ως παράμετρος στην κλάση «Round Two NodeIterator», η οποία είναι mapper που διαβάζει μέσα από την πιθανή λίστα τρίγωνων. Τέλος, με τα αποτελέσματα και αυτής της φάσης, γίνονται οι τελικοί υπολογισμοί με την κλάση «Total Triangles Counter».

Στιγμιότυπο από την μορφή των δεδομένων εξόδου δόθηκε παραπάνω. Σε κάθε φάση εκτέλεσης των κλάσεων παίρνονται μετρήσεις, όπως αυτές δίνονται στους πίνακες που παραθέτουμε:

Φάσεις εκτέλεσης αλγορίθμου ανά αριθμό reducers Nodes =1	**Χρόνος εκτέλεσης (Sec)**
RON1 (Round One NodeIterator (Reducers=1)	96
RON2 (Round One NodeIterator (Reducers=2)	20
RON3 (Round One NodeIterator (Reducers=3)	25
RON4 (Round One NodeIterator (Reducers=4)	29
RTN1 = Round Two NodeIterator (Reducers=1)	812
RTN2 = Round Two NodeIterator (Reducers=2)	27
RTN3 = Round Two NodeIterator (Reducers=3)	35
RTN4 = Round Two NodeIterator (Reducers=4)	38
TTC1 = Total Triangles Counter (Reducers=1)	159
TTC2 = Total Triangles Counter (Reducers=2)	20
TTC3 = Total Triangles Counter (Reducers=3)	23
TTC4 = Total Triangles Counter (Reducers=4)	23

Πίνακας 5.14: Φάσεις εκτέλεσης αλγορίθμου ανά αριθμό reducers- Node 1

Στον παραπάνω πίνακα αποτυπώνονται οι χρόνοι εκτέλεσης για τον 1ο κόμβο. Τα ευρήματα επεξηγούνται στον συγκριτικό πίνακα. Οι παραπάνω μετρήσεις αποτυπώνονται γραφικά για υπάρχει ένα οπτικό μέτρο σύγκρισης των αποτελεσμάτων.



Εικόνα 5.15: Φάσεις εκτέλεσης αλγορίθμου ανά αριθμό reducers- Node 1

Στην συνέχεια λαμβάνοντας υπόψη τα παραπάνω αποτελέσματα και για να είναι πιο εύκολη η ανάδειξη των συμπερασμάτων, ομαδοποιήθηκαν σε συγκριτικούς πίνακες. Μαζί με τους πίνακες αυτούς, έναν για κάθε κόμβο που έγινε το τρέξιμο του κώδικα, δίνεται και η γραφική παράσταση για ευκολότερη κατανόηση του αποτελέσματος.

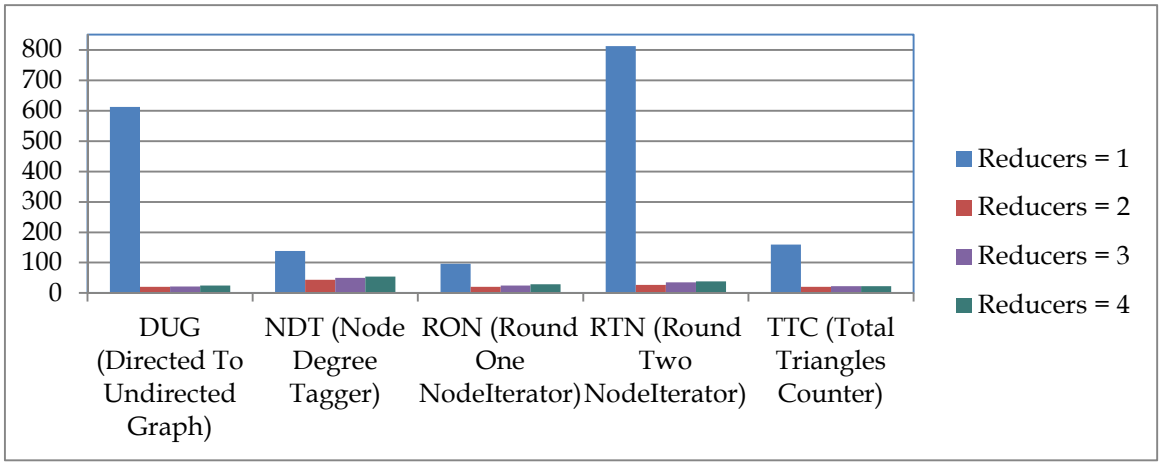
Στον άξονα x δίνονται οι χρόνοι εκτέλεσης και στον y ομαδοποιημένα τα στάδια εκτέλεσης του αλγορίθμου τόσο για το preprocessing όσο και για το κύριο στάδιο εκτέλεσης του αλγορίθμου. Όπως αποτυπώνεται και στο υπόμνημα του γραφήματος με διαφορετικά χρώματα δίνονται και οι reducers που χρησιμοποιήθηκαν.

Στον παρακάτω συγκριτικός πίνακα αποτυπώνονται όλες οι φάσεις εκτέλεσης του αλγορίθμου ανά αριθμό reducers.

Συγκριτικος πίνακας εφαρμογής αλγορίθμου NodeIterator κατά αριθμό reducers στον Node 1						
Node 1	Φάσεις εκτέλεσης	Reducers = 1	Reducers = 2	Reducers = 3	Reducers = 4	Red3-Red2
Pre-Processing	DUG (Directed To Undirected Graph)	613	20	22	25	5
	NDT (Node Degree Tagger)	138	43	49	54	11
Αλγόριθμος NodeIterator	RON (Round One NodeIterator)	96	20	25	29	9
	RTN (Round Two NodeIterator)	812	27	35	38	11
	TTC (Total Triangles Counter)	159	20	23	23	3
						*Οι μετρήσεις είναι σε sec

Πίνακας 5.16: Συγκριτικός πίνακας εφαρμογής αλγορίθμου NodeIterator κατά αριθμό reducers στον Node 1

Παρατηρώντας τους χρόνους ανά προσθήκη reducer στο πείραμα βλέπουμε μια θεαματική βελτίωση από τον ένα στους δύο reducers το οποίο είναι αποδεκτό. Αυτό που παρατηρούμε είναι μια αύξηση του χρόνου μετά την προσθήκη του 3ου και 4ου reducer. Αυτό συμβαίνει διότι στην παρούσα φάση χρησιμοποιείται ένας node, ο οποίος έχει 2 reducers και όταν ζητηθεί η συνδρομή και άλλων reducers το σύστημα μη έχοντας να τους διαθέσει, συνεχίζει την επεξεργασία με τους 2 που έχει, άσχετα με τους πόσους ζητήθηκαν να συμμετέχουν. Το υπολογιστικό κόστος για αυτή την διαδικασία δικαιολογεί την αύξηση του χρόνου επεξεργασίας. Οι συγκριτικοί χρόνοι ανά φάση και reducers, αποτυπώνονται γραφικά ως:



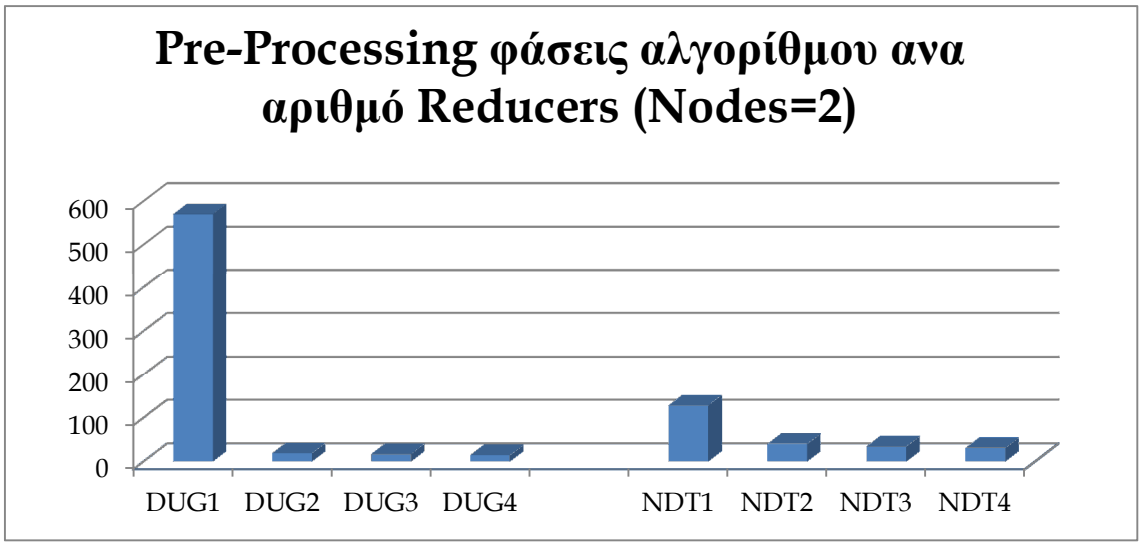
Εικόνα 5.17: Συγκριτικός πίνακας εφαρμογής αλγορίθμου NodeIterator κατά αριθμό reducers (Node 1)

Στην συνέχεια, το πείραμα συνεχίσθηκε, ενεργοποιώντας και τον 2ο κόμβο και ξεκινώντας ξανά τα πειράματα από την αρχή, ακολουθώντας τα ίδια βήματα που έγιναν για τον έναν κόμβο, παίρνουμε τα παρακάτω αποτελέσματα που αφορούν τους χρόνους εκτέλεσης ανά φάση:

Pre-Processing φάσεις αλγορίθμου ανά αριθμό Reducers **2 Nodes**	**Χρόνος εκτέλεσης (Sec)**
TUG1 (Directed To Undirected Graph (Reducers=1))	570
TUG2 (Directed To Undirected Graph (Reducers=1))	19
TUG3 (Directed To Undirected Graph (Reducers=3))	17
TUG4 (Directed To Undirected Graph (Reducers=4))	15
NDT1 (Node Degree Tagger (Reducers=1))	128
NDT2 (Node Degree Tagger (Reducers=2))	40
NDT3 (Node Degree Tagger (Reducers=3))	34
NDT4 (Node Degree Tagger (Reducers=4))	31

Πίνακας 5.18: Pre-Processing φάσεις αλγορίθμου ανά αριθμό Reducers - Node 2

Τα παραπάνω αποτελέσματα αποτυπώνονται γραφικά ως:



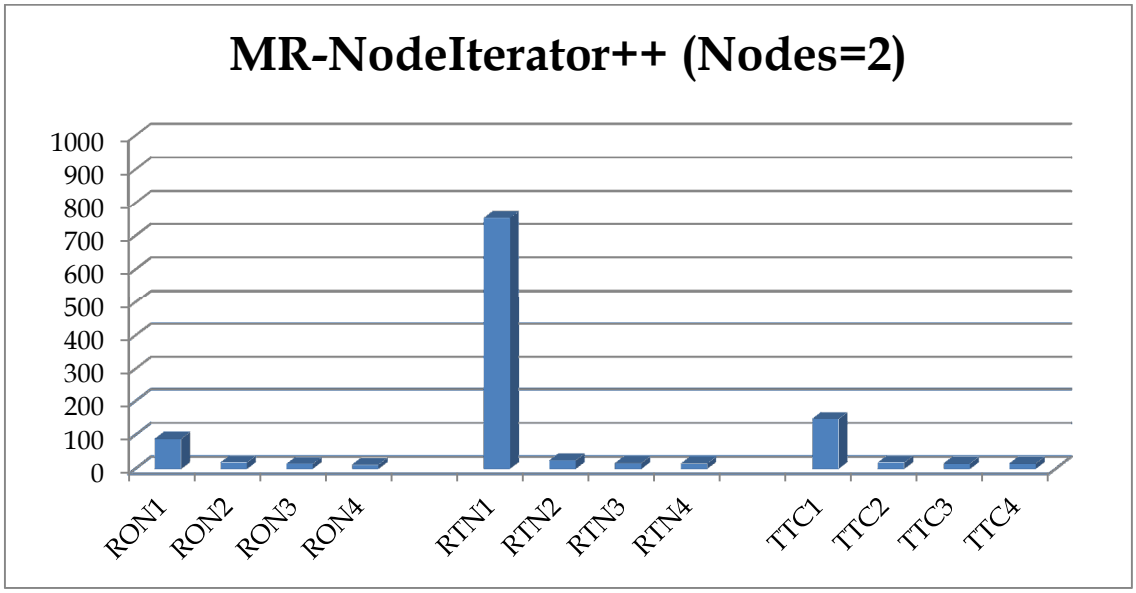
Εικόνα 5.19: Pre-Processing φάσεις αλγορίθμου ανά αριθμό Reducers - Node 2

Στην συνέχεια, ομοίως για την κύρια διαδικασία εκτέλεσης του αλγορίθμου με επιπλέον παράμετρο την χρήση 2 κόμβων, λαμβάνουμε τους παρακάτω χρόνους:

Φάσεις εκτέλεσης αλγορίθμου ανά αριθμό reducers **Nodes =2**	**Χρόνος εκτέλεσης (Sec)**
RON1 (Round One NodeIterator (Reducers=1)	89
RON2 (Round One NodeIterator (Reducers=2)	19
RON3 (Round One NodeIterator (Reducers=3)	15
RON4 (Round One NodeIterator (Reducers=4)	12
RTN1 = Round Two NodeIterator (Reducers=1)	753
RTN2 = Round Two NodeIterator (Reducers=2)	25
RTN3 = Round Two NodeIterator (Reducers=3)	17
RTN4 = Round Two NodeIterator (Reducers=4)	15
TTC1 = Total Triangles Counter (Reducers=1)	147
TTC2 = Total Triangles Counter (Reducers=2)	19
TTC3 = Total Triangles Counter (Reducers=3)	16
TTC4 = Total Triangles Counter (Reducers=4)	16

Πίνακας 5.20: Φάσεις εκτέλεσης αλγορίθμου ανά αριθμό reducers- Node 2

Όπως και σε όλες τις μετρήσεις, ακολουθεί η γραφική αποτύπωση των αποτελεσμάτων:



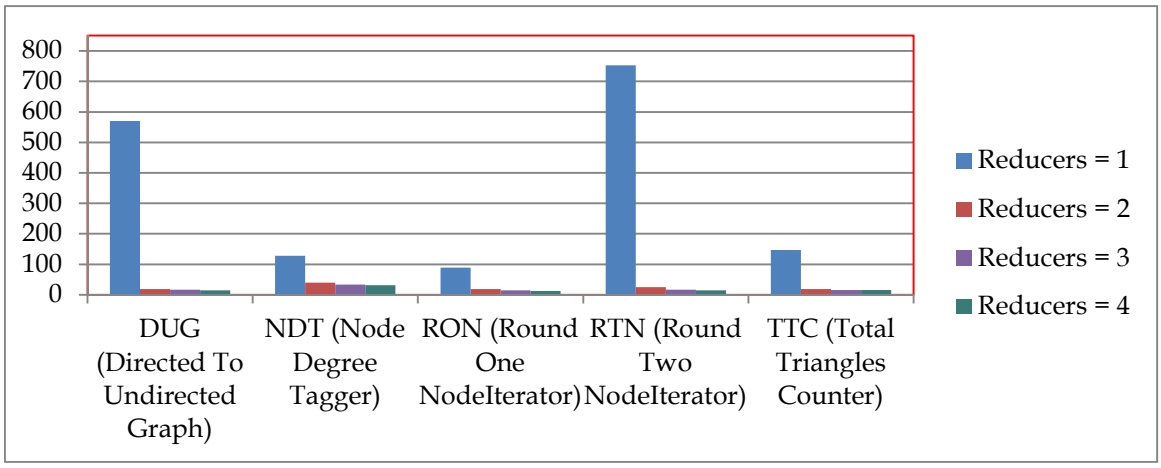
Εικόνα 5.21: Φάσεις εκτέλεσης αλγορίθμου ανά αριθμό reducers- Node 2

Τα συγκριτικά αποτελέσματα από τους παραπάνω πίνακες που συμμετείχαν στο πείραμα με τους δύο κόμβους, παίρνουμε τους παρακάτω χρόνους. Παρατηρώντας τα αποτελέσματα βλέπουμε την βελτίωση των χρόνων εκτέλεσης για όλες τις φάσεις. Η προσθήκη ενός επιπλέον (2ου) κόμβου αρχίζει να δείχνει ότι ο αλγόριθμος αρχίζει να είναι αποδοτικός σε σχέση με τον χρόνο εκτέλεσης, μετά και την προσθήκη αυτού του επιπλέον κόμβου και έχοντας κατά την φάση εκτέλεσης 6 reducers το σύστημα προς χρήση.

Συγκριτικος πίνακας εφαρμογής αλγορίθμου NodeIterator κατά αριθμό reducers στον Node 2						
Node 2	Φάσεις εκτέλεσης	Reducers = 1	Reducers = 2	Reducers = 3	Reducers = 4	Red3-Red2
Pre-Processing	DUG (Directed To Undirected Graph)	570	19	17	15	-4
	NDT (Node Degree Tagger)	128	40	34	31	-9
Αλγόριθμος NodeIterator	RON (Round One NodeIterator)	89	19	15	12	-7
	RTN (Round Two NodeIterator)	753	25	17	15	-10
	TTC (Total Triangles Counter)	147	19	16	16	-3
						*Οι μετρήσεις είναι σε sec

Πίνακας 5.22: Συγκριτικός πίνακας εφαρμογής αλγορίθμου NodeIterator κατά αριθμό reducers στον Node 2

Τα παραπάνω αποτελέσματα αποτυπώνονται γραφικά παρακάτω.



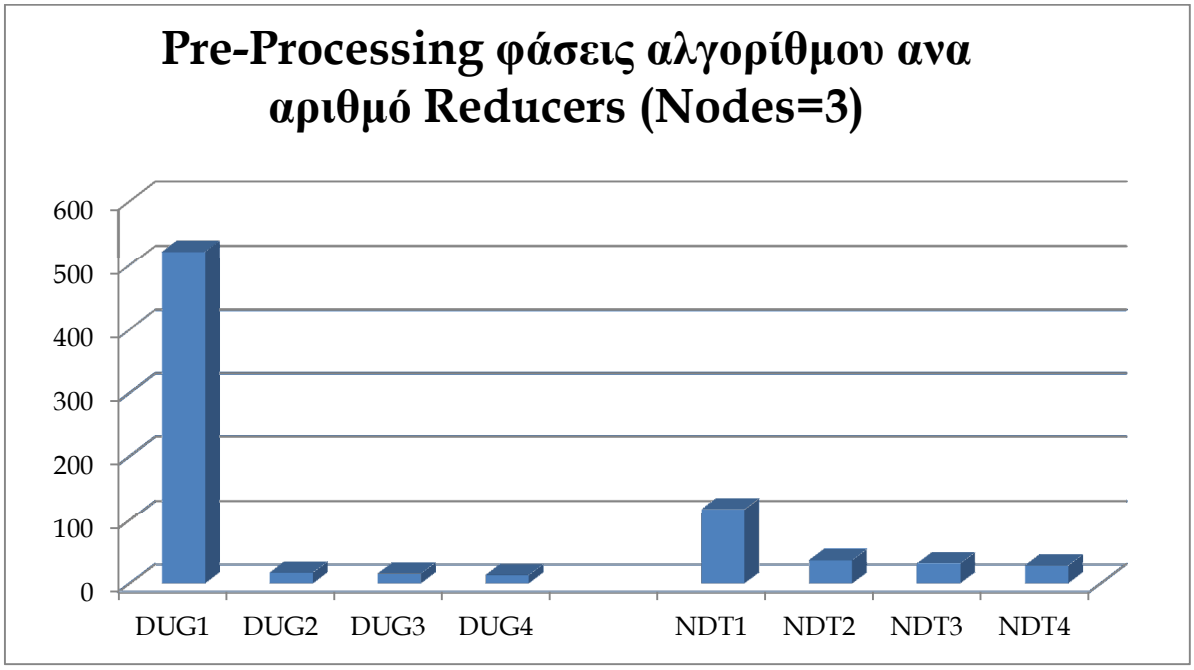
Εικόνα 5.23: Συγκριτικός πίνακας εφαρμογής αλγορίθμου NodeIterator κατά αριθμό reducers (Nodes 2)

Στην συνέχεια του πειράματος προστίθεται και ένα επιπλέον κόμβος (3ος) και ξεκινάμε την διαδικασία που τρέξαμε για τους 2 προηγούμενους κόμβους και για αυτή τη φάση. Οι χρόνοι για την pre-processing διαδικασία αποτυπώνονται στο παρακάτω πίνακα.

Pre-Processing φάσεις αλγορίθμου ανά αριθμό Reducers **Node 3**	**Χρόνος εκτέλεσης (Sec)**
TUG1 (Directed To Undirected Graph (Reducers=1))	522
TUG2 (Directed To Undirected Graph (Reducers=1))	17
TUG3 (Directed To Undirected Graph (Reducers=3))	16
TUG4 (Directed To Undirected Graph (Reducers=4))	13
NDT1 (Node Degree Tagger (Reducers=1))	118
NDT2 (Node Degree Tagger (Reducers=2))	37
NDT3 (Node Degree Tagger (Reducers=3))	32
NDT4 (Node Degree Tagger (Reducers=4))	28

Πίνακας 5.24: Pre-Processing φάσεις αλγορίθμου ανά αριθμό Reducers - Node 3

Γραφικά οι παραπάνω χρόνοι αποτυπώνονται ως:



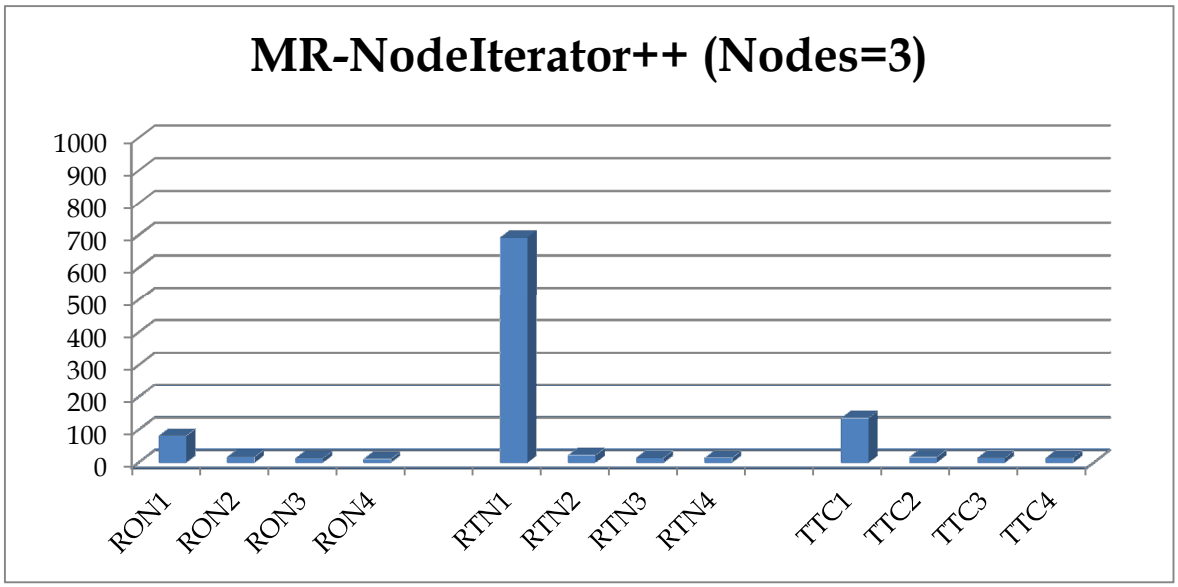
Εικόνα 5.25: Pre-Processing φάσεις αλγορίθμου ανά αριθμό Reducers - Node 3

Στην συνέχεια για την κύρια διαδικασία και για τον ίδιο όγκο δεδομένων, δίνονται οι παρακάτω χρόνοι εκτέλεσης του αλγορίθμου:

Φάσεις εκτέλεσης αλγορίθμου ανά αριθμό reducers **Nodes =3**	**Χρόνος εκτέλεσης (Sec)**
RON1 (Round One NodeIterator (Reducers=1)	82
RON2 (Round One NodeIterator (Reducers=2)	17
RON3 (Round One NodeIterator (Reducers=3)	13
RON4 (Round One NodeIterator (Reducers=4)	11
RTN1 = Round Two NodeIterator (Reducers=1)	693
RTN2 = Round Two NodeIterator (Reducers=2)	23
RTN3 = Round Two NodeIterator (Reducers=3)	16
RTN4 = Round Two NodeIterator (Reducers=4)	14
TTC1 = Total Triangles Counter (Reducers=1)	136
TTC2 = Total Triangles Counter (Reducers=2)	17
TTC3 = Total Triangles Counter (Reducers=3)	15
TTC4 = Total Triangles Counter (Reducers=4)	15

Πίνακας 5.26: Φάσεις εκτέλεσης αλγορίθμου ανά αριθμό reducers- Node 3

Γραφικά αποτυπώνεται ως:



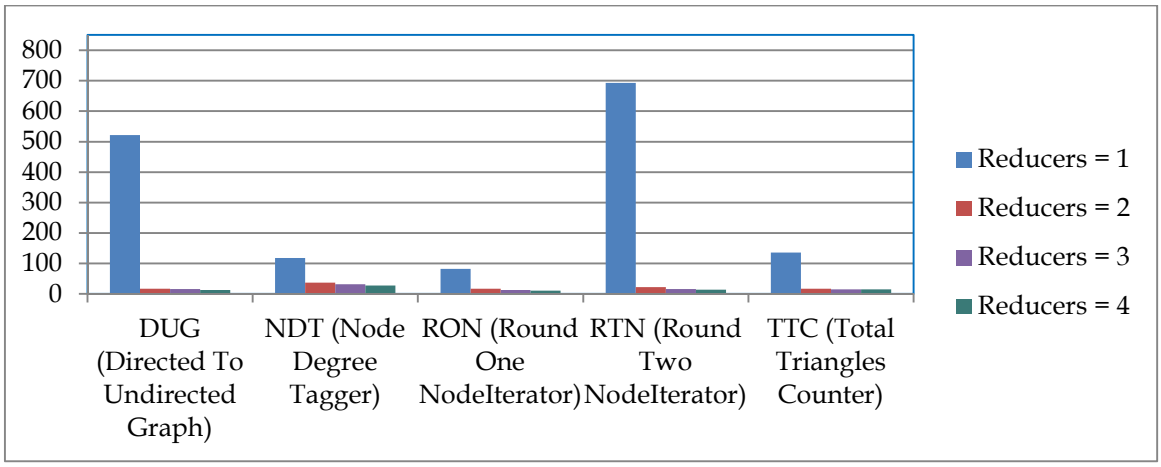
Εικόνα 5.27: Φάσεις εκτέλεσης αλγορίθμου ανά αριθμό reducers- Node 3

Τα συγκριτικά αποτελέσματα για τους χρόνους εκτέλεσης στο πείραμα με 3 κόμβους , δίνονται στο παρακάτω πίνακα. Αυτό που παρατηρούμε είναι σταδιακή μείωση του χρόνου εκτέλεσης του αλγορίθμου σε σχέση με τις προηγούμενες φάσεις. Αρχίζει λοιπόν η συστοιχία των κόμβων που συμμετέχουν να γίνεται αποδοτικότερη. Σε αυτή την φάση συμμετέχουν 3 κόμβοι με συνολικό αριθμό 6 reducers.

Συγκριτικος πίνακας εφαρμογής αλγορίθμου NodeIterator κατά αριθμό reducers στον Node 3						
Node 3	Φάσεις εκτέλεσης	Reducers = 1	Reducers = 2	Reducers = 3	Reducers = 4	Red3-Red2
Pre-Processing	DUG (Directed To Undirected Graph)	522	17	16	13	-4
	NDT (Node Degree Tagger)	118	37	32	28	-9
Αλγόριθμος NodeIterator	RON (Round One NodeIterator)	82	17	13	11	-6
	RTN (Round Two NodeIterator)	693	23	16	14	-9
	TTC (Total Triangles Counter)	136	17	15	15	-2
						*Οι μετρήσεις είναι σε sec

Πίνακας 5.28: Συγκριτικός πίνακας εφαρμογής αλγορίθμου NodeIterator κατά αριθμό reducers στον Node 3

Στην γραφική παράσταση παρατηρούμε την μεγάλη χρονική καθυστέρηση που έχει ο αλγόριθμος με παράμετρο τον έναν reducer:

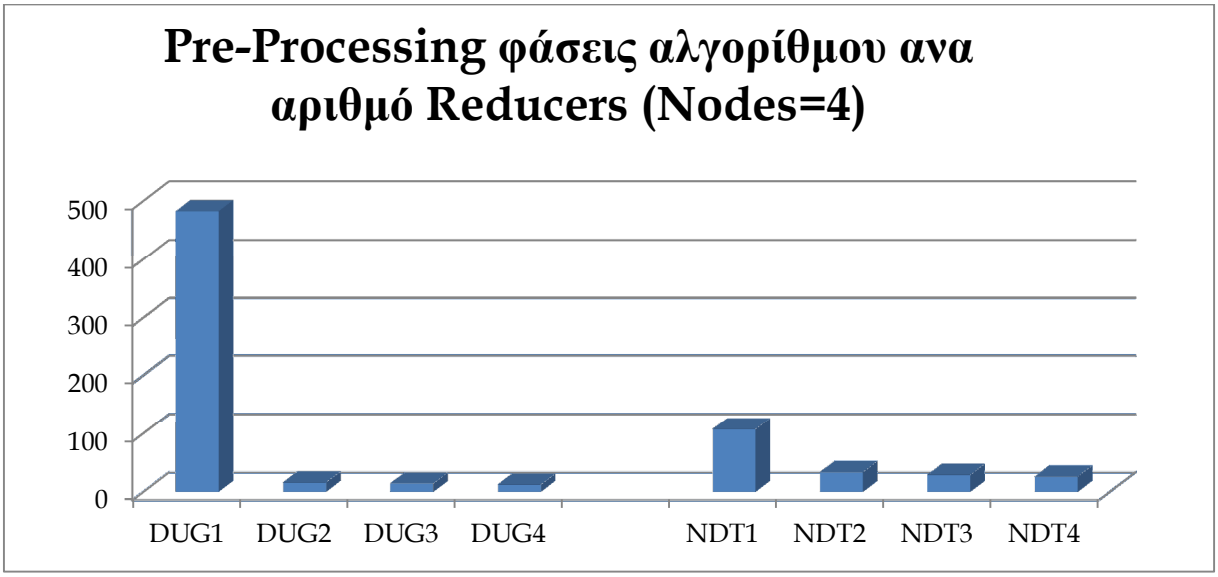


Εικόνα 5.29: Συγκριτικός πίνακας εφαρμογής αλγορίθμου NodeIterator κατά αριθμό reducers (Node 3)

Τέλος, αξιοποιώντας και τον 4ο κόμβο που διέθετε η συστοιχία που μας παραχωρήθηκε, επαναλήφθηκε η διαδικασία, όπου πήραμε τα παρακάτω χρονικά αποτελέσματα, τόσο την pre-processing φάση:

Pre-Processing φάσεις αλγορίθμου ανά αριθμό Reducers **Nodes 4**	**Χρόνος εκτέλεσης (Sec)**
TUG1 (Directed To Undirected Graph (Reducers=1))	482
TUG2 (Directed To Undirected Graph (Reducers=1))	16
TUG3 (Directed To Undirected Graph (Reducers=3))	14
TUG4 (Directed To Undirected Graph (Reducers=4))	12
NDT1 (Node Degree Tagger (Reducers=1))	108
NDT2 (Node Degree Tagger (Reducers=2))	34
NDT3 (Node Degree Tagger (Reducers=3))	29
NDT4 (Node Degree Tagger (Reducers=4))	26

Πίνακας 5.30: Pre-Processing φάσεις αλγορίθμου ανά αριθμό Reducers - Node 4



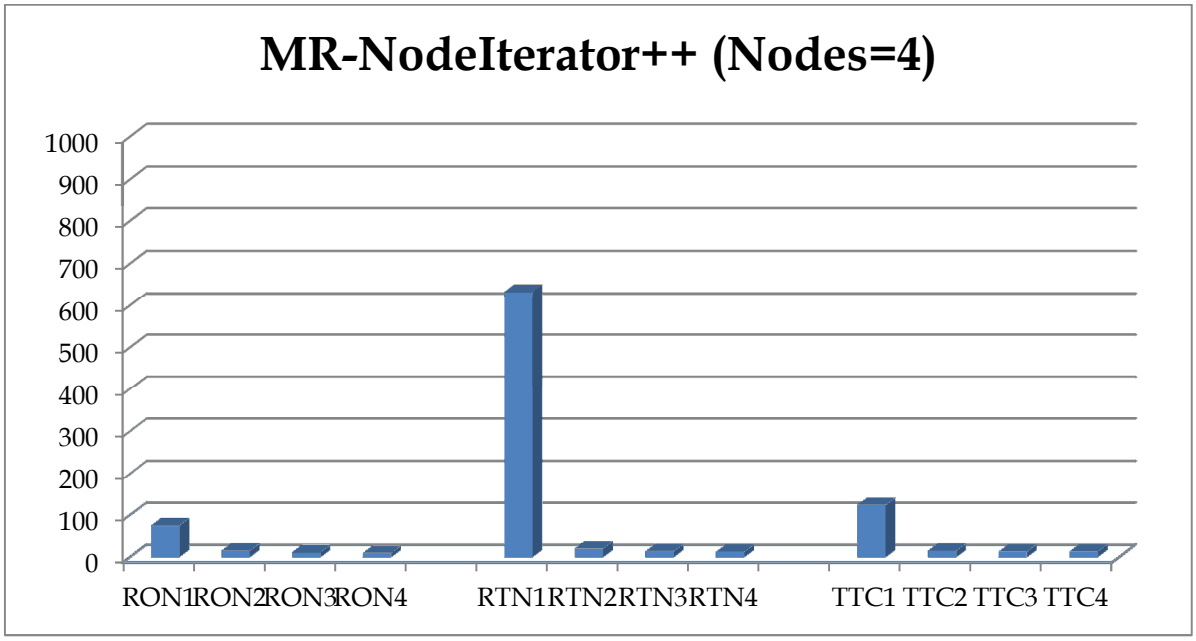
Εικόνα 5.31: Pre-Processing φάσεις αλγορίθμου ανά αριθμό Reducers - Node 4

Όσο και για την κύρια διαδικασία.

Φάσεις εκτέλεσης αλγορίθμου ανά αριθμό reducers **Nodes =4**	**Χρόνος εκτέλεσης (Sec)**
RON1 (Round One NodeIterator (Reducers=1)	75
RON2 (Round One NodeIterator (Reducers=2)	16
RON3 (Round One NodeIterator (Reducers=3)	12
RON4 (Round One NodeIterator (Reducers=4)	10
RTN1 = Round Two NodeIterator (Reducers=1)	634
RTN2 = Round Two NodeIterator (Reducers=2)	21
RTN3 = Round Two NodeIterator (Reducers=3)	14
RTN4 = Round Two NodeIterator (Reducers=4)	13
TTC1 = Total Triangles Counter (Reducers=1)	124
TTC2 = Total Triangles Counter (Reducers=2)	16
TTC3 = Total Triangles Counter (Reducers=3)	13
TTC4 = Total Triangles Counter (Reducers=4)	13

Πίνακας 5.32: Φάσεις εκτέλεσης αλγορίθμου ανά αριθμό reducers- Node 4

Γραφικά οι χρόνοι δίνονται από την γραφική παράσταση:



Εικόνα 5.33: Φάσεις εκτέλεσης αλγορίθμου ανά αριθμό reducers- Node 4

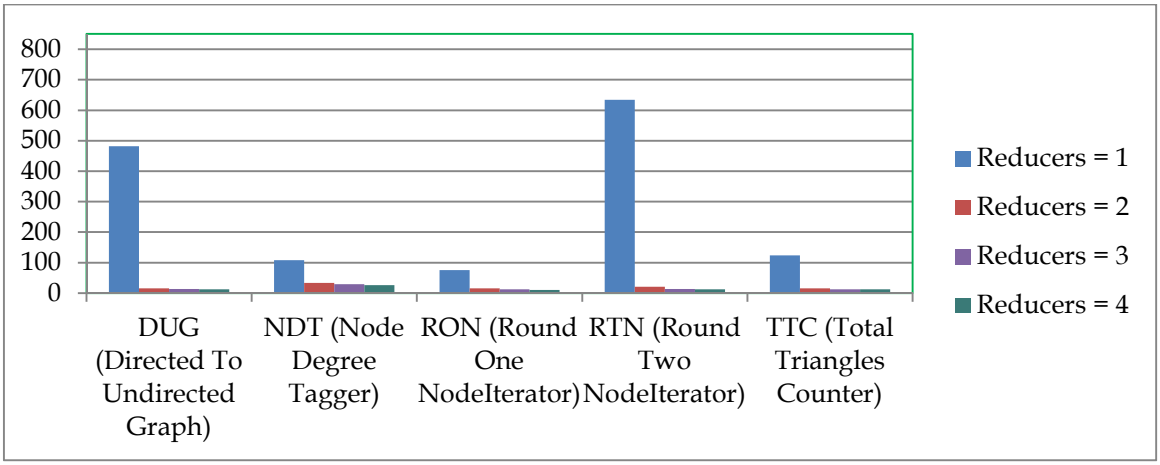
Αυτό που παρατηρούμε προσθέτοντας και τον 4ο κόμβο, είναι ότι τα αποτελέσματα επιβεβαιώνουν την μείωση του χρόνου εκτέλεσης του αλγορίθμου, όπως δίνεται και στον συγκριτικό πίνακα αποτελεσμάτων.

Μια παρατήρηση που ισχύει και για τις φάσεις εκτέλεσης του πειράματος για 2 reducers και άνω είναι ότι για την παράμετρο με έναν reducer, ο χρόνος εκτέλεσης παραμένει μεγάλος, με σχετικά μικρή βελτίωση από κάθε προσθήκη κόμβου. Για να έχουμε ικανοποιητικούς χρόνους απόδοσης από την εκτέλεση του αλγορίθμου είναι ότι θα πρέπει να τρέχει με παράμετρο άνω των 2 reducers.

Συγκριτικος πίνακας εφαρμογής αλγορίθμου NodeIterator κατά αριθμό reducers στον Node 4						
Node 4	Φάσεις εκτέλεσης	Reducers = 1	Reducers = 2	Reducers = 3	Reducers = 4	Red3-Red2
Pre-Processing	DUG (Directed To Undirected Graph)	482	16	14	12	-4
	NDT (Node Degree Tagger)	108	34	29	26	-8
Αλγόριθμος NodeIterator	RON (Round One NodeIterator)	75	16	12	10	-6
	RTN (Round Two NodeIterator)	634	21	14	13	-8
	TTC (Total Triangles Counter)	124	16	13	13	-3

Πίνακας 5.34: Συγκριτικός πίνακας εφαρμογής αλγορίθμου NodeIterator κατά αριθμό reducers στον Node 4

Παρατηρώντας και την τελευταία γραφική παράσταση, παρατηρούμε ότι η χρήση ενός reducer δεν είναι αποδοτική για τον αλγόριθμο και το πείραμα θα πρέπει να συνεχισθεί, αγνοώντας την παράμετρο για έναν reducer.



Εικόνα 5.35: Συγκριτικός πίνακας εφαρμογής αλγορίθμου NodeIterator κατά αριθμό reducers (Node 4)

Συμπερασματικά, θεωρητικά η προσθήκη περισσοτέρων κόμβων θα μείωνε περισσότερο τον χρόνο εκτέλεσης του αλγορίθμου. Δυστυχώς το περιβάλλον των πειραμάτων μας είχε μόνο τέσσερις κόμβους. Συνεχίζοντας την θεωρητική προσέγγιση η συνεχής προσθήκη κόμβων θα έχει ένα άνω όριο διότι η συμμετοχή ενός μεγάλου αριθμού κόμβων, άνω των 50, θα δημιουργούσε προβλήματα συντονισμού και επικοινωνιακού κόστους που θα απορροφούσε τα τυχόν οφέλη της επιπλέον υπολογιστικής ισχύος.

Κεφάλαιο 6
Συμπεράσματα – Επίλογος

Για την εκπόνηση της παρούσας μεταπτυχιακής διατριβής αρχικά οριοθετήθηκαν τα κίνητρα, ο στόχος και η συγκεκριμένη μεθοδολογία που θα ακολουθούσε. Η συνέχεια αφορούσε στην θεωρητική διερεύνηση του θέματος, καθώς το αντικείμενο απαιτούσε την πλήρη διερεύνηση της θεωρίας της μοντελοποίησης των ιδιοτήτων των κοινωνικών δικτύων και την αναπαράστασή τους σε μορφή γράφων. Η μετάβαση προς μία συστηματική μελέτη των κοινωνικών δικτύων με χρήση της υπολογιστικής ισχύος, κινείται ταυτόχρονα στην αιχμή της καινοτομίας της επιστήμης των υπολογιστών και στο μεταίχμιο των κοινωνικών επιστημών. Η δε χρησιμότητά της είναι μεγάλη σε θεωρητική ανάλυση κοινωνιολογικού χαρακτήρα, όσο και σε πρακτικές εφαρμογές στην οικονομία, την ανάλυση αγοράς, την προβολή στα νέα κοινωνικά μέσα (Social Media). Έτσι, μελετήθηκε η σχετική βιβλιογραφία και παρουσιάστηκαν οι σχετικές με τα κοινωνικά δίκτυα και την αναπαράστασή τους εργασίες.

Στη συνέχεια παρουσιάστηκαν εκτενώς τα τεχνολογικά εκείνα εργαλεία που δίνουν την δυνατότητα της ανάλυσης των παραπάνω υποθέσεων με συστηματικό και επιστημονικά

αποδεκτό τρόπο. Παρουσιάστηκαν πέρα από τα προγραμματιστικά εργαλεία MapReduce και Hadoop, εργαλεία απεικόνισης των κοινωνικών δικτύων μέσω των λειτουργιών της οπτικής απεικόνισης και στατιστικής ανάλυσης (Pegasus, Gephi). Τα αποτελέσματα της εφαρμογής αυτών των εργαλείων σε πραγματικά σύνολα δεδομένων – και εδώ έγκειται μία ουσιαστική συνεισφορά της παρούσας διατριβής – δείχνουν απτά τις δυνατότητες και τα όρια της χρησιμότητας των σημερινών εργαλείων απεικόνισης.

Τα σύνολα δεδομένων που χρησιμοποιήθηκαν, Twitter και BerkStan, είναι πραγματικά δεδομένα με εγγενή τα χαρακτηριστικά των λεγόμενων «φυσικών γράφων». Συνεπώς, παρουσιάζουν την επιπλέον πρόκληση ζητημάτων όπως η κλίση των δεδομένων προς μία κατανομή εκθετικού χαρακτήρα ("power law"). Αυτή η πραγματικότητα ωθεί στα όρια τις δυνατότητες του τεχνολογικού εξοπλισμού και, κατά συνέπεια, μας επιστρέφει στην ανάγκη ύπαρξης αλγορίθμων αρκετά έξυπνων ώστε να μειώνουν το χρόνο εκτέλεσης των υπολογισμών.

Η δεύτερη, λοιπόν, και σημαντικότερη συνεισφορά της εργασίας είναι η εφαρμογή τέτοιων αλγορίθμων στην πράξη και η επιβεβαίωση των θεωρητικών τους αποτελεσμάτων σε πραγματικές συνθήκες. Σε αντίθεση με ασκήσεις επί χάρτου, η πραγματικότητα της αγοράς επιβάλλει πραγματικές λύσεις με αποδεκτό κόστος και ο κώδικας που συνοδεύει την παρούσα διατριβή έχει αντιμετωπίσει όλα τα αναμενόμενα και μη προβλήματα στη διερεύνηση γράφων στο επίπεδο της τριγωνικής κοινότητας.

Τα πειραματικά αποτελέσματα οδηγούν στο συμπέρασμα ότι όχι μόνο η παρούσα εργασία έχει νόημα, αλλά και ότι η μεγαλύτερη εκμετάλλευση της παραλληλοποίησης στην υπολογιστική διαδικασία αντιμετώπισης γράφων είναι ένα αντικείμενο που αξίζει να μελετηθεί περεταίρω, και αυτό είναι κάτι που διαφαίνεται από τις ερευνητικές τάσεις της εποχής.

Τελικά, ο στόχος της εύρεσης κοινωνικών ομάδων σε δίκτυα κοινωνικής δικτύωσης επιτεύχθηκε και, στα πλαίσια των χρονικών και θεματικών περιορισμών που θέτει μια μεταπτυχιακή διατριβή, παρουσιάσθηκαν και αναλύθηκαν οι συγκεκριμένες τεχνικές όπως και υλοποιήθηκε ο αλγόριθμος MR-NodeIterator++ σε περιβάλλον Hadoop. Τα πειράματα που έγιναν πάνω σε αυτόν, επιβεβαίωσαν με ανεξάρτητο τρόπο τα ερευνητικά αποτελέσματα άλλων ερευνητών που ασχολούνται με αυτές τις τεχνολογίες, δηλαδή ότι ο συγκεκριμένος αλγόριθμος είναι μια αποδοτική τεχνική εύρεσης τριγώνων (σχέσεις φιλίας σε κοινωνικά δίκτυα).

Ως μελλοντική εργασία βασιζόμενη στην παρούσα μεταπτυχιακή διατριβή, προτείνεται η χρήση cloud μεγαλυτέρου μεγέθους ώστε να αναζητηθεί το όριο των δυνατοτήτων του αλγόριθμου που υλοποιήθηκε. Επίσης, σε αυτήν την περίπτωση θα ήταν δυνατό να εξεταστούν και νέα πολύ μεγαλύτερα σετ δεδομένων. Η πειραματική ανάλυση θα συσχετίσει εγγενείς διαφορές ανάμεσα σε σετ δεδομένων με διαφορετικά αποτελέσματα δίνοντας έτσι, τρόπον τινά, μία σχέση αιτίου αιτιατού.

Πρέπει ωστόσο να σημειωθεί, ότι παρά το γεγονός ότι η παρούσα μεταπτυχιακή διατριβή δεν υπερέβη το χρονικό διάστημα ενός έτους, έγιναν ήδη σημαντικές αλλαγές όσον αφορά τις τεχνολογίες που έχουν περιγραφεί παραπάνω, κάτι που επιβεβαιώνει το γεγονός ότι η θεματολογία της κινείται στην αιχμή της εξέλιξης. Συνεπώς, οι ραγδαίες αλλαγές των τεχνολογιών αυτών, αλλά και ο τρόπος ενσωμάτωσής τους στα παραγωγικά συστήματα γεννούν συνεχώς διάφορα επιστημονικά ερωτήματα προς διερεύνηση σε μελλοντικούς ερευνητές στο συγκεκριμένο γνωστικό πεδίο.

Βιβλιογραφία

[01] Alejandro Portes. Social capital: Its origins and applications in modern sociology. Annual Review of Sociology, 24:1–24, 1998.

[02] Anand Rajaraman, Jeffrey D. Ullman, "Mining of Massive Datasets", 2011

[03] Audrey Watters , "How Recent Changes to Twitter's Terms of Service Might Hurt Academic Research" March 3, 2011 05:56 PM, available at: http://www.readwriteweb.com/archives/how_recent_changes_to_twitters_terms _of_service_mi.php, Ημερομηνία τελευταίας πρόσβασης: 10/2/2012

[04] Baumes, J., Goldberg, M., Krishnamoorhy, M., Magdon-Ismail, M and N. Preston (2005) 'Finding Communities by Dustering a Graph into Overlapping Subgraphs, Proceedings of International Conference on Applied Computing (IADIS 05) Algrave, Portugal, 27-36

[05] Berkstan data set, "http://snap.stanford.edu/data/web-BerkStan.html", Ημερομηνία τελευταίας πρόσβασης: 10/4/2012

[06] Charalampos E. Tsourakakis, U Kang, Gary L. Miller, and Christos Faloutsos. Doulion: Counting triangles in massive graphs with a coin. In Knowledge Discovery and Data Mining (KDD), 2009,

[07] D. J. Watts and Steven Strogatz (June 1998). "Collective dynamics of 'small-world' networks". Nature 393 (6684): 440–442. Bibcode 1998Natur.393..440W. doi:10.1038/30918. PMID 9623998.

[08] Dangalchev Ch., Residual Closeness in Networks, Phisica A 365, 556 (2006)

[09] Don Coppersmith and Ravi Kumar. An improved data stream algorithm for frequency moments. In Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, SODA '04, pages 151–156, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[10] Foto Afrati, Dimitris Fotakis & Jeffrey D. Ullman , "Enumerating Subgraph Instances Using Map-Reduce", 8 Apr 2012 στο: http://arxiv.org/pdf/1204.1754.pdf

[11] Freeman, Linton (1977). "A set of measures of centrality based upon betweenness". Sociometry 40: 35–41.

[12] Hadoop MapReduce Tutorial, <http://hadoop.apache.org/common/docs/current /mapred_tutorial.html>, Ημερομηνία τελευταίας πρόσβασης: 10/3/2012

[13] Haewoon Kwak, Changhyun Lee, Hosung Park and Sue Moon, "What is Twitter, a Social Network or a News Media?", at http://an.kaist.ac.kr/traces/WWW2010.html

[14] HubSpot, State of the Twittersphere, June 2009, "http://blog.hubspot.com/Portals/249/ sotwitter09.pdf", Ημερομηνία τελευταίας πρόσβασης: 10/4/2012

[15] James S. Coleman. Social capital in the creation of human capital. American Journal of Sociology, 94:S95–S120, 1988.

[16] Kernighan B.W. and S. Li (1970) 'An Efficient Hueristic procedure for Partitioning Graphs' Bell System Technical Journal, 49, 291-307

[17] Leskovec, J. Lang K.J., Dasgupta, A., and M.W. Mahoney (2009) Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. Internet Mathematics 6(1) 29--123

[18] Leskovec, J., Lang, K.J., Dasgupta, A., and M.W. Mahoney (2008) 'Community Structure in Large Networks: Natural Cluster Sizes and the Absense of Large Well-Defined Clusters' Computing Research Repository, CORP

[19] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '08, pages 16–24, New York, NY, USA, 2008. ACM.

[20] Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, PODS '06, pages 253–262, New York, NY, USA, 2006. ACM.

[21] Michael Noll., Hadoop tutorial, « http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/#running-a-mapreduce-job", Ημερομηνία τελευταίας πρόσβασης: 10/4/2012

[22] Newman M. E. J. and M. Girvan (2004) 'Trading and Evaluating Community Structure in Networks', Physical Review E, 69, 026113

[23] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. SIAM J. Comput., 14(1):210–ˆa˘A¸S223, 1985.

[24] Palla, G., Imre, D., Farkas, I and T. Vicsek (2005) 'Uncovering the Overlapping Community Structure of Complex Networks in Nature and Society' Nature, 435 (7043), 814-818

[25] Pothen, A. Horst, D.S. and K. P. Liou (1990) 'Partitioning Sparse Matrices with Eigenvectors of Graphs' Society for Industrial and Applied Mathematics (SIAM) Journal of Matrix Analysis and Application, 11(3), 430-453

[26] Presentation from Nikos Stasinopoulos for "Siddharth Suri & Sergei Vassilvitskii, «Counting Triangles and the Curse of the Last Reducer», Yahoo! Research, April 2011"

[27] Ronald S. Burt. Structural holes and good ideas, American Journal of Sociology, 110(2):349–99, September 2004.

[28] Siddharth Suri & Sergei Vassilvitskii, «Counting Triangles and the Curse of the Last Reducer», Yahoo! Research, April 2011

[29] Stanley Wasserman, Kathrine Faust, 1994. Social Network Analysis: Methods and Applications. Cambridge: Cambridge University Press.

[30] Thomas Schank. Algorithmic Aspects of Triangle-Based Network Analysis. PhD thesis, Universit¨at Karlsruhe (TH), 2007.

[31] Tom White, «Hadoop: The Definitive Guide», O'Reilly, 2nd Edition, October 2010

[32] Tore Opsahl (2009). "Clustering in Two-mode Networks". Conference and Workshop on Two-Mode Social Analysis (Sept 30-Oct 2, 2009).

[33] Tore Opsahl and Pietro Panzarasa (2009). "Clustering in Weighted Networks". Social Networks 31 (2): 155–163. doi:10.1016/j.socnet.2009.02.002.

[34] U Kang, Charalampos E. Tsourakakis, and Christos Faloutsos, PEGASUS: A Peta-Scale Graph Mining System - Implementation and Observations. IEEE International Conference on Data Mining (ICDM) 2009, Miami, Florida, USA

[35] Wiki Twitter, <http://en.wikipedia.org/wiki/Twitter>, Ημερομηνία τελευταίας πρόσβασης: 10/3/2012

[36] Duen Horng "Polo" Chau, Aniket Kittur, Jason I. Hong & Christos Faloutsos, «Supporting sensemaking in large network data», School of Computer Science - Carnegie Mellon University, April 2012

[37] Slides by Matthew Damigos, "Google News Personalization: Scalable Online Collaborative Filtering", Abhinandan Das, Mayur Datar, Ashutosh Garg, WWW2007, May 8-12, 2007, Banff, Alberta, Canada

[38] Ζήνων Ζένιου, «Υλοποίηση και Πειραματική Αξιολόγηση Αλγορίθμων για ανακάλυψη γνώσης σε κοινωνικά Δίκτυα», Μεταπτυχιακή Διατριβή, Πανεπιστήμιο Κύπρου, 2011

Παράρτημα

Java classes

Triangles MR-NodeIterator++

nodedegreestagger.java

```
package triangles.nodeiterator;

import triangles.preprocessing.SecondNodeTagger;

/**
 * Map Reduce job counting degree for each node in the graph.<br>
 * Linear cost is analogous to size of the edge list. Ω(|E|)
 *
 * @author Vassilis Trizonis
 * All right reserved
 */

public class NodeDegreesTagger extends Configured implements
            Tool {

    /**
     * The mapper class, outputs node occurrences in edge list.
     */
    public static class EdgeCounterMapper extends
                    Mapper<Object, Text, IntWritable, IntWritable> {

            private final static IntWritable zero = new IntWritable(0);

            @Override
            public void map(Object key, Text value, Context context)
                            throws IOException, InterruptedException {

                    IntWritable keyOut = new IntWritable();
                    IntWritable valOut = new IntWritable();
```

```
                    keyOut.set(Integer.parseInt(key.toString()));
                    StringTokenizer valueIterator = new StringTokenizer(
                                    value.toString());
                    while (valueIterator.hasMoreTokens()) {
                            String val = valueIterator.nextToken();
                            valOut.set(Integer.parseInt(val));
                            // Map output (node1,node2)
                            context.write(keyOut, valOut);
                            // Map output (node1, zero)
                            // used to sum for degree in reducer step
                            context.write(keyOut, zero);
                    }
            }
    }

    /**
     * Reducer counting zero occurrences
     */
    public static class EdgeCounterReducer extends
                    Reducer<IntWritable, IntWritable, Text, Text> {

            /**
             * Reduce function performs on (node1, node2) or (node1, zero) map
             * output pairs. <br>
             * It stores in memory (hopefully no more than default split size) node
             * 2 values, calculates degree by summing zeros.<br>
             * Output is a reversed node list in the form of (node2,
             * node1[tab]degree) (IntWritable, Text)
             */
            public void reduce(IntWritable key, Iterable<IntWritable> values,
                            Context context) throws IOException, InterruptedException {
                    int degree = 0;
                    // a buffer that collects node 2 from (node 1, node2)
                    List<Text> buffer = new ArrayList<Text>();
                    for (IntWritable val : values) {
                            if (val.get() == 0) {
                                    degree++;
                            } else
                                    buffer.add(new Text(Integer.toString(val.get())));

                    }
                    // Tag node 1 with its degree i.e. node1[tab]degree
                    Text taggedKey = new Text(key.toString() + "\t"
                                    + Integer.toString(degree));
                    for (Text b : buffer) {
                            context.write(b, taggedKey);
                    }
                    context.write(new Text(key.toString()),
                                    new Text("0\t" + Integer.toString(degree)));
            }
    }

    /**
     * Contains the Configuration properties for the job
     *
     * @throws Exception
     */
    public int run(String[] args) throws Exception {

            Path edgeListPath = new Path(args[0]);
            Path outputPath = new Path("/user/vassilis/temppath/");
//          Path outputPath = new Path(args[1]);
            int numReducers = Integer.parseInt(args[2].trim());

            Configuration conf = getConf();
            // conf.set("mapred.tasktracker.map.tasks.maximum", "4");
            // conf.set("mapred.tasktracker.reduce.tasks.maximum", "4");
            conf.set("mapred.child.java.opts", "-Xmx512m");
            @SuppressWarnings("deprecation")
            Job job = new Job(conf, "Calculate node degree and tag first node");
            job.setJarByClass(NodeDegreesTagger.class);
            job.setNumReduceTasks(numReducers);

            job.setMapperClass(EdgeCounterMapper.class);
            job.setReducerClass(EdgeCounterReducer.class);

            // job.setPartitionerClass(CustomPartitioner.class);
            job.setOutputKeyClass(IntWritable.class);
            job.setOutputValueClass(IntWritable.class);
            // customized Input Format : node1 \t node2
            job.setInputFormatClass(triangles.utils.EdgeInputFormat.class);
            FileInputFormat.addInputPath(job, edgeListPath);
            FileOutputFormat.setOutputPath(job, outputPath);

            return job.waitForCompletion(true) ? 0 : 1;
    }

    /**
     * Loads the driver for the job
     *
```

```
     * @param args
     *            Command line arguments <br>
     *            Usage: EdgeListFolder outputFolder # of reducers
     * @throws Exception
     */
    public static void main(String[] args) throws Exception {

            // Verify arguments
            if (args.length != 3) {
                    System.err
                                    .println("Usage: <EdgeListFolder> <outputFolder> <# of
reducers>");
                    System.exit(1);
            }

            // int res = ToolRunner.run(new Configuration(), new
            // FirstNodeTaggerWithDegreeCounter(),
            // args);
            // System.exit(res);


            //Start counting time
        double starttime = (double) System.currentTimeMillis();


            int res = ToolRunner.run(new Configuration(),
                            new NodeDegreesTagger(), args);
            if (res == 0) {
                    res = ToolRunner.run(new Configuration(), new SecondNodeTagger(),
                                    args);
            }


            //delete temp path
            if (res == 0) {
                    Configuration config = new Configuration();
                      FileSystem hdfs = FileSystem.get(config);
                      Path path =  new Path("/user/vassilis/temppath/");
                      boolean isDeleted = hdfs.delete(path, true);
            }

            //Stop counting time
        double endtime = (double) System.currentTimeMillis();

        //Initialize the file you print the time results
        File timeFile = new File("/home/hduser/hadoop/time_triangles.txt");
        //Initialize the Output and Print streams
        FileOutputStream out;
        PrintStream p ;
        out = new FileOutputStream(timeFile, true);
        p = new PrintStream(out);
        Date now = new Date();
        p.append(now + " Elapsed Time - Node DegreesTagger: " + ((endtime - starttime)) + "\r\n");
        p.close();
        out.close();

            System.exit(res);

    }
}
```

roundonenodeiterator.java

```
package triangles.nodeiterator;

import triangles.preprocessing.LongWritablePair;
import triangles.utils.OrderByDegree;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.List;
import java.util.Date;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
```

```java
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

/**
 * @author Vassilis Trizonis
 * All right reserved
 *
 */
public class RoundOneNodeIterator extends Configured implements Tool {
	/**
	 * The first part of the MR-NodeIterator++ Algorithm, which calculates
	 * possible 2-paths (ie 3 nodes that may form a triangle on the graph) Reads
	 * from an edge list enhanced by the degrees of the nodes, as calculated
	 * before.
	 *
	 *
	 */

	public static class DegreeComparatorMapper extends
			Mapper<Object, Text, LongWritable, LongWritable> {
		@Override
		public void map(Object key, Text value, Context context)
				throws IOException, InterruptedException {

			OrderByDegree order = new OrderByDegree();
			order.parse(value);
			if (order.compareDegrees()) {
				context.write(new LongWritable(order.getNodeTwo()),
						new LongWritable(order.getNodeOne()));
			}
		}
	}

	/**
	 * @author Vassilis Trizonis
	 * All right reserved
	 */
	public static class PossiblePathReducer extends
			Reducer<LongWritable, LongWritable, LongWritable, LongWritablePair> {
		@Override
		public void reduce(LongWritable key, Iterable<LongWritable> values,
				Context context) throws IOException, InterruptedException {

			List<LongWritable> buffer = new ArrayList<LongWritable>();
			List<LongWritable> buffer2 = new ArrayList<LongWritable>();
			for (LongWritable val : values) {
				buffer.add(new LongWritable(val.get()));
				buffer2.add(new LongWritable(val.get()));
			}

			for (LongWritable b1 : buffer) {
				for (LongWritable b2 : buffer2) {
					if (b1.get() < b2.get()) {
						context.write(key, new LongWritablePair(b1, b2));
					}
				}
			}
		}
	}

	public int run(String[] args) throws Exception {

		Configuration conf = getConf();
		@SuppressWarnings("deprecation")
		Job job = new Job(conf);
		job.setJarByClass(RoundOneNodeIterator.class);
		job.setJobName("Discover Possible Triangles");

		job.setMapperClass(DegreeComparatorMapper.class);
		job.setReducerClass(PossiblePathReducer.class);

		FileInputFormat.addInputPath(job, new Path(args[0]));
		FileOutputFormat.setOutputPath(job, new Path(args[1]));

		job.setOutputKeyClass(LongWritable.class);
		job.setOutputValueClass(LongWritable.class);

		job.setNumReduceTasks(Integer.parseInt(args[2].trim()));

		return job.waitForCompletion(true) ? 0 : 1;

	}

	/**
```

```
	 * Loads the Driver
	 *
	 *
	 * @param args
	 * @throws Exception
	 */
	public static void main(String[] args) throws Exception {

		// Verify arguments
		if (args.length != 3) {
		System.err.println("Usage: <taggedEdgeListFolder> <outputFolder> <# of reducers>");
		System.exit(1);
		}

		//Start counting time
	  double starttime = (double) System.currentTimeMillis();

		int res = ToolRunner.run(new Configuration(), new RoundOneNodeIterator(),
				args);

		//Stop counting time
	  double endtime = (double) System.currentTimeMillis();

	  //Initialize the file you print the time results
	  File timeFile = new File("/home/hduser/hadoop/time_triangles.txt");
	  //Initialize the Output and Print streams
	  FileOutputStream out;
	  PrintStream p ;
	  out = new FileOutputStream(timeFile, true);
	  p = new PrintStream(out);
	  Date now = new Date();
	  p.append(now + " Elapsed Time - Round One NodeIterator: " + ((endtime - starttime)) + "\r\n");
	  p.close();
	  out.close();


		System.exit(res);
	}
}
```

roundtwonodeiterator.java

```
package triangles.nodeiterator;

import triangles.utils.EdgeInputFormat;
import triangles.utils.TextPair;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.List;
import java.util.Date;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Mapper.Context;
import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

/**
 * @author Vassilis Trizonis
 * All right reserved
 *
 */
public class RoundTwoNodeIterator extends Configured implements Tool {
	private static final Text zero = new Text("0");
	private static final IntWritable one = new IntWritable(1);

	/**
	 * Mapper that reads through the possible triangle list and emits.
	 *
	 * @author Vassilis Trizonis
```

```
 * All right reserved
 *
 */
public static class PossibleTriangleMapper extends
                Mapper<Text, Text, TextPair, Text> {

        @Override
        public void map(Text baseNode, Text twoPath, Context context)
                        throws IOException, InterruptedException {
                String[] twoNodes = twoPath.toString().split("\t");
                context.write(new TextPair(twoNodes[0], twoNodes[1]), baseNode);

        }
}

/**
 * Mapper that reads through the edge list and emits node pair, and zero
 *
 * @author Vassilis Trizonis
 * All right reserved
 *
 */
public static class EdgeListMapper extends
                Mapper<Text, Text, TextPair, Text> {

        public void map(Text firstNode, Text secondNode, Context context)
                        throws IOException, InterruptedException {

                context.write(new TextPair(firstNode, secondNode), zero);

        }
}

public static class CloseTriangleReducer extends
                Reducer<TextPair, Text, Text, IntWritable> {
        private boolean flag = false;

        public void reduce(TextPair twoNodes, Iterable<Text> baseNode,
                        Context context) throws IOException, InterruptedException {
                List<Text> buffer = new ArrayList<Text>();
                for (Text bN : baseNode) {
                        if (bN.equals(zero)) {
                                setFlag(true);
                        } else {
                                buffer.add(bN);
                        }
                }
                for (Text bf : buffer) {
                        if (getFlag()) {
                                context.write(bf, one);
                        }
                }
        }

        public void setFlag(boolean flag) {
                this.flag = flag;
        }

        public boolean getFlag() {
                return flag;
        }
}

public int run(String[] args) throws Exception {

        Path roundOneOutputPath= new Path(args[0]);
        Path edgeListPath = new Path(args[1]);
        Path outputPath = new Path(args[2]);
        int numReducers = Integer.parseInt(args[3].trim());

        Configuration conf = getConf();

        @SuppressWarnings("deprecation")
        Job job = new Job(conf);
        job.setJarByClass(RoundTwoNodeIterator.class);
        job.setJobName("NodeIterator Round 2");
        job.setNumReduceTasks(numReducers);


        MultipleInputs.addInputPath(job, roundOneOutputPath, EdgeInputFormat.class,
                        PossibleTriangleMapper.class);
        MultipleInputs.addInputPath(job, edgeListPath, EdgeInputFormat.class,
                        EdgeListMapper.class);

        FileOutputFormat.setOutputPath(job, outputPath);

        job.setReducerClass(CloseTriangleReducer.class);
        job.setMapOutputKeyClass(TextPair.class);
        job.setMapOutputValueClass(Text.class);
        job.setOutputKeyClass(Text.class);
```

```
                job.setOutputValueClass(Text.class);

                return job.waitForCompletion(true) ? 0 : 1;

        }

        /**
         * Loads the Driver
         *
         *
         * @param args
         * @throws Exception
         */
        public static void main(String[] args) throws Exception {

                /*// Verify arguments
         if (args.length != 4) {
             System.err
                     .println("Usage: <EdgeListFolder> <RoundOne Folder> <outputFolder> <# of reducers>");
             System.exit(1);
         }
                */

                //Start counting time
         double starttime = (double) System.currentTimeMillis();

                int res = ToolRunner.run(new Configuration(),
                                new RoundTwoNodeIterator(), args);

                //Stop counting time
         double endtime = (double) System.currentTimeMillis();

         //Initialize the file you print the time results
         File timeFile = new File("/home/hduser/hadoop/time_triangles.txt");
         //Initialize the Output and Print streams
         FileOutputStream out;
         PrintStream p ;
         out = new FileOutputStream(timeFile, true);
         p = new PrintStream(out);
         Date now = new Date();
         p.append(now + " Elapsed Time - Round Two NodeIterator: " + ((endtime - starttime)) + "\r\n");
         p.close();
         out.close();


                System.exit(res);
        }

}
```

totaltrianglecounter.java

```
package gr.ntua.stasino.triangles.nodeiterator;

import gr.ntua.stasino.triangles.nodeiterator.RoundTwoNodeIterator.CloseTriangleReducer;
import gr.ntua.stasino.triangles.nodeiterator.RoundTwoNodeIterator.EdgeListMapper;
import gr.ntua.stasino.triangles.nodeiterator.RoundTwoNodeIterator.PossibleTriangleMapper;
import gr.ntua.stasino.triangles.utils.EdgeInputFormat;
import gr.ntua.stasino.triangles.utils.TextPair;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;
import java.util.Date;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Mapper.Context;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

```
public class TotalTriangleCounter extends Configured implements Tool {

	public static class DumbMapper extends
			Mapper<Text, Text, Text, IntWritable> {

		public void map(Text node, Text numofTriangles, Context context)
				throws IOException, InterruptedException {
			IntWritable nT = new IntWritable(Integer.parseInt(numofTriangles
					.toString()));
			context.write(node, nT);

		}
	}

	public static class TotalCountReducer extends
			Reducer<Text, IntWritable, Text, IntWritable> {
		private int sum = 0;

		public void reduce(Text node, Iterable<IntWritable> appearance,
				Context context) throws IOException, InterruptedException {
			for (IntWritable ap : appearance) {
				sum += ap.get();
			}
			context.write(node, new IntWritable(sum));
		}
	}

	public int run(String[] args) throws Exception {

		Path trianglePath = new Path(args[0]);
		Path outputPath = new Path(args[1]);
		int numReducers = Integer.parseInt(args[2]);

		Configuration conf = getConf();

		@SuppressWarnings("deprecation")
		Job job = new Job(conf);
		job.setJarByClass(TotalTriangleCounter.class);
		job.setJobName("Count Total Graph Triangles");
		job.setNumReduceTasks(numReducers);

		FileInputFormat.addInputPath(job, trianglePath);
		FileOutputFormat.setOutputPath(job, outputPath);

		job.setInputFormatClass(EdgeInputFormat.class);
		job.setMapperClass(DumbMapper.class);
		job.setReducerClass(TotalCountReducer.class);

		job.setOutputKeyClass(Text.class);
		job.setOutputValueClass(IntWritable.class);


		return job.waitForCompletion(true) ? 0 : 1;
	}

	public static void main(String[] args) throws Exception {

		// Verify arguments
		if (args.length != 3) {
		System.err.println("Usage: <Round 2 output folder> <outputFolder> <# of reducers>");
		System.exit(1);
		}

		//Start counting time
	   double starttime = (double) System.currentTimeMillis();

		int res = ToolRunner.run(new Configuration(), new TotalTriangleCounter(),
				args);

		//Stop counting time
	   double endtime = (double) System.currentTimeMillis();

	   //Initialize the file you print the time results
	   File timeFile = new File("/home/hduser/hadoop/time_triangles.txt");
	   //Initialize the Output and Print streams
	   FileOutputStream out;
	   PrintStream p ;
	   out = new FileOutputStream(timeFile, true);
	   p = new PrintStream(out);
	   Date now = new Date();
	   p.append(now + " Elapsed Time - Total Triangles Counter: " + ((endtime - starttime)) + "\r\n");
	   p.close();
	   out.close();

		System.exit(res);
	}
}
```

Triangles Pre-processing phase

degreecalculator.java

```
package triangles.preprocessing;

import triangles.utils.EdgeInputFormat;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

/**
 * Degree Calculator map reduce job counts the degree of an undirected graph
 * coming in an edgelist format
 *
 * @author Vassilis Trizonis
 * All right reserved
 *
 */
public class DegreeCalculator {
     /**
      * The map class of DegreeCalculator.
      */
     public static class TokenCounterMapper extends
                       Mapper<Object, Text, IntWritable, IntWritable> {

               private final static IntWritable one = new IntWritable(1);
               private final static IntWritable node2 = new IntWritable();
               private IntWritable word = new IntWritable();

               // private Text enhKey = new Text();

               public void map(Object key, Text value, Context context)
                                 throws IOException, InterruptedException {
                         final IntWritable key1 = new IntWritable(Integer.parseInt(key
                                           .toString()));

                         StringTokenizer valueIterator = new StringTokenizer(
                                           value.toString());
                         while (valueIterator.hasMoreTokens()) {
                                  String val = valueIterator.nextToken();
                                  int node = Integer.parseInt(val);
                                  node2.set(node);
                                  word.set(node);
                                  // enhKey.set(key.toString().concat("#").concat(val));
                                  context.write(word, one); // emit second node (=value)
                                  context.write(key1, one); // emit first node (=key)
                                  // context.write(enhKey, zero);
                                  context.write(key1, node2);
                         }
               }
     }

     // public static final class CustomPartitioner extends Partitioner<Text,
     // IntWritable> {
     // public int getPartition(Text key, IntWritable value, int numPartitions) {
     // // just partition by the first character of each key since that's
     // // how we are grouping for the reducer
     // int position = key.toString().indexOf("#");
     // String real_key = key.toString().substring(position+1);
     // return real_key.hashCode() % numPartitions;
     // }
     //
     // // public void configure(JobConf conf) { }
     // }
```

```
	/**
	 * The reducer class of WordCount
	 */
	public static class TokenCounterReducer extends
			Reducer<IntWritable, IntWritable, NodeWritable, NodeWritable> {

		private final static IntWritable zero = new IntWritable(0);
		private final static IntWritable minusOne = new IntWritable(-1);

		public void reduce(IntWritable key, Iterable<IntWritable> values,
				Context context) throws IOException, InterruptedException {

			NodeWritable node1 = new NodeWritable();
			NodeWritable node2 = new NodeWritable();

			node1.setNodeId(key);
			node1.setNodeDegree(minusOne);
			node1.toggleSum(false);

			int sum = 0;
			for (IntWritable value : values) {

				if (value.get() != 1) {
					node2.setNodeId(value);
					node2.setNodeDegree(minusOne);
					node2.toggleSum(false);
					context.write(node1, node2);

				} else if (value.get() == 1) {
					sum += value.get();
				}

			}

			node1.setNodeDegree(new IntWritable(sum));
			node1.toggleSum(true);
			if (node1.hasSum()) {
				node2.setNodeDegree(zero);
				node2.setNodeDegree(zero);
				node2.toggleSum(false);
				context.write(node1, node2);
			}
		}
	}

	/**
	 * The main entry point.
	 *
	 *
	 * @param args
	 * @throws Exception
	 */
	public static void main(String[] args) throws Exception {
		Configuration conf = new Configuration();
		String[] otherArgs = new GenericOptionsParser(conf, args)
				.getRemainingArgs();
		@SuppressWarnings("deprecation")
		Job job = new Job(conf, "Example Hadoop 0.20.1 WordCount");
		job.setJarByClass(DegreeCalculator.class);

		job.setMapperClass(TokenCounterMapper.class);
		job.setReducerClass(TokenCounterReducer.class);

		// job.setPartitionerClass(CustomPartitioner.class);
		job.setOutputKeyClass(IntWritable.class);
		job.setOutputValueClass(IntWritable.class);
		// customized key/value input based on 0.22 iteration of Hadoop
		job.setInputFormatClass(EdgeInputFormat.class);

		FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
		FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

		System.exit(job.waitForCompletion(true) ? 0 : 1);
	}
}
```

directedtoundirectedgraph.java

```
package triangles.preprocessing;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.StringTokenizer;
```

```java
import java.io.*;
//import java.net.URI;
import java.util.Date;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Partitioner;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

/**
 * Map reduce job that creates the undirected version of a directed graph that
 * comes in the form of an edgelist.
 *
 *
 */
public class DirectedToUndirectedGraph extends Configured implements Tool {

    /**
     * Mapper for the DirectedToUndirectedGraph job
     *
     * @author Vassilis Trizonis
     *
     */
    public static class Dir2UndirMapper extends
            Mapper<Object, Text, IntWritable, IntWritable> {

        private final static IntWritable one = new IntWritable(1);

        @Override
        public void map(Object key, Text value, Context context)
                throws IOException, InterruptedException {

            IntWritable keyOut = new IntWritable();
            IntWritable valOut = new IntWritable();

            keyOut.set(Integer.parseInt(key.toString()));
            StringTokenizer valueIterator = new StringTokenizer(
                    value.toString());
            while (valueIterator.hasMoreTokens()) {
                String val = valueIterator.nextToken();
                valOut.set(Integer.parseInt(val));
                // Map output (node1,node2)
                 context.write(keyOut, valOut);
                // Map output (node2, node1)
                 context.write(valOut, keyOut);

//                context.write(new Text(key.toString().concat("#").concat(val)),
//                        valOut);
//                context.write(new Text(val.concat("#").concat(key.toString())),
//                        keyOut);
            }
        }
    }

    /**
     * Reducer for the DirectedToUndirectedGraph job
     *
     *
     */
    public static class Dir2UndirReducer extends
            Reducer<IntWritable, IntWritable, IntWritable, IntWritable> {

        public void reduce(IntWritable key, Iterable<IntWritable> values,
                Context context) throws IOException, InterruptedException {
            // Text realKey = new Text(key.toString().split("#")[0]);
            // a buffer that collects node 2 from (node 1, node2)
            List<Integer> buffer = new ArrayList<Integer>();
            for (IntWritable val : values) {
            // check for possible duplicates
                if (!buffer.contains(val.get())) {
                    buffer.add(val.get());
                }
            }

            for (int i = 0; i < buffer.size(); i++) {
                context.write(key, new IntWritable(buffer.get(i)));
            }
        }
    }

    /**
      * @author Vassilis Trizonis
```

```
     * All right reserved
     *
     */
    public static final class CustomPartitioner extends
            Partitioner<Text, IntWritable> {
        public int getPartition(Text key, IntWritable value, int numPartitions) {
            // just partition by the first character of each key since that's
            // how we are grouping for the reducer
            int position = key.toString().indexOf("#");
            String real_key = key.toString().substring(position + 1);
            return real_key.hashCode() % numPartitions;
        }

        // public void configure(JobConf conf) { }
    }

    /**
     * Contains the Configuration properties for the job
     *
     * @throws Exception
     */
    public int run(String[] args) throws Exception {

        Path edgeListPath = new Path(args[0]);
        Path outputPath = new Path(args[1]);
        int numReducers = Integer.parseInt(args[2].trim());

        Configuration conf = getConf();
        // conf.set("mapred.tasktracker.map.tasks.maximum", "4");
        // conf.set("mapred.tasktracker.reduce.tasks.maximum", "4");
        conf.set("mapred.child.java.opts", "-Xmx512m");
        @SuppressWarnings("deprecation")
        Job job = new Job(conf, "Transform directed to undirected graph");
        job.setJarByClass(DirectedToUndirectedGraph.class);
        job.setNumReduceTasks(numReducers);

        job.setMapperClass(Dir2UndirMapper.class);
//          job.setPartitionerClass(CustomPartitioner.class);
        job.setCombinerClass(Dir2UndirReducer.class);
        job.setReducerClass(Dir2UndirReducer.class);
        job.setCombinerClass(Dir2UndirReducer.class);

        job.setMapOutputKeyClass(IntWritable.class);
        job.setMapOutputValueClass(IntWritable.class);

        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(IntWritable.class);



        // customized Input Format : node1 \t node2
        job.setInputFormatClass(triangles.utils.EdgeInputFormat.class);
        FileInputFormat.addInputPath(job, edgeListPath);
        FileOutputFormat.setOutputPath(job, outputPath);

        return job.waitForCompletion(true) ? 0 : 1;
    }

    /**
     * Loads the driver for the job
     *
     * @param args
     *            Command line arguments <br>
     *            Usage: EdgeListFolder outputFolder # of reducers
     * @throws Exception
     */
    public static void main(String[] args) throws Exception {

        // Verify arguments
        if (args.length != 3) {
            System.err
                    .println("Usage: <EdgeListFolder> <outputFolder> <# of reducers>");
            System.exit(1);
        }

        //Start counting time
        double starttime = (double) System.currentTimeMillis();

        // *****
        int res = ToolRunner.run(new Configuration(),
                new DirectedToUndirectedGraph(), args);


        //Stop counting time
        double endtime = (double) System.currentTimeMillis();

        //Initialize the file you print the time results
        File timeFile = new File("/home/hduser/hadoop/time_triangles.txt");
        //Initialize the Output and Print streams
```

```
        FileOutputStream out;
        PrintStream p ;
        out = new FileOutputStream(timeFile, true);
        p = new PrintStream(out);
        Date now = new Date();
        p.append(now + " Elapsed Time - Directed To Undirected Graph: " + ((endtime - starttime)) +
"\r\n");
        p.close();
        out.close();

        System.exit(res);
    }

}
```

longwritablepair.java

```
package triangles.preprocessing;

//TextPairComparator A RawComparator for comparing TextPair byte representations
//TextPairFirstComparator A custom RawComparator for comparing the first field of TextPair byte
representations
//TextPair
import java.io.*;

import org.apache.hadoop.io.*;

/**
 * A Writable implementation that stores a pair of LongWritable objects
 *
 * @author Vassilis Trizonis
 *
 */
public class LongWritablePair implements WritableComparable<LongWritablePair> {

       private LongWritable first;
       private LongWritable second;

       /**
        * LongWritablePair no args Constructor
        */
       public LongWritablePair() {
                set(new LongWritable(), new LongWritable());
       }

       /**
        * LongWritablePair Constructor
        *
        * @param first
        *             long
        * @param second
        *             long
        */
       public LongWritablePair(long first, long second) {
                set(new LongWritable(first), new LongWritable(second));
       }

       /**
        * LongWritablePair Constructor
        *
        * @param first
        *             LongWritable
        * @param second
        *             LongWritable
        */
       public LongWritablePair(LongWritable first, LongWritable second) {
                set(first, second);
       }

       /**
        * Setter
        *
        * @param first
        * @param second
        */
       public void set(LongWritable first, LongWritable second) {
                this.first = first;
                this.second = second;
       }

       /**
        * Getter
        *
        * @return first LongWritable
```

```
	 */
	public LongWritable getFirst() {
		return first;
	}

	/**
	 * Setter
	 *
	 * @return second LongWritable
	 */
	public LongWritable getSecond() {
		return second;
	}

	@Override
	public void write(DataOutput out) throws IOException {
		first.write(out);
		second.write(out);
	}

	@Override
	public void readFields(DataInput in) throws IOException {
		first.readFields(in);
		second.readFields(in);
	}

	@Override
	public int hashCode() {
		return first.hashCode();
	}

	@Override
	public boolean equals(Object o) {
		if (o instanceof LongWritablePair) {
			LongWritablePair tp = (LongWritablePair) o;
			return first == (tp.first) && second == tp.second;
		}
		return false;
	}

	@Override
	public String toString() {
		return first + "\t" + second;
	}

	@Override
	public int compareTo(LongWritablePair tp) {
		int cmp = first.compareTo(tp.first);
		if (cmp != 0) {
			return cmp;
		}
		return second.compareTo(tp.second);
	}

}
```

nodewritable.java

```
package triangles.preprocessing;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.*;

/**
 * Writable Type representing a graph node<br>
 * Consists of node Id and node degree
 *
 * @author Vassilis Trizonis
 *
 */
public class NodeWritable implements Writable {

	IntWritable nodeId;
	IntWritable nodeDegree;
	Boolean active = false;

	/**
	 * NodeWritable Constructor
	 *
	 * @param nodeId
```

```
         *            Node's id (IntWritable)
         * @param nodeDegree
         *            Node's degree (IntWritable)
         */
        public NodeWritable(IntWritable nodeId, IntWritable nodeDegree) {
                super();
                this.nodeId = nodeId;
                this.nodeDegree = nodeDegree;
                this.active = false;
        }

        /**
         *
         */
        public NodeWritable() {
                super();
        }

        /**
         * Getter
         *
         * @return node id
         */
        public IntWritable getNodeId() {
                return nodeId;
        }

        /**
         * Setter
         *
         * @param nodeId
         */
        public void setNodeId(IntWritable nodeId) {
                this.nodeId = nodeId;
        }

        /**
         * Getter
         *
         * @return nodeDegree
         */
        public IntWritable getNodeDegree() {
                return nodeDegree;
        }

        /**
         * Setter
         *
         * @param nodeDegree
         */
        public void setNodeDegree(IntWritable nodeDegree) {
                this.nodeDegree = nodeDegree;
        }

        /**
         * @return active
         */
        public Boolean hasSum() {
                return active;
        }

        /**
         * @param active
         */
        public void toggleSum(Boolean active) {
                this.active = active;
        }

        @Override
        public void write(DataOutput out) throws IOException {
                nodeId.write(out);
                nodeDegree.write(out);
        }

        @Override
        public void readFields(DataInput in) throws IOException {
                nodeId.readFields(in);
                nodeDegree.readFields(in);

        }

        @Override
        public int hashCode() { // same permId must go to same reducer. there fore
                                                                // just permId

                return nodeId.hashCode();
        }

}
```

secondnodetagger.java

```
package triangles.preprocessing;

import triangles.utils.EdgeInputFormat;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.io.*;
import org.apache.hadoop.fs.Path;

import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

/**
 * Tags the second node with the appropriate degree.
 *
 * @author Vassilis Trizonis
 *
 */
public class SecondNodeTagger extends Configured implements Tool {

	/**
	 * Identity Mapper
	 *
	 * @author Vassilis Trizonis
	 *
	 */
	public static class TagSecondNodeMapper extends
			Mapper<Text, Text, Text, Text> {//

		public void map(Text firstNode, Text secondNodePair, Context context)
				throws IOException, InterruptedException {

			context.write(firstNode, secondNodePair);
		}
	}

	/**
	 * The Reducer reads through the emitted pairs. As a pair indicated by "0"
	 * is first in order with its tag as its degree. Next k,v pairs are
	 * considered edge nodes and the first gets tagged with the -unique- degree.
	 *
	 * @author Vassilis Trizonis
	 *
	 */
	public static class TagSecondNodeReducer extends
			Reducer<Text, Text, Text, Text> {

		@Override
		public void reduce(Text key, Iterable<Text> values, Context context)
				throws IOException, InterruptedException {

			String getDegree = null;
			List<String> otherValues = new ArrayList<String>();
			for (Text v : values) {
				if (v.toString().split("\t")[0].equals("0")) {
					getDegree = v.toString().split("\t")[1];
				} else {
					otherValues.add(v.toString());
				}
			}

			for (String vout : otherValues) {
				context.write(new Text(key.toString() + "\t" + getDegree),
						new Text(vout));
			}

		}
```

```
        }

        /**
         * Contains the Configuration properties for the job
         *
         * @throws Exception
         */
        public int run(String[] args) throws Exception {

//              Path edgeListPath = new Path(args[0]);
                Path edgeListPath = new Path("/user/vassilis/temppath/");
                Path outputPath = new Path(args[1]);
                int numReducers = Integer.parseInt(args[2].trim());

                Configuration conf = getConf();
                // conf.set("mapred.tasktracker.map.tasks.maximum", "4");
                // conf.set("mapred.tasktracker.reduce.tasks.maximum", "4");
                // increase java heap space, because reducers are failing
                conf.set("mapred.child.java.opts", "-Xmx512m");
                @SuppressWarnings("deprecation")
                Job job = new Job(conf, "Tag Second Node with its Degreee");
                job.setJarByClass(SecondNodeTagger.class);
                job.setNumReduceTasks(numReducers);

                job.setMapperClass(TagSecondNodeMapper.class);
                job.setReducerClass(TagSecondNodeReducer.class);

                // job.setMapOutputKeyClass(Text.class);
                // job.setMapOutputValueClass(Text.class);

                // job.setPartitionerClass(CustomPartitioner.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(Text.class);

                job.setInputFormatClass(EdgeInputFormat.class);
                FileInputFormat.addInputPath(job, edgeListPath);
                FileOutputFormat.setOutputPath(job, outputPath);

                return job.waitForCompletion(true) ? 0 : 1;
        }

        /**
         * Loads the driver for the job
         *
         * @param args
         *            Command line arguments <br>
         *            Usage: EdgeListFolder outputFolder # of reducers
         * @throws Exception
         */
        public static void main(String[] args) throws Exception {
                // Verify arguments
                if (args.length != 3) {
                        System.err
                                        .println("Usage: <EdgeListFolder> <outputFolder> <# of
reducers>");
                        System.exit(1);
                }

                int res = ToolRunner.run(new Configuration(), new SecondNodeTagger(),
                                args);
                System.exit(res);
        }

}
```

tagedgelist.java

```
package triangles.preprocessing;

import triangles.nodeiterator.NodeDegreesTagger;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.util.ToolRunner;

/**
 * Tags an undirected graph edgelist with the degree of each node<br>
 * Essentially calls FirstNodeTaggerWithDegreeCounter and SecondNodeTagger
 * sequentially
 * <p>
 * Input: node1[tab]node2 <br>
 * Output: node1[tab]degree1[tab]node2[tab]degree2 <br>
 *
```

```
 *
 * @author Vassilis Trizonis
 *
 */
public class TagEdgeList {
	/**
	 * Calls the map reduce jobs in a row, then deletes the intermediate
	 * temporary path
	 * <p>
	 * Usage: EdgeListFolder outputFolder # of reducers<br>
	 *
	 * @param args
	 *
	 * @throws Exception
	 */
	public static void main(String[] args) throws Exception {

		// Verify arguments
		if (args.length != 3) {
			System.err
					.println("Usage: <EdgeListFolder> <outputFolder> <# of
reducers>");
			System.exit(1);
		}
		Configuration config = new Configuration();
		int res = ToolRunner.run(config,
				new NodeDegreesTagger(), args);
		if (res == 0) {
			res = ToolRunner.run(config, new SecondNodeTagger(),
					args);
		}
		//delete temporary output
		FileSystem hdfs = FileSystem.get(config);
		Path path = new Path("/user/hduser/triangles/t*");
		boolean isDeleted = hdfs.delete(path, true);

		//also delete initial edgelist

//		hdfs.delete(new Path(args[0]), false);
		System.exit(res);

	}
}
```

Utils

edgeinputformat.java

```
package triangles.utils;


/**
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you 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
 *a
 *     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.
 */


import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.compress.CompressionCodec;
import org.apache.hadoop.io.compress.CompressionCodecFactory;
```

```java
import org.apache.hadoop.mapreduce.InputFormat;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.JobContext;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;


/**
 * An {@link InputFormat} for plain text files. Files are broken into lines.
 * Either line feed or carriage-return are used to signal end of line.
 * Each line is divided into key and value parts by a separator byte. If no
 * such a byte exists, the key will be the entire line and value will be empty.
 */
//@InterfaceAudience.Public
//@InterfaceStability.Stable
public class EdgeInputFormat extends FileInputFormat<Text, Text> {

  @Override
  protected boolean isSplitable(JobContext context, Path file) {
    final CompressionCodec codec =
      new CompressionCodecFactory(context.getConfiguration()).getCodec(file);
    if (null == codec) {
      return true;
    }
    return codec instanceof SplittableCompressionCodec;
  }

  public RecordReader<Text, Text> createRecordReader(InputSplit genericSplit,
      TaskAttemptContext context) throws IOException {

    context.setStatus(genericSplit.toString());
    return new EdgeListRecordReader(context.getConfiguration());
  }

}
```

edgelistrecordreader.java

```java
package triangles.utils;
/**
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you 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.
 */


import java.io.IOException;

//import org.apache.hadoop.classification.InterfaceAudience;
//import org.apache.hadoop.classification.InterfaceStability;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.LineRecordReader;

/**
 * This class treats a line in the input as a key/value pair separated by a
 * separator character. The separator can be specified in config file
 * under the attribute name mapreduce.input.keyvaluelinerecordreader.key.value.separator. The default
 * separator is the tab character ('\t').
 */
//@InterfaceAudience.Public
//@InterfaceStability.Stable
public class EdgeListRecordReader extends RecordReader<Text, Text> {
  public static final String KEY_VALUE_SEPERATOR =
    "mapreduce.input.keyvaluelinerecordreader.key.value.separator";

  private final LineRecordReader lineRecordReader;
```

```
  private byte separator = (byte) '\t';

  private Text innerValue;

  private Text key;

  private Text value;

  public EdgeListRecordReader(Configuration conf)
    throws IOException {

    lineRecordReader = new LineRecordReader();
    String sepStr = conf.get(KEY_VALUE_SEPERATOR, "\t");
    this.separator = (byte) sepStr.charAt(0);
  }

  public void initialize(InputSplit genericSplit,
      TaskAttemptContext context) throws IOException {
    lineRecordReader.initialize(genericSplit, context);
  }

  public static int findSeparator(byte[] utf, int start, int length,
      byte sep) {
    for (int i = start; i < (start + length); i++) {
      if (utf[i] == sep) {
        return i;
      }
    }
    return -1;
  }

  public static void setKeyValue(Text key, Text value, byte[] line,
      int lineLen, int pos) {
    if (pos == -1) {
      key.set(line, 0, lineLen);
      value.set("");
    } else {
      key.set(line, 0, pos);
      value.set(line, pos + 1, lineLen - pos - 1);
    }
  }
  /** Read key/value pair in a line. */
  public synchronized boolean nextKeyValue()
    throws IOException {
    byte[] line = null;
    int lineLen = -1;
    if (lineRecordReader.nextKeyValue()) {
      innerValue = lineRecordReader.getCurrentValue();
      line = innerValue.getBytes();
      lineLen = innerValue.getLength();
    } else {
      return false;
    }
    if (line == null)
      return false;
    if (key == null) {
      key = new Text();
    }
    if (value == null) {
      value = new Text();
    }
    int pos = findSeparator(line, 0, lineLen, this.separator);
    setKeyValue(key, value, line, lineLen, pos);
    return true;
  }

  public Text getCurrentKey() {
    return key;
  }

  public Text getCurrentValue() {
    return value;
  }

  public float getProgress() throws IOException {
    return lineRecordReader.getProgress();
  }

  public synchronized void close() throws IOException {
    lineRecordReader.close();
  }
}
```

orderbydegree.java

```
package triangles.utils;

import org.apache.hadoop.io.Text;

/**
 * @author Vassilis Trizonis
 *
 */
public class OrderByDegree {
	private int nodeOne;
	private int nodeTwo;
	private int degreeOne;
	private int degreeTwo;

	/**
	 * @param value
	 */
	public void parse(Text value) {

		String[] str = value.toString().split("\t");
		nodeOne = Integer.parseInt(str[0].trim());
		nodeTwo = Integer.parseInt(str[2].trim());
		degreeOne = Integer.parseInt(str[1].trim());
		degreeTwo = Integer.parseInt(str[3].trim());

	}

//	public boolean isValid(){
//		return (str.length ==4) ? true : false;
//	}
	/**
	 * @param deg1
	 * @param deg2
	 * @return True if node 1 > node 2
	 */
	public boolean compareDegrees() {
		return (degreeTwo > degreeOne) ? false : ((degreeOne > degreeTwo) ? true :
tieBreaker(nodeOne,
				nodeTwo));

	}

	/**
	 * @param n1
	 * @param n2
	 * @return True if node 1 id is bigger
	 */
	public boolean tieBreaker(long n1, long n2) {
		return (n1 > n2) ? true : false;
	}

	/**
	 * @return the nodeOne
	 */
	public long getNodeOne() {
		return nodeOne;
	}

	/**
	 * @param nodeOne the nodeOne to set
	 */
	public void setNodeOne(int nodeOne) {
		this.nodeOne = nodeOne;
	}

	/**
	 * @return the nodeTwo
	 */
	public long getNodeTwo() {
		return nodeTwo;
	}

	/**
	 * @param nodeTwo the nodeTwo to set
	 */
	public void setNodeTwo(int nodeTwo) {
		this.nodeTwo = nodeTwo;
	}
}
```

splittablecompressioncodec.java

```
package triangles.utils;

/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you 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.
 */


import java.io.IOException;
import java.io.InputStream;

import org.apache.hadoop.io.compress.*;


/**
 * This interface is meant to be implemented by those compression codecs
 * which are capable to compress / de-compress a stream starting at any
 * arbitrary position.
 *
 * Especially the process of de-compressing a stream starting at some arbitrary
 * position is challenging.  Most of the codecs are only able to successfully
 * de-compress a stream, if they start from the very beginning till the end.
 * One of the reasons is the stored state at the beginning of the stream which
 * is crucial for de-compression.
 *
 * Yet there are few codecs which do not save the whole state at the beginning
 * of the stream and hence can be used to de-compress stream starting at any
 * arbitrary points.  This interface is meant to be used by such codecs.  Such
 * codecs are highly valuable, especially in the context of Hadoop, because
 * an input compressed file can be split and hence can be worked on by multiple
 * machines in parallel.
 */

//@InterfaceAudience.Public
//@InterfaceStability.Evolving


public interface SplittableCompressionCodec extends CompressionCodec {

  /**
   * During decompression, data can be read off from the decompressor in two
   * modes, namely continuous and blocked.  Few codecs (e.g. BZip2) are capable
   * of compressing data in blocks and then decompressing the blocks.  In
   * Blocked reading mode codecs inform 'end of block' events to its caller.
   * While in continuous mode, the caller of codecs is unaware about the blocks
   * and uncompressed data is spilled out like a continuous stream.
   */
  public enum READ_MODE {CONTINUOUS, BYBLOCK};

  /**
   * Create a stream as dictated by the readMode.  This method is used when
   * the codecs wants the ability to work with the underlying stream positions.
   *
   * @param seekableIn  The seekable input stream (seeks in compressed data)
   * @param start The start offset into the compressed stream. May be changed
   *              by the underlying codec.
   * @param end The end offset into the compressed stream. May be changed by
   *            the underlying codec.
   * @param readMode Controls whether stream position is reported continuously
   *                 from the compressed stream only only at block boundaries.
   * @return  a stream to read uncompressed bytes from
   */
  SplitCompressionInputStream createInputStream(InputStream seekableIn,
      Decompressor decompressor, long start, long end, READ_MODE readMode)
      throws IOException;

}
```

taggedinputsplit.java

```
/**
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you 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.
 */

package triangles.utils;

import java.io.DataInput;
import java.io.DataInputStream;
import java.io.DataOutput;
import java.io.DataOutputStream;
import java.io.IOException;

import org.apache.hadoop.conf.Configurable;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.serializer.Deserializer;
import org.apache.hadoop.io.serializer.SerializationFactory;
import org.apache.hadoop.io.serializer.Serializer;
import org.apache.hadoop.mapreduce.InputFormat;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.util.ReflectionUtils;

/**
 * An {@link InputSplit} that tags another InputSplit with extra data for use
 * by {@link DelegatingInputFormat}s and {@link DelegatingMapper}s.
 */
class TaggedInputSplit extends InputSplit implements Configurable, Writable {

  private Class<? extends InputSplit> inputSplitClass;

  private InputSplit inputSplit;

  @SuppressWarnings("unchecked")
  private Class<? extends InputFormat> inputFormatClass;

  @SuppressWarnings("unchecked")
  private Class<? extends Mapper> mapperClass;

  private Configuration conf;

  public TaggedInputSplit() {
    // Default constructor.
  }

  /**
   * Creates a new TaggedInputSplit.
   *
   * @param inputSplit The InputSplit to be tagged
   * @param conf The configuration to use
   * @param inputFormatClass The InputFormat class to use for this job
   * @param mapperClass The Mapper class to use for this job
   */
  @SuppressWarnings("unchecked")
  public TaggedInputSplit(InputSplit inputSplit, Configuration conf,
      Class<? extends InputFormat> inputFormatClass,
      Class<? extends Mapper> mapperClass) {
    this.inputSplitClass = inputSplit.getClass();
    this.inputSplit = inputSplit;
    this.conf = conf;
    this.inputFormatClass = inputFormatClass;
    this.mapperClass = mapperClass;
  }

  /**
   * Retrieves the original InputSplit.
   *
   * @return The InputSplit that was tagged
   */
  public InputSplit getInputSplit() {
    return inputSplit;
  }
```

```
  /**
   * Retrieves the InputFormat class to use for this split.
   *
   * @return The InputFormat class to use
   */
  @SuppressWarnings("unchecked")
  public Class<? extends InputFormat> getInputFormatClass() {
    return inputFormatClass;
  }

  /**
   * Retrieves the Mapper class to use for this split.
   *
   * @return The Mapper class to use
   */
  @SuppressWarnings("unchecked")
  public Class<? extends Mapper> getMapperClass() {
    return mapperClass;
  }

  public long getLength() throws IOException, InterruptedException {
    return inputSplit.getLength();
  }

  public String[] getLocations() throws IOException, InterruptedException {
    return inputSplit.getLocations();
  }

  @SuppressWarnings("unchecked")
  public void readFields(DataInput in) throws IOException {
    inputSplitClass = (Class<? extends InputSplit>) readClass(in);
    inputFormatClass = (Class<? extends InputFormat<?, ?>>) readClass(in);
    mapperClass = (Class<? extends Mapper<?, ?, ?, ?>>) readClass(in);
    inputSplit = (InputSplit) ReflectionUtils
       .newInstance(inputSplitClass, conf);
    SerializationFactory factory = new SerializationFactory(conf);
    Deserializer deserializer = factory.getDeserializer(inputSplitClass);
    deserializer.open((DataInputStream)in);
    inputSplit = (InputSplit)deserializer.deserialize(inputSplit);
  }

  private Class<?> readClass(DataInput in) throws IOException {
    String className = Text.readString(in);
    try {
      return conf.getClassByName(className);
    } catch (ClassNotFoundException e) {
      throw new RuntimeException("readObject can't find class", e);
    }
  }

  @SuppressWarnings("unchecked")
  public void write(DataOutput out) throws IOException {
    Text.writeString(out, inputSplitClass.getName());
    Text.writeString(out, inputFormatClass.getName());
    Text.writeString(out, mapperClass.getName());
    SerializationFactory factory = new SerializationFactory(conf);
    Serializer serializer =
          factory.getSerializer(inputSplitClass);
    serializer.open((DataOutputStream)out);
    serializer.serialize(inputSplit);
  }

  public Configuration getConf() {
    return conf;
  }

  public void setConf(Configuration conf) {
    this.conf = conf;
  }

}
```

textpair.java

```
package gr.ntua.stasino.triangles.utils;

// cc TextPair A Writable implementation that stores a pair of Text objects
// cc TextPairComparator A RawComparator for comparing TextPair byte representations
// cc TextPairFirstComparator A custom RawComparator for comparing the first field of TextPair byte
representations
// vv TextPair
import java.io.*;

import org.apache.hadoop.io.*;
```

```
public class TextPair implements WritableComparable<TextPair> {

  private Text first;
  private Text second;

  public TextPair() {
    set(new Text(), new Text());
  }

  public TextPair(String first, String second) {
    set(new Text(first), new Text(second));
  }

  public TextPair(Text first, Text second) {
    set(first, second);
  }

  public void set(Text first, Text second) {
    this.first = first;
    this.second = second;
  }

  public Text getFirst() {
    return first;
  }

  public Text getSecond() {
    return second;
  }

  @Override
  public void write(DataOutput out) throws IOException {
    first.write(out);
    second.write(out);
  }

  @Override
  public void readFields(DataInput in) throws IOException {
    first.readFields(in);
    second.readFields(in);
  }

  @Override
  public int hashCode() {
    return first.hashCode() * 163 + second.hashCode();
  }

  @Override
  public boolean equals(Object o) {
    if (o instanceof TextPair) {
      TextPair tp = (TextPair) o;
      return first.equals(tp.first) && second.equals(tp.second);
    }
    return false;
  }

  @Override
  public String toString() {
    return first + "\t" + second;
  }

  @Override
  public int compareTo(TextPair tp) {
    int cmp = first.compareTo(tp.first);
    if (cmp != 0) {
      return cmp;
    }
    return second.compareTo(tp.second);
  }
  // ^^ TextPair

  // vv TextPairComparator
  public static class Comparator extends WritableComparator {

    private static final Text.Comparator TEXT_COMPARATOR = new Text.Comparator();

    public Comparator() {
      super(TextPair.class);
    }

    @Override
    public int compare(byte[] b1, int s1, int l1,
                       byte[] b2, int s2, int l2) {

      try {
        int firstL1 = WritableUtils.decodeVIntSize(b1[s1]) + readVInt(b1, s1);
        int firstL2 = WritableUtils.decodeVIntSize(b2[s2]) + readVInt(b2, s2);
        int cmp = TEXT_COMPARATOR.compare(b1, s1, firstL1, b2, s2, firstL2);
        if (cmp != 0) {
```

```
          return cmp;
        }
        return TEXT_COMPARATOR.compare(b1, s1 + firstL1, l1 - firstL1,
                                       b2, s2 + firstL2, l2 - firstL2);
      } catch (IOException e) {
        throw new IllegalArgumentException(e);
      }
    }
  }

  static {
    WritableComparator.define(TextPair.class, new Comparator());
  }
  // ^^ TextPairComparator

  // vv TextPairFirstComparator
  public static class FirstComparator extends WritableComparator {

    private static final Text.Comparator TEXT_COMPARATOR = new Text.Comparator();

    public FirstComparator() {
      super(TextPair.class);
    }

    @Override
    public int compare(byte[] b1, int s1, int l1,
                       byte[] b2, int s2, int l2) {

      try {
        int firstL1 = WritableUtils.decodeVIntSize(b1[s1]) + readVInt(b1, s1);
        int firstL2 = WritableUtils.decodeVIntSize(b2[s2]) + readVInt(b2, s2);
        return TEXT_COMPARATOR.compare(b1, s1, firstL1, b2, s2, firstL2);
      } catch (IOException e) {
        throw new IllegalArgumentException(e);
      }
    }

    @Override
    public int compare(WritableComparable a, WritableComparable b) {
      if (a instanceof TextPair && b instanceof TextPair) {
        return ((TextPair) a).first.compareTo(((TextPair) b).first);
      }
      return super.compare(a, b);
    }
  }
  // ^^ TextPairFirstComparator

// vv TextPair
}
// ^^ TextPair
```