Σύγκριση απόδοσης απλών τελεστών σε δεδομένα γράφου ανάμεσα στις κατανεμημένες μηχανές επεξεργασίας δεδομένων γράφου **Gelly** και **GraphX**

Μυροπούλου Νεφέλη
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο
Αθήνα, Ελλάδα
el17197@mail.ntua.gr

Τσεριώτης Άδωνις
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο
Αθήνα, Ελλάδα
el17838@mail.ntua.gr

Συρογιάννης Γεώργιος
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο
Αθήνα, Ελλάδα
el17140@mail.ntua.gr

***Abstract*—**Στόχος της παρούσας αναφοράς αποτελεί η παρουσίαση**,** και μετέπειτα η πειραματική σύγκριση των σύγχρονων μηχανών κατανεμημένης επεξεργασίας δεδομένων γράφων **GraphX** του **Apache Spark,** και **Gelly** του **Apache Flink.** Για τη διεξαγωγή των πειραμάτων χρησιμοποιήθηκε σύμπλεγμα τριών υπολογιστικών κόμβων**,** ενώ ως μετρικές σύγκρισης εφαρμόστηκαν οι γνωστοί αλγόριθμοι **PageRank** και **Connected Components.** Αντίστοιχα**,** ως δεδομένα εισόδου χρησιμοποιήθηκαν τυχαίοι**,** αλλά και **scale-free** γράφοι**,** με διαβαθμισμένο μέγεθος**.** Τέλος**,** πραγματοποιήθηκε σχολιασμός των αποτελεσμάτων και επιχειρήθηκε η διεξαγωγή γενικότερων συμπερασμάτων για τα ύπο μελέτη **frameworks.**

***Keywords*—** Big Graph frameworks, Distributed computing, Graph algorithms, GraphX, Gelly, Apache Spark, Apache Flink.

I. ΕΙΣΑΓΩΓΗ

Οι κατανεμημένες μηχανές επεξεργασίας γράφων στοχεύουν στην αποδοτικότερη επεξεργασία εξαιρετικά μεγάλων γράφων, οι οποίοι πολλές φορές περιλαμβάνουν ως και δισεκατομμύρια κορυφές και τρισεκατομμύρια ακμές. Τέτοιοι γράφοι χρησιμοποιούνται συχνά για την περιγραφή πραγματικών δικτύων, όπως αυτών που συναντώνται στα μέσα κοινωνικής δικτύωσης. Σε αυτά τα σύνθετα δίκτυα, οι κορυφές V αντιπροσωπεύουν ένα σύνολο αντικειμένων (π.χ. χρήστες ενός κοινωνικού δικτύου), ενώ το σύνολο ακμών E αναπαριστά συνδέσεις μεταξύ των αντικειμένων αυτών (π.χ. σχέσεις ακολούθων). Η ανάγκη εφαρμογής τελεστών και αλγορίθμων με στόχο την κατανόηση της δομής, ή και των ειδικών ιδιοτήτων τέτοιων γράφων, έχει οδηγήσει στην ανάπτυξη πλήθους προγραμματιστικών μοντέλων. Τα μοντέλα αυτά συχνά εξειδικεύονται στην κατανεμημένη, παράλληλη επεξεργασία μεγάλων γράφων (graph-specific distributed computing), χωρίς όμως να εκλείπουν και οι πιο απλές προσεγγίσεις κατανεμημένης επεξεργασίας (general-purpose distributed computing), με χαρακτηριστικό παράδειγμα το MapReduce. Η εκτέλεση τους πραγματοποιείται με τη χρήση συστάδας (cluster) υπολογιστικών κόμβων, δηλαδή μηχανών που λειτουργούν αυτόνομα, διαθέτουν ξεχωριστή μνήμη και ξεχωριστό επεξεργαστή και ανταλλάζουν μηνύματα μέσω δικτύου.
Στην παρούσα εργασία πραγματοποιείται πειραματική σύγκριση των κατανεμημένων μηχανών επεξεργασίας γράφων, GraphX (του Apache Spark) και Gelly, του Apache Flink. Οι μηχανές αυτές, στηριζόμενες σε μερικά από τα προαναφερθείσα προγραμματιστικά μοντέλα, αποτελούν δημοφιλείς επιλογές για τη διαχείριση μεγάλων δεδομένων (Big Data). Στη συνέχεια, αρχικά ακολουθεί συνοπτική παρουσίαση ορισμένων καίριων προγραμματιστικών μοντέλων, καθώς και περιγραφή των βασικών χαρακτηριστικών των GraphX και Gelly. Σκοπός του τμήματος αυτού αποτελεί η θεωρητική θεμελίωση της εργασίας, καθώς και η εξοικείωση του αναγνώστη με τις σημαντικότερες έννοιες των μηχανισμών που πρόκειται να μελετηθούν. Το 2ο μέρος αφορά το πειραματικό σκέλος. Σε αυτό, περιγράφονται η υποδομή, οι υπολογιστικοί πόροι, οι μετρικές και τα σύνολα δεδομένων (γράφοι) που χρησιμοποιήθηκαν, καθώς και η διαδικασία που ακολουθήθηκε για τη διεξαγωγή των πειραμάτων. Τέλος, μετά την παρουσίαση των αποτελεσμάτων επιχειρείται η ερμηνεία τους και η εξαγωγή χρήσιμων συμπερασμάτων.

II. ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΑ ΜΟΝΤΕΛΑ

A. General-Purpose Distributed Computing: MapReduce

Το MapReduce αποτελεί ένα ευρέως εδραιωμένο προγραμματιστικό μοντέλο και χαρακτηριστικό δείγμα κατανεμημένης επεξεργασίας δεδομένων γενικού σκοπού. Εισήχθη αρχικά από τη Google το 2004 [1], με στόχο την ταχύτερη και αποδοτικότερη διαχείριση αποτελεσμάτων αναζήτησης. Αρχή της MapReduce μεθόδου αποτελεί ο διαχωρισμός μεγάλου όγκου δεδομένων σε μικρότερα τμήματα, και στη συνέχεια, η παράλληλη επεξεργασία αυτών. Ένα βασικό MapReduce πρόγραμμα περιλαμβάνει δύο συναρτήσεις ορισμένες από τον χρήστη, τη Map και τη Reduce συνάρτηση. Και οι δύο αυτές συναρτήσεις δέχονται ως είσοδο και παράγουν ως έξοδο, key-value ζεύγη. Αναλυτικότερα, τα δεδομένα εισόδου διαιρούνται σε ομάδες

key-value ζευγών ανά κόμβο, οι οποίες στη συνέχεια υφίστανται επεξεργασία, με βάση την ορισμένη Map συνάρτηση. Η επεξεργασία αυτών των τμημάτων δεδομένων πραγματοποιείται παράλληλα και ανεξάρτητα, επιταχύνοντας έτσι σημαντικά τους συνολικούς υπολογισμούς. Το Reduce στάδιο εφαρμόζεται πάντα έπειτα από το αντίστοιχο Map. Κατά αυτή τη φάση, λαμβάνονται τα σύνολα key-value ζευγών που έχουν προκύψει ως έξοδος, και με χρήση τη Reduce, συνδυάζονται παράλληλα σε μικρότερα, αντίστοιχα σύνολα. Τα σύνολα αυτά αποτελούν το τελικό αποτέλεσμα της παραπάνω διαδικασίας και αποθηκεύονται στο κατανεμημένο σύστημα μνήμης.

Τα σημαντικότερα οφέλη του MapReduce μοντέλου σχετίζονται με την υψηλή κλιμακωσιμότητα, την ταχύτητα και την ευελιξία του. Αξίζει να σημειωθεί ακόμα ότι στις περισσότερες υλοποιήσεις του, με εξέχουσα αυτή του Apache Hadoop [2], περιλαμβάνονται μηχανισμοί ανάκτησης δεδομένων, για την αντιμετώπιση περιπτώσεων σφάλματος ή απώλειας κάποιου υπολογιστικού κόμβου. Επίσης, επιχειρείται η ελαχιστοποίηση της συμφόρησης του αντίστοιχου δικτύου, επιδιώκοντας data locality, δηλαδή επεξεργασία των δεδομένων στους κόμβους όπου αυτά ήδη βρίσκονται αποθηκευμένα. [3] Παρά την ευρεία εφαρμοσιμότητα του, το MapReduce δεν αποτελεί την ιδανική επιλογή για την επεξεργασία μεγάλων γράφων, καθώς πολλοί αλγόριθμοι σχετιζόμενοι με γράφους απαιτούν επαναληπτική εκτέλεση. Η ανάγκη αυτή περιορίζει σημαντικά τη δυνατότητα παραλληλοποίησης διαδικασιών κι επομένως και τα οφέλη του μοντέλου. Επίσης, η προσομοίωση των επαναλήψεων μέσω διαδοχικών MapReduce κύκλων αποτελεί μια υπολογιστικά δαπανηρή επιλογή, ενώ παράλληλα απαιτεί την επαναλαμβανόμενη μεταφορά των ίδιων δεδομένων εντός του δικτύου, οδηγώντας τελικά σε αχρείαστη συμφόρηση του. [4] Για τους παραπάνω λόγους, και με στόχο την αποδοτικότερη κατανεμημένη διαχείριση γράφων, έχουν αναπτυχθεί νέα προγραμματιστικά μοντέλα, εμπνεόμενα και από τη δομή των ίδιων των γράφων. Μερικά εξ αυτών παρουσιάζονται παρακάτω.

B. Graph-Specific Distributed Computing: Vertex-centric graph processing

Ένα είδος κατανεμημένης επεξεργασίας ειδικά για γράφους αποτελεί η vertex-centric προσέγγιση. Όπως φανερώνει και η ονομασία, σε αυτά τα μοντέλα, τα προγράμματα εκφράζονται από την προοπτική των κόμβων του γράφου και οι αλγόριθμοι εκτελούνται επαναληπτικά για κάθε κορυφή. Ένα από τα βασικότερα vertex-centric προγραμματιστικά μοντέλα αποτελεί το μοντέλο Pregel. [5] Το Pregel έχει βασιστεί στο Bulk Synchronous Parallel (BSP) μοντέλο, ένα γενικό επαναληπτικό μοντέλο για παράλληλη επεξεργασία. Μια επανάληψη ('superstep') περιλαμβάνει την εκτέλεση ενός αλγορίθμου σε κάθε επεξεργαστή, ακολουθούμενη από μια φάση επικοινωνίας, όπου μηνύματα ανταλλάσσονται εντός του δικτύου. Η μεταβολή της κατάστασης μιας κορυφής του γράφου είναι δυνατή μόνο κατά το πρώτο στάδιο, ενώ στη συνέχεια, η φάση επικοινωνίας διασφαλίζει την ενημέρωση των γειτονικών κορυφών για τυχόν αλλαγές. Η εκτέλεση πραγματοποιείται με σύγχρονο τρόπο, καθώς κάθε κόμβος αρχίζει την εκτέλεση ενός νέου superstep αφού όλοι οι υπόλοιποι έχουν ολοκληρώσει το προηγούμενο, διασφαλίζοντας έτσι την εξάλειψη τυχόν deadlocks ή data races. [4], [6]

Αξίζει να σημειωθεί ακόμα ότι, η βιβλιοθήκη Pregel διαιρεί τους γράφους εισόδου σε τμήματα (partitions), καθένα εκ των οποίων αποτελείται από ένα σύνολο κορυφών και από όλες τις εξερχόμενες ακμές τους. Η ανάθεση μιας κορυφής σε ένα partition εξαρτάται μόνο από το vertex ID της, ενώ το πλήθος των partitions και ο διαμοιρασμός τους σε υπολογιστικούς κόμβους συνήθως καθορίζονται από έναν master κόμβο. Ο κόμβος αυτός αναλαμβάνει επίσης το συγχρονισμό των υπόλοιπων κόμβων (slaves) και την ανίχνευση τυχών αποτυχιών τους. Τέλος, αναφέρουμε ότι στην περίπτωση που σε έναν υπολογιστικό κόμβο του cluster ανατεθούν πάνω από ένα partition, ο κόμβος πραγματοποιεί παράλληλη επεξεργασία τους. [4], [6]

Μια μετεξέλιξη του Pregel μοντέλου, αποτελεί το (επίσης επαναληπτικό και vertex-centric) Gather-Apply-Scatter (GAS) μοντέλο. Σε αυτό, κάθε superstep συντίθεται από τρεις φάσεις: τη gather, apply και scatter φάση. Στη gather φάση κάθε κορυφή συλλέγει δεδομένα από τις γειτονικές της κορυφές, με βάση μια οριζόμενη gather συνάρτηση. Με βάση τα συλλεχθέντα δεδομένα ένας κόμβος μπορεί να μεταβάλει την κατάσταση του στο apply στάδιο, ενώ τέλος, η τελική κατάσταση του κάθε κόμβου μεταβιβάζεται στις υπόλοιπες κορυφές στη scatter φάση, μέσω των εξερχόμενων του ακμών. Κατά τη scatter φάση μπορεί να γίνει ανανέωση και δεδομένων σχετιζόμενων με τις ακμές, ενώ αλλαγές στις τιμές των κορυφών μπορούν να πραγματοποιηθούν μόνο κατά την apply φάση. Επομένως, καθώς vertex δεδομένα δεν εγγράφονται κατά τα gather και scatter στάδια, για αυτά υπάρχει δυνατότητα παραλληλοποίησης. Παραλλαγές του GAS μοντέλου είναι επίσης δημοφιλείς, με χαρακτηριστική τη λεγόμενη Gather-Sum-Apply, όπου η λογική κάποιων από των σταδίων του superstep έχει παραλλαχθεί. [4], [7]

III. Frameworks

A. Apache Spark: GraphX

Το GraphX αποτελεί ένα νέο συστατικό του Apache Spark, με ειδίκευση στους γράφους και στην παράλληλη επεξεργασία τους. Υλοποιεί την προαναφερθείσα Gather-Apply-Scatter προγραμματιστική τεχνική, ενώ παρέχει τις Java και Scala ως επιλογές γλώσσας προγραμματισμού. Στο GraphX, οι γράφοι συνίστανται από ένα RDD (Resilient Distributed Dataset) κορυφών και από ένα RDD ακμών. Το RDD αποτελεί βασική, abstract κλάση του Apache Spark, με την οποία μοντελοποιείται μια immutable, partitioned συλλογή αντικειμένων που μπορεί να υποστεί παράλληλη επεξεργασία. Έτσι, όπως και τα RDDs, οι γράφοι στο GraphX είναι immutable, κατανεμημένα και ανθεκτικά στα

σφάλματα (fault-tolerant) αντικείμενα. Αλλαγές σε ένα γράφο πραγματοποιούνται επομένως μέσω της κατασκευής νέου γράφου, που περιέχει τις επιθυμητές αλλαγές. Τμήματα κάθε γράφου μοιράζονται προς επεξεργασία σε διάφορα μηχανήματα, με χρήση vertex cuts, ενώ στην περίπτωση απώλειας κάποιου υπολογιστικού κόμβου, το εκεί στεγαζόμενο τμήμα γράφου δεν χάνεται, αλλά ανακατασκευάζεται σε κάποιον άλλον κόμβο. Δυνατή είναι η κατασκευή κατευθυνόμενων ή μη γράφων, αλλά και η ύπαρξη παράλληλων ακμών σε αυτούς.
Επιπλέον, αναφέρουμε ότι κάθε κορυφή ενός γράφου αναπαρίσταται από ένα vertex ID, οι ακμές περιγράφονται από τα IDs της κορυφής αφετηρίας και της κορυφής προορισμού, ενώ και κορυφές και ακμές μπορούν να φέρουν δεδομένα. Το GraphX προσφέρει ακόμα μια πληθώρα υλοποιημένων τελεστών που μπορούν να χρησιμοποιηθούν για την παροχή πληροφοριών για το γράφο εισόδου, για τη δημιουργία νέων γράφων με νέα δομή και χαρακτηριστικά και για την αποτελεσματικότερη μετάδοση μηνυμάτων από μια κορυφή σε άλλες, μεταξύ άλλων. Τέλος, το GraphX παρέχει υλοποιημένο ένα σύνολο αλγορίθμων γράφων, με χαρακτηριστικά παραδείγματα τους συχνά χρησιμοποιούμενους PageRank, Connected Components, Triangle Count κ.α.. Μερικές από αυτές τις υλοποιήσεις χρησιμοποιήθηκαν και στη συνέχεια, για τη διεξαγωγή των πειραμάτων. [8]

B. Apache Flink: Gelly

Το Gelly αποτελεί το API και τη βιβλιοθήκη του Apache Flink framework για επεξεργασία γράφων. Παρόμοια με το GraphX, υλοποιεί vertex-centric προγραμματιστικές τεχνικές και συγκεκριμένα εφαρμόζει μια παραλλαγή του Pregel μοντέλου (Spargel API), καθώς και την παραλλαγή της Gather-Apply-Scatter τεχνικής, Gather-Sum-Apply. Έως τώρα, το Gelly υποστηρίζεται στις γλώσσες προγραμματισμού Java και Scala, με τις μεθόδους της δεύτερης να είναι υλοποιημένες ως wrappers πάνω σε βασικές Java μεθόδους. Στο Gelly, ένας γράφος περιγράφεται ως ένα DataSet από κορυφές κι ένα DataSet από ακμές. Σημειώνουμε ότι η κλάση DataSet χρησιμοποιείται από το Apache Flink για την αναπαράσταση μιας immutable, πεπερασμένης συλλογής από αντικείμενα, η οποία μπορεί να περιέχει διπλότυπα.
Όπως και στο GraphX, μια κορυφή αναπαρίσταται με τη χρήση ενός vertex ID, ενώ μια ακμή περιγράφεται από τα IDs των κορυφών που ενώνει. Τελεστές υψηλού επιπέδου επιτρέπουν την ανάκτηση σημαντικών μετρικών και ιδιοτήτων του γράφου εισόδου, τη δημιουργία νέων γράφων από τον τρέχοντα, καθώς και τη μετάδοση μηνυμάτων από μια κορυφή, στη γειτονιά της. Τέλος, για τη διευκόλυνση της ανάλυσης μεγάλων σε όγκο γράφων, παρέχεται υλοποιημένη μια βιβλιοθήκη από σχετιζόμενους αλγορίθμους, με χαρακτηριστικά παραδείγματα τους PageRank, Single-Source-Shortest-Paths, Label Propagation, Connected Components κ.α. [9]

IV. ΔΙΕΞΑΓΩΓΗ ΠΕΙΡΑΜΑΤΩΝ

A. Βήματα Εγκατάστασης

1) Δημιουργία VMs: Οι αναγκαίοι πόροι για την διεξαγωγή των πειραμάτων παρέχονται απο την υπηρεσία *Okeanos* και συγκεκριμένα απο τις *Cyclades*. Παρέχονται 90 GB χώρος αποθήκευσης, 12 CPUs, 24 GB RAM και 1 Private Network τα οποία έχουμε διαμερίσει ισόποσα σε 3 VMs (1 master & 2 workers) με λογισμικό Ubuntu Server LTS.

2) Private network: Μετά το πέρας της δημιουργίας των VM, χρειάζεται η εξασφάλιση της επικοινωνίας μεταξύ τους μέσω του Ιδιωτικού δικτύου που παρέχεται απο τον Ωκεανό. Με βάση το documentation, δημιουργούμε το ιδιωτικό δίκτυο και αναθέτουμε μια IP σε καθενα.

3) Passwordless SSH: Για την χρήση των GraphX και Flink, θα χρειαστούμε επικοινωνία μεταξύ των μηχανημάτων μέσω SSH χωρίς κωδικό. Αυτό γίνεται εφικτό προσθέτοντας τις IP και τα hostnames στο **/etc/hosts** αρχείο.

```
192.168.0.1 master
192.168.0.2 slave2
192.168.0.3 slave1
```

Αυτή η ρύθμιση κάνει εφικτή την αναφορά σε μια IP π.χ. *192.168.0.1* χρησιμοποιώντας το όνομα *master*. Στην συνέχεια, με την εκτέλεση του παρακάτω script δημιουργούμε και προσθέτουμε το ssh public key του master σε καθένα απο τους workers.

```
#!/bin/bash

ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/
   authorized_keys
scp -r ~/.ssh/ user@slave1:~/
scp -r ~/.ssh/ user@slave2:~/
```

4) Πρόσβαση στο διαδίκτυο απο τους *workers:* Οι 2 workers δεν έχουν πρόσβαση στο διαδίκτυο (no public IP) και για να τους δοθεί, θα χρησιμοποιήσουμε τον master κόμβο σαν NAT χρησιμοποιώντας τα παρακάτω scripts.

Run on master:

```
#!/bin/bash

echo "Enabling ipv4 forwarding (cleaning
    old rules)"
# flushing old rules -- USE WITH CARE
iptables --flush
iptables --table nat --flush
# MASQUERADE each request form the
   inside to the outer world
iptables -t nat -A POSTROUTING -j
   MASQUERADE
# enable IPv4 packet forwarding in the
   kernel
echo 1 > /proc/sys/net/ipv4/ip_forward
echo "Master is now operating as router"
```

Run on workers:

```
#!/bin/bash

ENDPOINT_INTERFACE=$(cat /etc/hosts |
   grep master | awk '{print $1}')
route add default gw $ENDPOINT_INTERFACE
echo "Gateway now points to
   $ENDPOINT_INTERFACE"
```

5) Εγκατάσταση *Java:* Η Java **8** χρησιμοποιείται για την επίτευξη συμβατότητας μεταξύ των Hadoop, Flink and Spark.

```
apt-get update
apt-get install -y openjdk-8-jdk
java -version
```

6) Εγκατάσταση *Hadoop:* Το Hadoop θα χρειαστεί και για τις 2 βάσεις δεδομένων. Χρησιμοποιείται η έκδοση 2.7 και για αρχή μεταφορτώνουμε στα μηχανήματα το binary του hadoop και στην συνέχεια, θέτουμε τα environmental variables που χρειάζονται (όπως JAVA_HOME κτλ.) και κάνουμε configure τα απαιτούμενα αρχεία. Τέλος ορίζουμε ποιοί κόμβοι θα λειτουργήσουν σαν "slaves" (master, slave1, slave2). Ο namenode εκτελείται στον master και οι datanodes στον master, slave1 και slave2.

7) Εγκατάσταση *Spark:* Για την εγκατάσταση του Spark[1], κατεβάζουμε την έκδοση 2.4.4 και πραγματοποιούμε την παραμετροποίηση για τα default configuration values και τις μεταβλητές περιβάλλοντος, όπως και στην εγκατάσταση του Hadoop.

Για να τρέξουμε το spark, χρησιμοποιούμε το παρακάτω:

```
hdfs namenode -f format
start-dfs.sh
start-all.sh
```

και επαληθεύουμε την ορθότητα της εκτέλεσης χρησιμοποιώντας την εντολή **jps** σε όλους τους κόμβους.

8) Εγκατάσταση *Flink:* Για το Flink εγκαταστήσαμε την stable έκδοση 1.15.2 απο τον προτεινόμενο ιστότοπο. Με βάση τις οδηγίες του documentation για την εγκατάσταση του σε έναν standalone cluster. Μεταβάλλουμε τα conf/workers και conf/masters θέτοντας τις local IP των μηχανημάτων αντίστοιχα. Για να ξεκινήσουμε τον cluster, τρέχουμε:

```
start-cluster.sh
```

B. Μετρικές Σύγκρισης

Στόχος αυτής της υποενότητας αποτελεί η παρουσίαση των αλγορίθμων που θα χρησιμοποιηθούν για το πειραματικό τμήμα της εργασίας, και συγκεκριμένα των γνωστών και συχνά χρησιμοποιούμενων αλγορίθμων, Connected Components και PageRank. Ο πρώτος πραγματοποιεί αποσύνθεση του δικτύου εισόδου σε συνεκτικά τμήματα, ενώ ο δεύτερος ταξινομεί τις κορυφές ανάλογα με τις συνδέσεις που διαθέτουν. Και οι δύο αυτοί αλγόριθμοι επωφελούνται από τη vertex-centric λογική, λόγω της επαναληπτικής τους φύσης, καθώς και του τρόπου με τον οποίο εκτελείται η κάθε επανάληψη. Επίσης, η ροή εκτέλεσής τους αποτελεί ενδεικτική της ροής πολλών ακόμα σημαντικών αλγορίθμων γράφων, ενώ παράλληλα τα GraphX και Gelly APIs παρέχουν έτοιμες υλοποιήσεις τους. Συμπεραίνουμε επομένως ότι η χρήση τους για την πειραματική σύγκριση των εξεταζόμενων μηχανών κατανεμημένης επεξεργασίας αποτελεί μια θεμιτή και σχετικά ασφαλή επιλογή.

1) Connected Components: Με βάση τον ορισμό, ένας μη κατευθυνόμενος γράφος είναι συνεκτικός εάν, για κάθε ζεύγος κορυφών του, υπάρχει ένα μονοπάτι στο γράφο που να τις ενώνει. Αντίστοιχα, μια συνεκτική συνιστώσα (connected component) ενός μη κατευθυνόμενου γράφου αποτελείται από ένα μέγιστο (maximal) συνεκτικό υπογράφο του γράφου. Κάθε κορυφή του γράφου ανήκει σε κάποια συνεκτική συνιστώσα, που συνίσταται από όλες τις κορυφές που συνδέονται μέσω μονοπατιού με την κορυφή αυτή, καθώς και όλες τις ακμές που τις συνδέουν. Εάν ένας μη κατευθυνόμενος γράφος είναι συνεκτικός, τότε διαθέτει μόνο μία συνεκτική συνιστώσα.

Πολλά δίκτυα του πραγματικού κόσμου αποτελούνται από αρκετές συνιστώσες, με μία όμως μόνο να κυριαρχεί σε μέγεθος κι ερευνητική αξία, οπότε και είναι επιθυμητή η εύρεση της συνιστώσας αυτής. Σε κάθε περίπτωση, ο υπολογισμός των συνεκτικών συνιστωσών ενός γραφήματος αποτελεί ζητούμενο πολλών χρήσιμων εφαρμογών κι επομένως έχει εξεταστεί εκτεταμένως και ερευνητικά. Μια συνήθης προγραμματιστική προσέγγιση του παραπάνω προβλήματος είναι η εκτέλεση κάποιου αλγορίθμου αναζήτησης, όπως των αλγορίθμων αναζήτησης κατά βάθος (DFS - Depth-first Search) ή κατά πλάτος (BFS - Breadth-first search) στο γράφο εισόδου. Οι απαιτούμενες, διαδοχικές εκτελέσεις των παραπάνω αλγορίθμων, με στόχο την προσπέλαση όλων των κορυφών του γράφου μία φορά, ισούνται με το συνολικό πλήθος των συνεκτικών συνιστώσεών του. [10]

Διαφορετικά, είναι ακόμα δυνατός ο υπολογισμός των συνιστωσών ενός δικτύου, αναθέτοντας ένα component ID σε κάθε κόμβο. Η μέθοδος αυτή εφαρμόζεται μέσω label propagation, μιας διαδικασίας κατά την οποία, σε κάθε βήμα εκτέλεσης, κάθε κορυφή διαθέτει ένα label (στην περίπτωση μας component ID), το οποίο και μοιράζεται με τους γειτονές της. Αναλυτικότερα, σε μια πιθανή εκδοχή της παραπάνω προσέγγισης, κάθε κορυφή αρχικοποιεί την τιμή του label της με το vertex ID της. Στη συνέχεια, κάθε κορυφή μεταδίδει την πληροφορία αυτήν στις γειτονικές της κορυφές κι έπειτα ανανεώνει την ετικέτα της με βάση την ελάχιστη τιμή των labels που δέχτηκε, και της δική της. Σημειώνουμε ότι μια κορυφή ενημερώνει τους γείτονές της μόνο σε περίπτωση αλλαγής της ετικέτας της. Τελικά, με τον παραπάνω τρόπο γίνεται διαχωρισμός όλων των κορυφών σε components, με την κάθε συνιστώσα να διαθέτει ως component ID το ελάχιστο vertex ID που σημειώνεται ανάμεσα στις κορυφές που περιέχει.

Η παραπάνω μέθοδος προτείνεται για vertex-centric προγραμματιστικά μοντέλα όπως το Pregel, όπου η εκτέλεση πραγ-

[1] Ακολουθήθηκαν οι οδηγίες απο το μάθημα Προχωρημένα Θέματα Βάσεων Δεδομένων

ματοποιείται με επαναληπτικό τρόπο και κάθε επανάληψη διαθέτει ένα στάδιο ανανέωσης των κόμβων, ακολουθούμενο από μια φάση επικοινωνίας. Πράγματι, ο Connected Components αλγόριθμος που παρέχεται από τα APIs των υπό μελέτη frameworks Gelly και GraphX, υλοποιεί τη μέθοδο που παρουσιάστηκε. Στη συνέχεια, ο αλγόριθμος αυτός παρουσιάζεται σε μορφή ψευδοκώδικα, για τα προγραμματιστικά μοντέλα Pregel και GAS. [11]

Algorithm 1 Connected Components in Pregel Model

]

function COMPUTE($vertex$, $messages$)
  if $getSuperstep() = 0$ **then**
    $vertex.component \leftarrow vertex.id$
    $changed \leftarrow true$
  else
    $minMsg \leftarrow getMinimum(messages)$
    $changed \leftarrow false$
    if $minMsg < vertex.component$ **then**
      $changed \leftarrow true$
      $vertex.component \leftarrow minMsg$
    end if
  end if
  if $changed$ **then**
    SENDTOALL($vertex.component$)
  end if
end function

Algorithm 2 Connected Components in GAS Model

function INIT($vertex$)
  $vertex.component \leftarrow vertex.id$
end function
function GATHER($vertex$, $edge$)
  return $vertex.component$
end function
function GATHERSUM($v1$, $v2$)
  return $minimum(v1, v2)$
end function
function APPLY($vertex$, $minimum$)
  $changed \leftarrow false$
  if $verex.label > minimum$ **then**
    $changed \leftarrow true$
    $vertex.component \leftarrow minimum$
  end if
end function
function SCATTER($vertex$, $edge$)
  if $changed$ **then**
    SIGNAL($edge.target()$)
  end if
end function

2) PageRank: Ο PageRank αποτελεί έναν γνωστό και δημοφιλή αλγόριθμο που μπορεί να χρησιμοποιηθεί για την ταξινόμηση των κορυφών ενός γράφου, ανάλογα με τη δομική και σχετική τους αξία. Προτάθηκε από τους ιδρυτές της Google, Brin και Page, το 1998 [12] για την ταξινόμηση των ιστότοπων του World Wide Web, και στη συνέχεια αποτέλεσε το βασικό εργαλείο για την κατάταξη των αποτελεσμάτων αναζήτησης της Google. Παραλλαγές του αλγορίθμου, σε συνδυασμό και με άλλες μετρικές, χρησιμοποιούνται ακόμα και σήμερα από την εταιρεία.
Στην πράξη, ο PageRank αλγόριθμος συνίσταται στον υπολογισμό ενός σκορ $P(v)$ για κάθε κόμβο v του δικτύου, με βάση την επαναληπτική σχέση: $P_{i+1}(v) = \frac{1-d}{N} + d \cdot \sum_{(v,u)\epsilon E} \frac{P_i(u)}{deg(u)}$, όπου i η τρέχουσα επανάληψη, N το συνολικό πλήθος συνδέσεων και d η παράμετρος damping factor. Λαμβάνοντας υπόψη ότι το PageRank υπολογίζεται προσομοιώνοντας τη συμπεριφορά ενός χρήστη που μπαίνει τυχαία σε μια σελίδα και κάνει κλικ σε συνδέσμους, εφαρμόζουμε αυτόν τον παράγοντα απόσβεσης d ως την πιθανότητα ο χρήστης να εγκαταλείψει μια σελίδα και να μην συνεχίσει να ακολουθεί άλλους συνδέσμους της. Η χρήση της παραμέτρου d εγγυάται επίσης ότι ο αλγόριθμος δεν θα εγκλωβιστεί σε κάποια κορυφή-καταβόθρα (sink), δηλαδή σε κάποια κορυφή δίχως εξερχόμενες ακμές (συνδέσμους).
Παρακάτω ακολουθούν ενδεικτικές υλοποιήσεις του αλγορίθμου PageRank για το Pregel και το GAS μοντέλο, με τη χρήση ψευδοκώδικα: [11]

Algorithm 3 PageRank in Pregel Model

function COMPUTE($vertex$, $messages$)
  if $getSuperstep() = 0$ **then**
    $vertex.rank \leftarrow 1$
  else
    $newScore \leftarrow \frac{1-d}{N} + d * sum(messages)$
    $delta \leftarrow newScore - vertex.rank$
    $vertex.rank \leftarrow newScore$
    if $|delta| > TOLERANCE$ **then**
      $converged \leftarrow false$
    else
      $converged \leftarrow true$
    end if
    SETAGGREGATOR($CONV, converged$)
  end if
  if GETAGGREGATOR($CONV$) **then**
    VOTETOHALT()
  else
    SENDTOALLEDGES($vetex.rank/vertex.numEdges$)
  end if
end function

Algorithm 4 PageRank in GAS Model

```
function GATHER(vertex, edge)
    return edge.source.score/edge.source.numOutEdges
end function
function APPLY(vertex, sum)
    newScore ← (1−d)/N + d ∗ sum
    delta ← newScore − vertex.rank
    vertex.score ← newScore
    if |delta| > TOLERANCE then
        converged ← false
    else
        converged ← true
    end if
end function
function SCATTER(vertex, edge)
    if notconverged then
        SIGNAL(edge.target())
    end if
end function
```

C. Σύνολα Δεδομένων

Για τη διεξαγωγή των πειραμάτων χρησιμοποιήθηκαν συνθετικά δεδομένα, τα οποία παράχθηκαν με χρήση του Python πακέτου igraph. Το πακέτο αυτό προτείνεται για την κατασκευή και τη διαχείριση μεγάλων γράφων, ενώ ακόμα προσφέρει πληθώρα υλοποιημένων graph generators, με τους οποίους μπορούν να δημιουργηθούν διάφοροι γνωστοί τύποι γραφημάτων. Στα πλαίσια της εφαρμογής μας, επιλέξαμε η σύνθεση των γράφων να γίνει με βάση τα μοντέλα Erdős–Rényi και Barabási–Albert, που παρουσιάζονται στη συνέχεια.

1) Στοχαστικοί γράφοι - μοντέλο *Erdős–Rényi:* Το μοντέλο Erdős–Rényi προτάθηκε από τους Ούγκους μαθηματικούς Erdős και Rényi το 1959, για την παραγωγή συνόλων τυχαίων γράφων. Με βάση το μοντέλο αυτό, όλοι οι γράφοι με ένα καθορισμένο πλήθος κορυφών και ακμών είναι εξίσου πιθανοί. Ισοδύναμα, η ύπαρξη κάθε ακμής μεταξύ δύο κορυφών είναι εξίσου πιθανή και ανεξάρτητη από τις υπόλοιπες ακμές.
Στην εκδοχή $G(n,p)$ του μοντέλου, η πιθανότητα δημιουργίας κάθε ακμής, σε ένα γράφο n ακμών, περιγράφεται από την παράμετρο p. Δεδομένων αυτών των χαρακτηριστικών, ο αναμενόμενος συνολικός αριθμός ακμών είναι $\binom{n}{2}p$. Παρατηρούμε επομένως, ότι καθώς το p αυξάνεται από το 0 στο 1, οι τυχαίοι γράφοι $G(n,p)$ γίνονται όλο και πιο πυκνοί. Αντίστοιχα, για $p = 0.5$, όλοι οι $2^{\binom{n}{2}}$ δυνατοί γράφοι είναι εξίσου πιθανοί.
Το μοντέλο Erdős–Rényi μπορεί να χρησιμοποιηθεί για την προσομοίωση πραγματικών δικτύων με απρόβλεπτη ή σχεδόν τυχαία ανάπτυξη, όπως των κοινωνικών δικτύων. Το γεγονός ότι οι ιδιότητες των γράφων $G(n,p)$ έχουν μελετηθεί εκτενώς μαθηματικά, και είναι γνωστές για διάφορες τιμές του p, έχει επίσης συνεισφέρει στην ευρεία χρήση του μοντέλου για ερευνητικούς σκοπούς. [13]

2) Scale-free γράφοι - μοντέλο *Barabási–Albert:* Για τη δημιουργία πιο ρεαλιστικών γραφημάτων συχνά χρησιμοποιούνται αλγόριθμοι όπως ο Barabási–Albert (ή BA). Το μοντέλο αυτό ανήκει στην κατηγορία των λεγόμενων scale-free ή power-law μοντέλων, μοντέλων που προσομοιάζουν σημαντική μερίδα τεχνητών, αλλά και φυσικών, σύγχρονων δικτύων. Κύρια χαρακτηριστικά των δικτύων αυτών αποτελούν η συνεχιζόμενη ανάπτυξη (growth) και η προνομιακή διαδικασία προσκόλλησης (preferential attachment). Λόγω αυτών των χαρακτηριστικών, τα scale-free δίκτυα συνεχίζουν να επεκτείνονται με την προσθήκη νέων κόμβων, ενώ παράλληλα, οι καινούργιοι αυτοί κόμβοι τείνουν να συνδέονται με κορυφές του γράφου με ήδη υψηλό αριθμό ακμών (βαθμό). Η δεύτερη αυτή ιδιότητα οδηγεί στο σχηματισμό των ονομαζόμενων hubs, δηλαδή κορυφών με πολύ υψηλό βαθμό, σε σχέση με αυτόν των υπόλοιπων κορυφών του δικτύου. Η ύπαρξη hubs είναι το κύριο διαφοροποιό στοιχείο των free-scale γράφων σε σχέση με τους τυχαίους, ενώ επίσης αντικατοπρίζει την πραγματικότητα πολλών ανθρωπογενών και φυσικών δικτύων. Χαρακτηριστικό παράδειγμα τα κοινωνικά δίκτυα, όπου άτομα με ήδη υψηλό πλήθος διασυνδέσεων πραγματοποιούν πιο εύκολα νέες γνωριμίες, σε σχέση με άλλους, λιγότερο γνωστούς στο δίκτυο, ανθρώπους.
Μαθηματικά, η κατανομή βαθμών σε ένα scale-free ή power-law δίκτυο, περιγράφεται από τη σχέση $P(k) \sim k^{-\gamma}$, όπου $P(k)$ το ποσοστό των κορυφών του γράφου με βαθμό k, και γ η λεγόμενη degree component παράμετρος. Στην περίπτωση του Barabási–Albert μοντέλου ισχύει ότι $\gamma = 3$. Τέλος, αναφέρουμε ότι ο αλγόριθμος BA μπορεί να χρησιμοποιηθεί για την περιγραφή αξιοσημείωτης ποικιλίας πραγματικών δικτύων, όπως αυτών που συναντώνται στο Internet, στο World Wide Web (www), καθώς και σε citation και βιολογικά δίκτυα. [14]

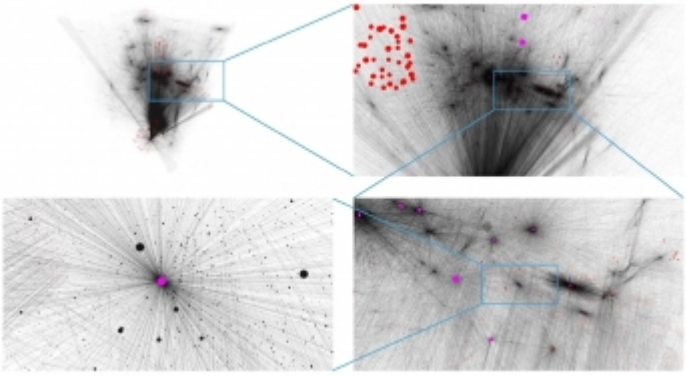


Fig. 1: Οπτικοποίηση τμημάτων του World Wide Web, 1998. Στις εικόνες, με πιο σκούρο χρώμα, διακρίνονται κορυφές-hubs. [14]

Σημειώνουμε ότι με κάθε μοντέλο παραγωγής γράφων παράχθηκαν τρία είδη δικτύων, διαβαθμιζόμενου μεγέθους. Συγκεκριμένα, η πρώτη κατηγορία (μικρό μέγεθος) περιλαμβάνει γράφους με 100 χιλιάδες κορυφές και περίπου

500000 ακμές. Αντίστοιχα, η 2η κατηγορία (μεσαίο μέγεθος) συνίσταται από γράφους με 500 χιλιάδες κορυφές και περίπου 12.5 εκατομμύρια ακμές. Τέλος, στην τρίτη ομάδα περιλαμβάνονται γραφήματα με 1 εκατομμύριο κορυφές και περίπου 50 εκατομμύρια ακμές.

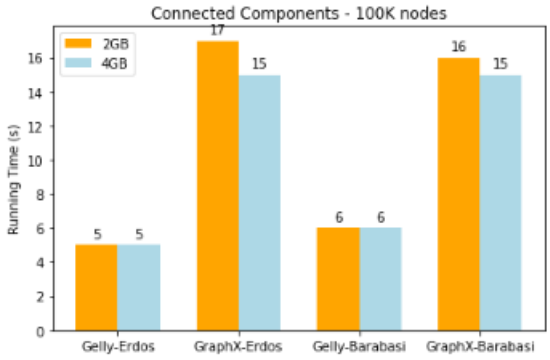
D. Αποτελέσματα

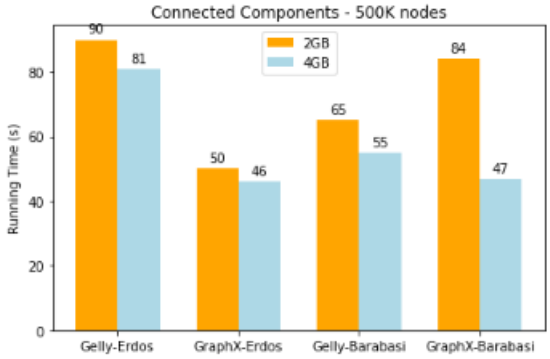
Για την σύγκριση των δύο συστημάτων (Graphx, Flink) μετρήθηκε με τη χρήση εντολών συστήματος, ο χρόνος εκτέλεσης (running time) σε sec, καθώς και η χρήση του πυρήνα (CPU usage) ενός worker κόμβου. Για κάθε σύστημα, τα πειράματα έγιναν παραμετροποιώντας την τιμή της μνήμης του Worker, στην περίπτωση του Spark, και του TaskManager, στην περίπτωση του Flink. Στην πρώτη περίπτωση η μνήμη των Workers/TaskManager τέθηκε παντού 2gb, ενώ στην δεύτερη 4gb. Η εντολή που χρησιμοποιήθηκε για την μέτρηση των μετρικών ήταν η:

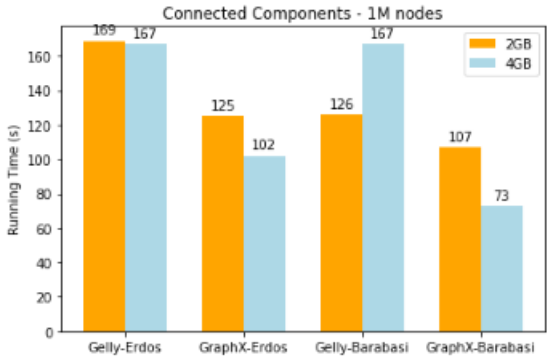
```
./start_pidstat.sh -o <output_file_name>
```

1) Σύγκριση χρόνων εκτέλεσης*:* Παρακάτω, σε ραβδογράμματα, και ομαδοποιημένα κατά framework και είδος γράφου, παρουσιάζονται οι χρόνοι εκτέλεσης των προγραμμάτων για κάθε αλγόριθμο και μέγεθος γραφήματος.

Αλγόριθμος Connected Components:



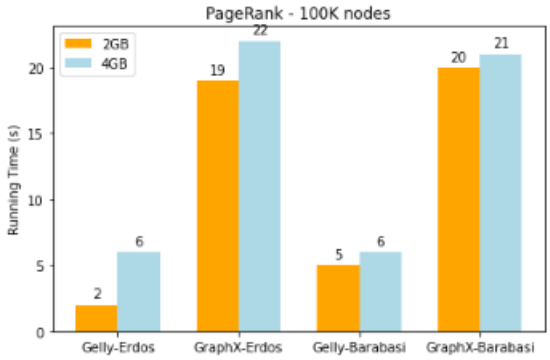


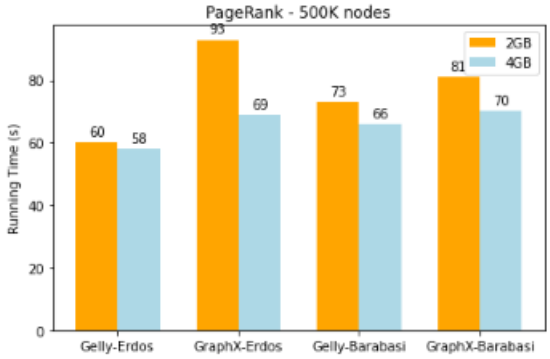


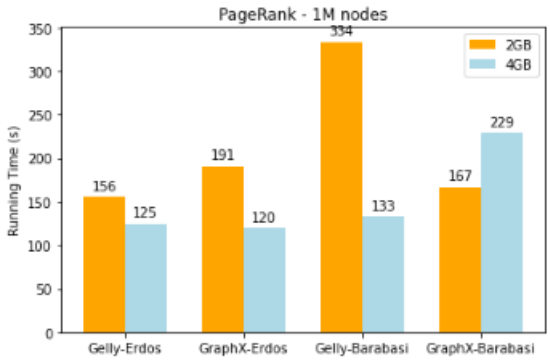
Παρατηρήσεις:

Για τον αλγόριθμο Connected Components παρατηρούμε αρχικά ότι ο χρόνος εκτέλεσης αυξάνεται, καθώς μεγαλώνει το μέγεθος του γράφου εισόδου για κάθε συνδυασμό framework και τύπο γράφου, όπως και αναμενόταν. Ακόμα, στις περισσότερες περιπτώσεις, η αύξηση της διαθέσιμης μνήμης από 2GB σε 4GB, φαίνεται να ευνοεί αισθητά την ταχύτητα της εκτέλεσης. Όσον αφορά τη σύγκριση των GraphX και Gelly frameworks, παρατηρούμε ότι για αριθμό κόμβων 100K και 500K αντίστοιχα, δεν φαίνεται να υπερτερεί ξεκάθαρα κάποιο από τα δύο, γεγονός που μπορεί να αιτιολογηθεί από το σχετικά περιορισμένο πλήθος κορυφών στις περιπτώσεις αυτές. Παρόλα αυτά, για το μέγιστο μέγεθος που χρησιμοποιήθηκε, το GraphX φαίνεται να επιτυγχάνει την ταχύτερη εκτέλεση για όλους τους συνδυασμούς μνήμης και graph generator. Όσον αφορά τον τύπο γράφου (τυχαίο ή scale-free), δεν φαίνεται να προκύπτει κάποιο οριστικό πόρισμα από τη συγκεκριμένη μερίδα πειραμάτων.

Αλγόριθμος PageRank:





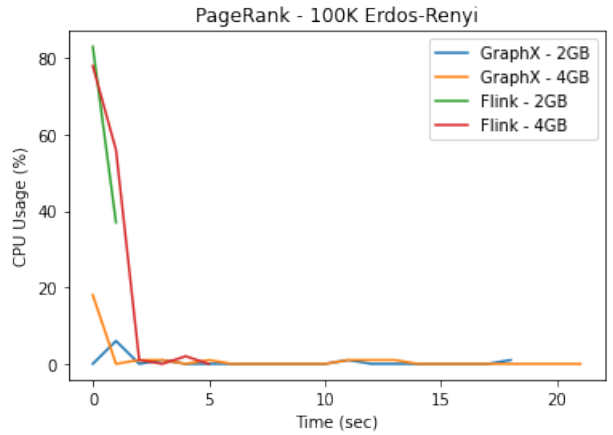


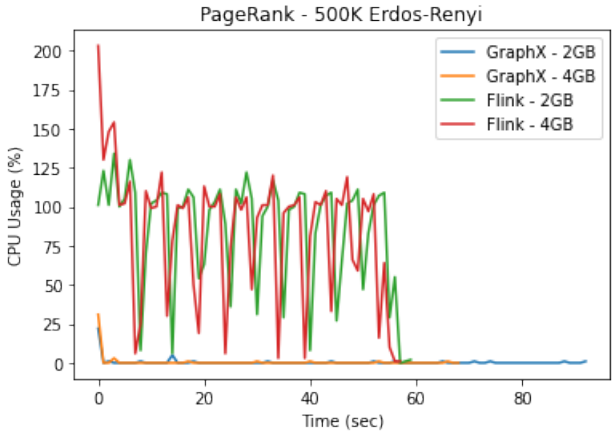
Παρατηρήσεις:

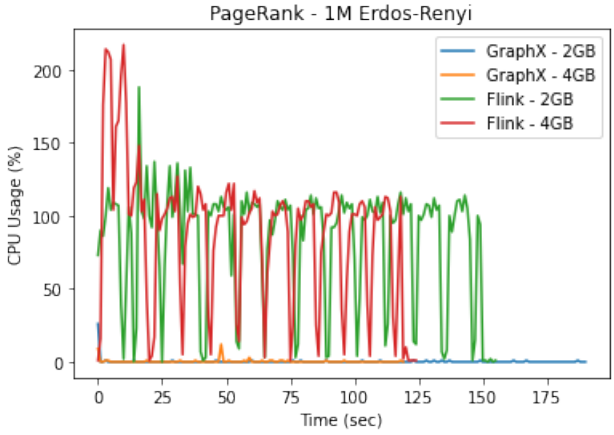
Όπως και προηγουμένως, ο μέσος χρόνος εκτέλεσης σε κάθε περίπτωση αυξάνεται παράλληλα με την αύξηση του μεγέθους του γράφου εισόδου. Ακόμα, η παραχώρηση περισσότερης RAM μνήμης φαίνεται να ευνοεί την εκτέλεση, καθώς το συνολικό πλήθος κορυφών αυξάνει. Όσον αφορά τη σχετική επίδοση των GraphX και Gelly, το δεύτερο φαίνεται να αποδίδει καλύτερα για αριθμό κόμβων 100 και 500 χιλιάδες. Για πλήθος κορυφών 1 εκατομμύριο, τα αποτελέσματα εμφανίζονται περισσότερο ανάμεικτα, με το GraphX να υπερτερεί σημαντικά στην περίπτωση όπου η παραχωρούμενη μνήμη είναι 2GB και ο τύπος του γράφου είναι scale-free.

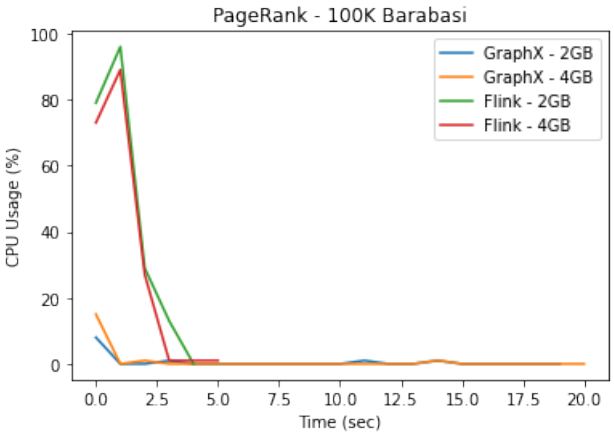
2) Σύγκριση *CPU Usage:* Στη συνέχεια, ομαδοποιημένη κατά framework και μέγεθος μνήμης, παρουσιάζεται η CPU usage κατά την εκτέλεση των προγραμμάτων για κάθε αλγόριθμο, μέγεθος και τύπο γραφήματος.

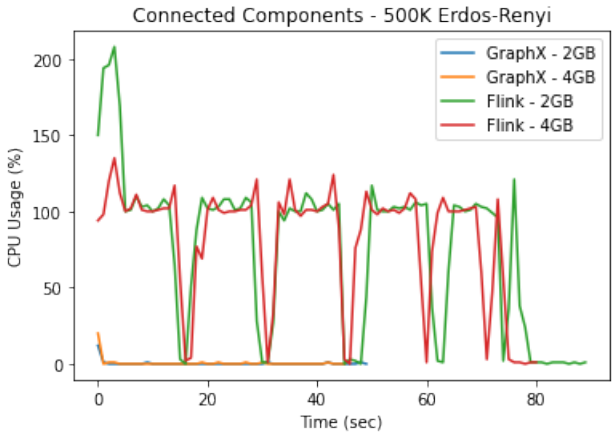
Αλγόριθμος PageRank:

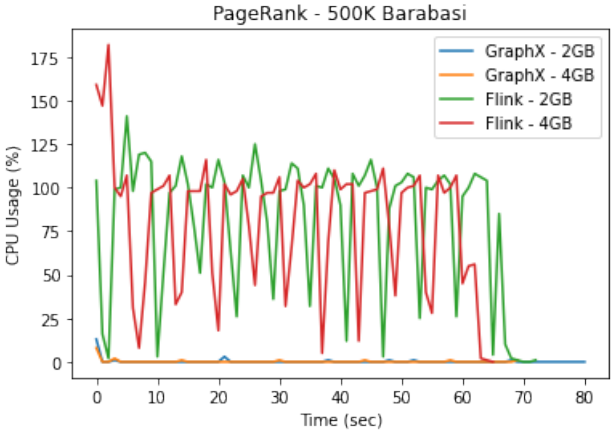


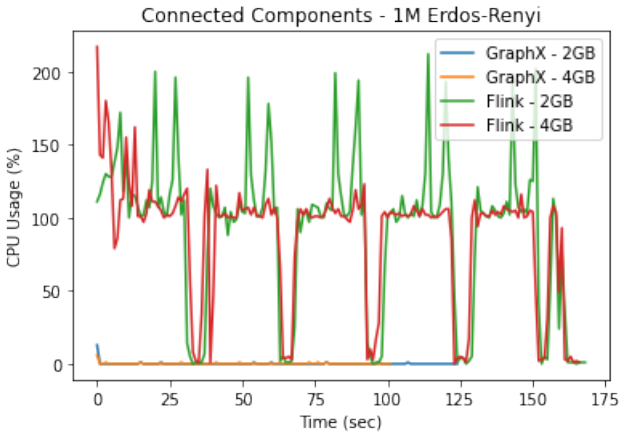


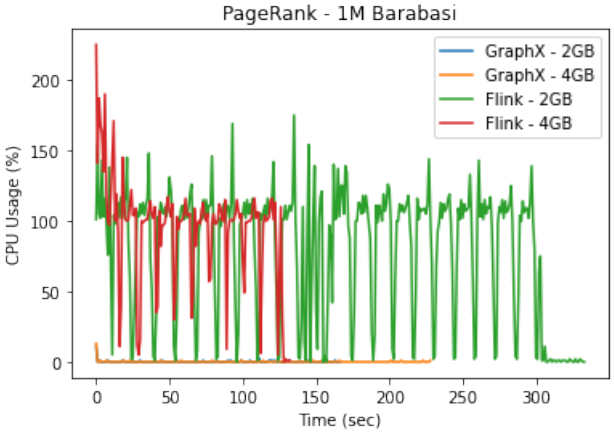


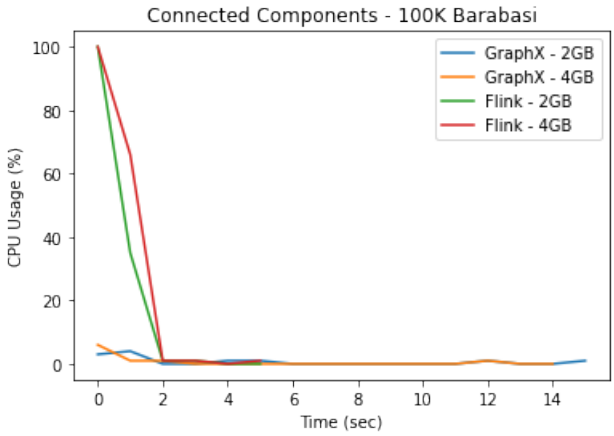




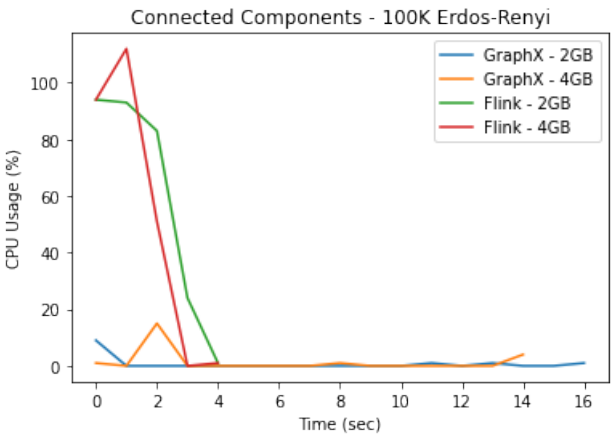


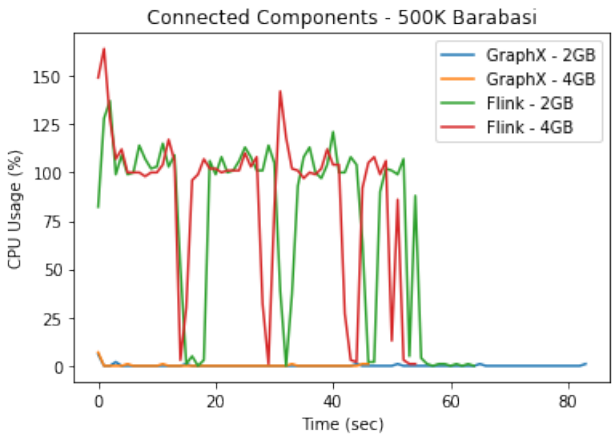


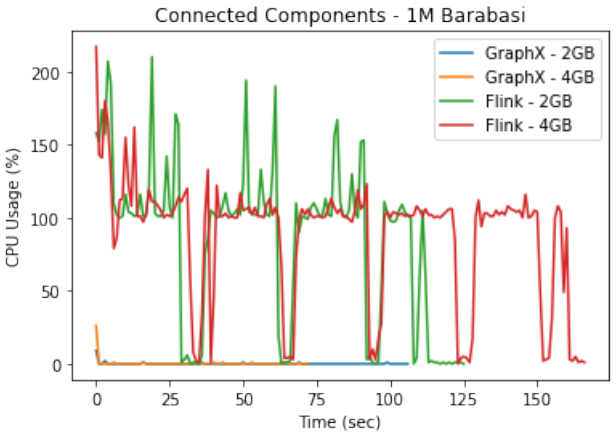




Αλγόριθμος Connected Components:







Σε αυτήν την περίπτωση ενδιαφέρον έχει η μελέτη της τάξης μεγέθους χρησιμοποίησης CPU ανά περίπτωση framework, αλγορίθμου, γράφου εισόδου, αλλά και παραχωρούμενης RAM μνήμης. Ακόμα, οι παρατηρούμενες διακυμάνσεις μπορούν να φανερώσουν πληροφορίες σχετικά με τον επαναληπτικό τρόπο εκτέλεσης και να παρέχουν μια καλύτερη ιδέα σχετικά με το πως εκτελείται ο διαμοιρασμός εργασιών εντός του cluster και η εκτέλεση των supersterps. Σημειώνουμε τέλος, ότι στις περιπτώσεις όπου παρατηρείται CPU usage μεγαλύτερη από 100%, πραγματοποιείται επεξεργασία της αντίστοιχης διεργασίας με χρήση άνω του ενός πυρήνων CPU.

V. ΣΥΖΗΤΗΣΗ-ΜΕΛΛΟΝΤΙΚΕΣ ΠΡΟΕΚΤΑΣΕΙΣ

Εν κατακλείδι, στα παραπάνω πειράματα, οι μηχανές κατανεμημένης επεξεργασίας GraphX και Flink σημειώνουν συγκρίσιμους χρόνους υπολογισμού, χωρίς κάποια από τις δύο να υπερτερεί με ξεκάθαρο τρόπο υπέρ της άλλης. Παρόλα αυτά, η παραπάνω πειραματική διαδικασία μας προσέφερε χρήσιμες γνώσεις και μας εξοικείωσε με τον τρόπο που τα δύο αυτά frameworks λειτουργούν και επεξεργάζονται ογκώδη δεδομένα. Ειδικά τα διαγράμματα CPU χρησιμοποίησης μπορούν να προσφέρουν μια ιδέα για τον τρόπο εκτέλεσης vertex-centric προγραμματιστικών τεχνικών σε κατανεμημένα συστήματα. Αντίστοιχα, η μεταβολή των παραμέτρων μνήμης RAM ανά υπολογιστικό κόμβο μπορεί να μας πληροφορήσει για την κατανάλωση και τη διαχείριση μνήμης ανά framework, καθώς και για τον τρόπο με τον οποίο λειτουργούν τα συστήματα αυτά σε συνθήκες περιορισμένων διαθέσιμων πόρων.

Μελλοντικά, σκόπιμη θα ήταν η χρήση μεγαλύτερων δεδομένων εισόδου για να είναι δυνατή η πιο ουσιαστική σύγκριση των παραπάνω frameworks, μέσω της τελικής διάκρισης ενός με σταθερά καλύτερες επιδόσεις. Έτσι θα είναι δυνατή η διεξαγωγή πιο εδραιωμένων κρίσεων σχετικά με τα συστήματα διαχείρισης εικονικής μνήμης, τους preprocessing χρόνους και τη διαχείριση του δικτύου, μεταξύ άλλων, από τα συστήματα αυτά. Ενδεικτικά αναφέρουμε ότι το GraphX χρησιμοποιεί διάφορες δομές δεδομένων όπως bitmask, routing, table και local indexes για την επιτάχυνση των επαναλήψεων του αλγορίθμου. Παρόλα αυτα, στα πειράματα μας, η βελτίωση αυτή καθίσταται αμελητέα λόγω περιορισμένων δεδομένων. Τέλος, η αύξηση των κορυφών και ακμών του γράφου θα μπορούσε να προσφέρει χρήσιμες πληροφορίες σχετικά με την καλύτερη προσέγγιση δομής για την περιγραφή της κλάσης των γράφων (RDDs vs DataFrames).

REFERENCES

[1] Jeffrey Dean and Sanjay Ghemawat. (2008). MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1, 107–113. https://doi.org/10.1145/1327452.1327492

[2] Apache Hadoop. https://hadoop.apache.org/

[3] Apache MapReduce. IBM. https://www.ibm.com/topics/mapreduce

[4] Bartosz Konieczy. (2018). Vertex-centric graph processing. Waiting for Code. https://www.waitingforcode.com/graphs/vertex-centric-graph-processing/read

[5] Malewicz, G., Austern, M.H., Bik, A.J., Dehnert, J.C., Horn, I., Leiser, N., & Czajkowski, G. (2010). Pregel: a system for large-scale graph processing. Proceedings of the 2010 ACM SIGMOD International Conference on Management of data.

[6] Coyer, A. (2015). Pregel: A System for Large-Scale Graph Processing. The Morning Paper. https://blog.acolyer.org/2015/05/26/pregel-a-system-for-large-scale-graph-processing/

[7] Coyer, A. (2015). PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. The Morning Paper. https://blog.acolyer.org/2015/05/29/powergraph-distributed-graph-parallel-computation-on-natural-graphs/

[8] GraphX Programming Guide. Apache Spark 3.3.0. https://spark.apache.org/docs/latest/graphx-programming-guide.html

[9] Introducing Gelly: Graph Processing with Apache Flink. (2015). Flink. https://flink.apache.org/news/2015/08/24/introducing-flink-gelly.html

[10] Connected Components in a Graph. (2020). Baeldung. https://www.baeldung.com/cs/graph-connected-components

[11] Koch, J., Staudt, C.L., Vogel, M. (2016). An empirical comparison of Big Graph frameworks in the context of network analysis. Soc. Netw. Anal. Min. 6, 84. https://doi.org/10.1007/s13278-016-0394-1

[12] S. Brin, L. Page, R. Motwami, T. Winograd. (1998). The PageRank Citation Ranking: Bringing Order to the Web. Stanford University Technical Report. http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf

[13] Hopcroft, J. (2006). Erdos-Renyi Model. CS485 Lecture 01. http://www.cs.cornell.edu/courses/cs485/2006sp/lecture

[14] Barabasi, A. (2014). Network Science, The Barabasi Model. http://networksciencebook.com/chapter/5