

ΙΟΝΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



– Πτυχιακή Εργασία –

Ανάλυση Μεγάλου Όγκου Δεδομένων Γράφων από Ανάλυση
Ανοικτού Κώδικα

Δημήτριος Γιαμπούρης Π2014206

Επιβλέπων: Δημήτριος Τσουμάκος

19 Ιουνίου 2020

Επιβλέπων

Δημήτριος Τσουμάκος, Αναπληρωτής Καθηγητής,
Ιόνιο Πανεπιστήμιο

Τριμελής Επιτροπή

Δημήτριος Τσουμάκος, Αναπληρωτής Καθηγητής,
Ιόνιο Πανεπιστήμιο

Θεόδωρος Ανδρόνικος, Αναπληρωτής Καθηγητής,
Ιόνιο Πανεπιστήμιο

Σπύρος Σιούτας, Καθηγητής,
Πανεπιστήμιο Πατρών

Περίληψη

Η εποχή της πληροφορίας έχει επιφέρει την χρήση ηλεκτρονικών συσκευών και υπηρεσιών σε ευρύτατη κλίμακα. Οι ηλεκτρονικές αγορές, η αναζήτησης πληροφοριών στο διαδίκτυο και το απανταχού υπολογίζειν είναι πλέον ένα μέρος της καθημερινότητάς μας. Η τεχνολογία η οποία πλαισιώνεται από δεδομένα αποτελεί κινητήριο μοχλό διαφόρων κρίσιμων κομματιών της κοινωνίας μας, όπως αυτών της οικονομίας και της υγείας. Σε αυτά τα δεδομένα καλούμαστε να βασίσουμε όλο και πιο σημαντικές αποφάσεις.

Όπως είναι γνωστό, με την διάδοση της τεχνολογίας ο όγκος προς συλλογή, επεξεργασία και αφομοίωση είναι ήδη τεράστιος και τείνει να αυξάνεται χρόνο με τον χρόνο. Όπως είναι φυσικό η διαχείριση αυτού του όγκου πληροφοριών παρουσιάζει διάφορες προκλήσεις. Με τις προκλήσεις αυτές ασχολείται ο τομέας των μεγάλων δεδομένων.

Παράλληλα, με την εμφάνιση όλο και περισσότερων λογισμικών ανοικτού κώδικα και την όλο και πιο διαδεδομένη χρήση αυτών, κρίθηκε θεμιτό, στο πλαίσιο αυτής της πτυχιακής εργασίας, να δημιουργηθεί ένα σύστημα συλλογής, επεξεργασίας και μελέτης δεδομένων που προέρχονται από πηγαίο κώδικα ανοικτού λογισμικού. Προκειμένου να εξεταστούν οι δυνατότητες αυτού του συστήματος, έγινε δοκιμή αυτού με δεδομένα από υπαρκτό και πραγματικό πηγαίο κώδικα από εδραιωμένα πακέτα λογισμικού.

Μετά το πέρας των δοκιμών έγινε ξεκάθαρο ότι η αντιμετώπιση του κώδικα ως δεδομένα μεγάλου όγκου είναι δόκιμη, καθώς προσδίδει αξία στο πεδίο της ανάλυσης κώδικα, παρ' όλους τους περιορισμούς του προτεινόμενου συστήματος.

Περιεχόμενα

A´	Εισαγωγή	1
A´.1	ΟΡΙΣΜΟΣ	1
A´.1.1	Volume	1
A´.1.2	Variety	2
A´.1.3	Velocity	2
A´.1.4	Veracity	2
A´.1.5	Variability	3
A´.1.6	Value	3
A´.2	Η ΑΝΑΠΤΥΞΗ ΤΩΝ ΜΕΓΑΛΩΝ ΔΕΔΟΜΕΝΩΝ	3
A´.3	Η ΣΗΜΑΣΙΑ ΤΩΝ ΜΕΓΑΛΩΝ ΔΕΔΟΜΕΝΩΝ	5
B´	Μοντέλα Επεξεργασίας	6
B´.1	MAPREDUCE	6
B´.1.1	Παράδειγμα Υπολογισμού	7
B´.1.2	Μοντέλο Εκτέλεσης	7
B´.2	VERTEX-CENTRIC BSP	8
B´.2.1	Pregel	9
Γ´	Κώδικας ως Μεγάλα Δεδομένα	12
Γ´.1	Η ΔΙΑΔΟΣΗ ΤΟΥ ΑΝΟΙΧΤΟΥ ΚΩΔΙΚΑ	13
Γ´.2	Ο ΣΚΟΠΟΣ ΤΗΣ ΑΝΑΛΥΣΗΣ ΚΩΔΙΚΑ	13
Δ´	NoSQL και Βάσεις Δεδομένων Γράφων	15
Δ´.1	ΜΟΝΤΕΛΑ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ ΓΡΑΦΩΝ	15

Περιεχόμενα	iii
Δ'.1.1	Edge-labelled Graphs 16
Δ'.1.2	Property Graphs 17
Δ'.2	CYPHER ΚΑΙ NEO4J 18
Δ'.2.1	Γραμμικά Ερωτήματα 19
Δ'.2.2	Εύρεση Μοτίβων 19
Δ'.2.3	Τροποποίηση Δεδομένων 19
Δ'.2.4	Αρχιτεκτονική Συστήματος 20
Ε'	Το Σύστημα Ανάλυσης Κώδικα Scrubj 21
Ε'.1	Η ΔΟΜΗ ΕΝΟΣ COMPILER 22
Ε'.1.1	Front End 23
Ε'.1.2	Scanner 24
Ε'.1.3	Parser 24
Ε'.1.4	Intermediate Representations 27
Ε'.2	ΙΔΙΑΙΤΕΡΟΤΗΤΕΣ ΤΟΥ GCC 29
Ε'.2.1	GENERIC 29
Ε'.2.2	GIMPLE 29
Ε'.2.3	Control Flow Graph 29
Ε'.3	SCRUBJ 30
Ε'.3.1	Μοντέλο Επικοινωνίας 31
Ε'.3.2	Collector 32
Ε'.3.3	Receiver 38
Γ'	Δοκιμή του Συστήματος 44
Γ'.1	ΠΑΡΑΔΕΙΓΜΑ ΧΡΗΣΗΣ 45
Γ'.1.1	Ερωτήματα με την Cypher 46
Γ'.2	ΤΕΛΙΚΟΣ ΑΠΟΛΟΓΙΣΜΟΣ 54
Βιβλιογραφία	57

Κατάλογος Σχημάτων

B'.1	Απεικόνιση του μοντέλου εκτέλεσης MapReduce.	8
B'.2	Παράδειγμα μέγιστης τιμής. Οι διακεκομμένες γραμμές αντιπροσωπεύουν τα μηνύματα. Οι σκιασμένοι κόμβοι έχουν εκδώσει ψήφο αδρανοποίησης.	11
Δ'.3	Παράδειγμα ενός απλού γράφου.	16
Δ'.4	Παράδειγμα ενός edge-labelled graph.	17
Δ'.5	Παράδειγμα ενός property graph.	18
Δ'.6	Μια βασική επισκόπηση της αρχιτεκτονικής συστήματος της Neo4j.	20
Ε'.7	Μια απλοποιημένη ροή λειτουργίας ενός compiler.	22
Ε'.8	Ροή λειτουργίας ενός τριφασικού compiler.	23
Ε'.9	Η δομή ενός τυπικού compiler.	23
Ε'.10	Το αντίστοιχο συντακτικό δέντρο.	27
Ε'.11	Παράδειγμα ενός CFG.	28
Ε'.12	Η δομή του συστήματος Scrubj.	30
Ε'.13	Μια τοπολογία publisher/subscriber υπό το ZeroMQ [30].	31

Κεφάλαιο Α΄

Εισαγωγή

TΑ big data είναι μια αφηρημένη έννοια. Εκτός από τον όγκο των δεδομένων, έχουν και άλλα χαρακτηριστικά που ορίζουν τη διαφορά τους από τα απλά μαζικά δεδομένα, καθώς απαιτούν νέες αρχιτεκτονικές, τεχνικές, αλγόριθμους και μεθόδους ανάλυσης για την διαχείρισή τους και την εξαγωγή αξίας από αυτά [14].

Α΄.1 Ορισμός

Τα μεγάλα δεδομένα ή αλλιώς big data, είναι ένα πεδίο έρευνας που μελετά μεγέθη δεδομένων που δεν μπορούν να συλληφθούν, διαχειριστούν και να επεξεργαστούν συμβατικοί υπολογιστές σε κοινότυπη έκταση [47]. Δεδομένων αυτών των όρων, οι τεχνολογίες μεγάλων δεδομένων περιγράφουν μία νέα γενιά από τεχνολογίες και αρχιτεκτονικές, σχεδιασμένες να εξάγουν με οικονομικό τρόπο αξία από πολύ μεγάλο όγκο και ποικιλία δεδομένων, ενσωματώνοντας υψηλής ταχύτητας καταγραφή, ανακάλυψη και ανάλυση [25]. Τα μεγάλα δεδομένα μπορούν να περιγραφούν αρχικά από το μοντέλο 3V, δηλαδή volume, velocity, variety, και από μερικές διαστάσεις που προστέθηκαν αργότερα όπως τις veracity, variability και value.

Α΄.1.1 Volume

Ο όγκος ή Volume αναφέρεται στο μέγεθος των δεδομένων. Τα μεγέθη των μεγάλων δεδομένων υπολογίζονται συνήθως στην κλίμακα των πολλαπλών terabytes και petabytes. Οι ορισμοί του όγκου μεγάλων δεδομένων είναι σχετικοί και διαφέρουν κατά παράγοντες χρόνου

και τύπων δεδομένων. Κάτι που μπορεί να θεωρείται ως μεγάλα δεδομένα σήμερα, μπορεί να μην πληρεί τις προϋποθέσεις στο μέλλον, καθώς οι δυνατότητες αποθήκευσης αυξάνονται, επιτρέποντας ακόμη μεγαλύτερα σύνολα δεδομένων να αποθηκεύονται [24].

A'.1.2 Variety

Η ποικιλία ή Variety αναφέρεται στην δομική ανομοιογένεια ενός συνόλου δεδομένων. Διάφορες τεχνολογικές εξελίξεις επιτρέπουν σε εταιρίες και οργανισμούς να χρησιμοποιούν διαφόρων τύπων δομημένα, ημι-δομημένα και μη δομημένα δεδομένα. Παραδείγματα δομημένων δεδομένων είναι οι βάσεις δεδομένων και τα υπολογιστικά φύλλα. Κείμενο, φωτογραφίες, ηχητικά αρχεία και οπτικοακουστικά αρχεία είναι παραδείγματα μη δομημένων δεδομένων. Μεταξύ αυτών των δύο άκρων βρίσκονται τα ημι-δομημένα αρχεία, τα οποία είναι συνήθως εκφρασμένα με κάποιο μορφότυπο όπως η Extensible Markup Language ή η JavaScript Object Notation [24].

A'.1.3 Velocity

Η ταχύτητα ή Velocity αναφέρεται στον ρυθμό δημιουργίας και ανάλυσης των δεδομένων. Ένα παράδειγμα που θα μπορούσε να επιδείξει την έννοια της ταχύτητας στο πλαίσιο των μεγάλων δεδομένων, είναι η τεράστια εισροή δεδομένων από τις σύγχρονες ηλεκτρονικές συσκευές, σε πραγματικό ή σχεδόν πραγματικό χρόνο. Τα δεδομένα αυτά μπορεί να είναι γεωχωρικά, δημογραφικά, και διάφορα μοτίβα συμπεριφοράς, τα οποία προκειμένου να προσφέρουν αξία πρέπει να μπορούν να αναλυθούν σε πραγματικό χρόνο [24].

A'.1.4 Veracity

Η φιλαλήθεια ή Veracity αναπαριστά την αναξιπιστία που μπορεί να έχουν κάποιες πηγές δεδομένων. Για παράδειγμα, τα συναισθήματα που εκφράζουν κάποιοι καταναλωτές στα μέσα μαζικής δικτύωσης είναι εκ φύσεως ασταθή, καθώς εμπεριέχουν το στοιχείο της ανθρώπινης κρίσης. Παρόλα αυτά, κρίνεται ότι αυτά τα δεδομένα μπορούν δυνητικά να αποδώσουν χρήσιμες πληροφορίες. Γενικά, η φιλαλήθεια αφορά την ανάγκη αντιμετώπισης ανακριβών και αβέβαιων δεδομένων [24].

A΄.1.5 Variability

Η μεταβλητότητα ή Variability αναφέρεται στην μεταβολή του ρυθμού εισροής των δεδομένων. Συχνά, η ταχύτητα των μεγάλων δεδομένων δεν είναι σταθερή, καθώς μπορεί να εμφανίσει περιοδικές αυξομειώσεις [24].

A΄.1.6 Value

Αξία ή Value των μεγάλων δεδομένων ορίζεται ως η δυνητική αξία των δεδομένων σε σχέση με τον όγκο τους. Συνήθως, τα ληφθέντα δεδομένα έχουν χαμηλή αξία σε σχέση με τον όγκο τους. Παρόλα αυτά, μέσα από την ανάλυση μεγάλου όγκου τέτοιων δεδομένων μπορεί να εξαχθεί πληροφορία ή οποία θεωρείται υψηλής αξίας [24].

A΄.2 Η Ανάπτυξη των Μεγάλων Δεδομένων

Στο τέλος του 1970 προέκυψε η έννοια της βάσης δεδομένων, η οποία είναι μία τεχνολογία που χρησιμοποιείται για την αποθήκευση και ανάλυση δεδομένων. Με την αύξηση του όγκου των δεδομένων, η αποθήκευση και η επεξεργασία από ένα απλό υπολογιστικό σύστημα έγινε αδύνατη. Κατά το 1980 προτάθηκε ένα παράλληλο σύστημα δεδομένων που θα μπορούσε να ανταποκριθεί στη ζήτηση αυξανόμενου όγκου δεδομένων [18].

Η αρχιτεκτονική αυτού του συστήματος βασιζόταν στη χρήση των cluster (συμπλεγμάτων) όπου κάθε μηχανή είχε τον δικό της επεξεργαστή, αποθηκευτικό χώρο και δίσκο. Το σύστημα Teradata ήταν το πρώτο παράλληλο database σύστημα με εμπορική επιτυχία. Παρόμοια συστήματα έγιναν πολύ δημοφιλή λίγο αργότερα. Στις 2 Ιουνίου του 1986 συνέβη ένα πραγματικά σημαντικό γεγονός όταν η Teradata πέτυχε το πρώτο σύστημα παραλλήλων δεδομένων με δυνατότητα αποθήκευσης ενός terabyte, ώστε να βοηθήσει μία εμπορική επιχείρηση μεγάλου μεγέθους στην βόρεια Αμερική να επεκτείνει τη δυνατότητα αποθήκευσης δεδομένων της. Στο τέλος του 1990 τα πλεονεκτήματα του συστήματος παραλλήλων δεδομένων είχαν αναγνωριστεί στο πεδίο των βάσεων δεδομένων [14].

Τα τελευταία 20 χρόνια η ποσότητα των δεδομένων έχει αυξηθεί ραγδαία σε διάφορους τομείς. Σύμφωνα με μια έρευνα από το Διεθνές Ινστιτούτο δεδομένων (ICD), συνολικά το 2011 δημιουργήθηκαν και αντιγράφηκαν δεδομένα της τάξης του 1,82 ZB. Αυτό το μέγεθος προβλέπεται να πολλαπλασιάζεται κάθε δύο χρόνια για το προσεχές μέλλον [25].

Λόγω των πολυποίκιλων, μεγάλων σε όγκο και ετερογένεια πληροφοριών που εμπλέκονται σε ένα διάσπαρτο περιβάλλον, μία από τις πιο σημαντικές προκλήσεις των big data είναι ο υπολογισμός σε μία κλίμακα από megabytes μέχρι exabytes με σύνθετα υπολογιστικά μέσα. Η χρήση μίας δομής παράλληλου υπολογισμού, η υποστήριξη από μία κατάλληλη γλώσσα υπολογισμού, τα μοντέλα του software που μπορούν με επάρκεια να αναλύσουν τα διάφορα και διάσπαρτα δεδομένα, είναι κρίσιμα έτσι ώστε τα big data και οι διαδικασίες επεξεργασίας τους να αποδίδουν αξία μέσα από τον όγκο των πληροφοριών [67].

Ο παραδοσιακός τρόπος διαχείρισης δεδομένων και ανάλυση συστημάτων στηρίζεται σε σχεσιακά συστήματα διαχείρισης δεδομένων (RDBMS). Παρόλα αυτά, τέτοια συστήματα μπορούν να εφαρμοστούν σε δομημένα δεδομένα, και όχι σε ημι-δομημένα ή καθόλου δομημένα δεδομένα. Γίνεται φανερό πώς τέτοια παραδοσιακά συστήματα δεν μπορούν να διαχειριστούν το μεγάλο όγκο και την ετερογένεια των big data. Συνήθως για λύσεις μόνιμης αποθήκευσης και διαχείρισης δεδομένων μεγάλης κλίμακας γίνεται χρήση κατανεμημένων συστημάτων αρχείων [33] και NoSQL [13] βάσεων δεδομένων.

Η αλλαγή στα συστήματα διαχείρισης και αποθήκευσης δεδομένων επέφερε μεταβολή και στα μοντέλα συνέπειας αυτών των συστημάτων. Η κλιμάκωση κλασσικών σχεσιακών συστημάτων αναδεικνύεται προβληματική, καθώς παρουσιάζονται επιπλοκές μεταξύ των διαφόρων πτυχών της υψηλής διαθεσιμότητας στα κατανεμημένα συστήματα, οι οποίες δεν είναι πλήρως επιλύσιμες [50].

Τα κλασσικά σχεσιακά συστήματα υποστηρίζουν τις ιδιότητες ACID, οι οποίες είναι ένα σετ ιδιοτήτων των σχεσιακών βάσεων δεδομένων το οποίο έχει την πρόθεση να εγγυηθεί την ακεραιότητα των συναλλαγών τους, ακόμα και στην περίπτωση σφαλμάτων. Τα κριτήρια ACID είναι η ατομικότητα (Atomicity), συνοχή (Consistency), απομόνωση (Isolation) και η αντοχή (Durability) [39][16] [61][1][27].

Δυστυχώς οι ιδιότητες ACID δεν προσφέρουν καμία εγγύηση σχετικά με την διαθεσιμότητα, καθώς είναι προτιμότερο για μια υπηρεσία ACID να μην είναι διαθέσιμη, παρά να λειτουργεί με κάποιο τρόπο που χαλαρώνει τα κριτήρια αυτά. Έτσι, οι περισσότερες λύσεις μόνιμης αποθήκευσης και διαχείρισης μεγάλων δεδομένων τείνουν να ακολουθούν το μοντέλο συνέπειας BASE: Basically Available, Soft State, Eventual Consistency ή Βασικά Διαθέσιμη, Ελαφριά Κατάσταση, Τελικώς Συνεκτική [22].

Α΄.3 Η Σημασία των Μεγάλων Δεδομένων

Τα μεγάλα δεδομένα μπορούν να έχουν σημαντικό αντίκτυπο όχι μόνο στον ιδιωτικό τομέα, αλλά και στις εθνικές οικονομίες και τους πολίτες αυτών. Έρευνες έχουν δείξει ότι τα δεδομένα μπορούν να παρέχουν σημαντική αξία στην παγκόσμια οικονομία, βελτιώνοντας την παραγωγικότητα και την ανταγωνιστικότητα των εταιριών και του δημοσίου τομέα, δημιουργώντας σημαντικό πλεόνασμα για τους καταναλωτές. Για παράδειγμα, αν το σύστημα υγείας στις Η.Π.Α. χρησιμοποιούσε τεχνολογίες μεγάλων δεδομένων με αποτελεσματικό τρόπο για την αύξηση της αποδοτικότητας και της ποιότητας, εκτιμάται ότι η δυνητική αξία που θα επέφεραν αυτά τα δεδομένα κάθε χρόνο θα ήταν 300 δισεκατομμύρια δολάρια, με τα δύο τρίτα από αυτά να είναι στην μορφή εξοικονόμησης των εξόδων κατά περίπου 8 τις εκατό. Στον ιδιωτικό τομέα, εκτιμάται ότι αν ένας έμπορος λιανικής κάνει πλήρη χρήση τεχνολογιών μεγάλων δεδομένων, έχει την δυνατότητα να αυξήσει το λειτουργικό του περιθώριο κατά 60 τις εκατό. Στις ανεπτυγμένες οικονομίες της Ευρώπης, εκτιμάται ότι οι κυβερνήσεις μπορούν να εξοικονομήσουν 100 δισεκατομμύρια ευρώ από βελτιώσεις επιχειρησιακής αποδοτικότητας, χρησιμοποιώντας τεχνολογίες μεγάλων δεδομένων. Αυτή η εκτίμηση δεν συμπεριλαμβάνει μεθόδους ανάλυσης μεγάλων δεδομένων που θα μπορούσαν να μειώσουν απάτες, σφάλματα και φορολογικά κενά [47].

Κεφάλαιο Β΄

Μοντέλα Επεξεργασίας

ΤΟ κύριο μέλημα της μαζικής συλλογής δεδομένων είναι η εξαγωγή χρήσιμων πληροφοριών [47]. Είναι εμφανές ότι η κλίμακα της υποδομής που αποθηκεύει, διαχειρίζεται και επεξεργάζεται τα μεγάλα δεδομένα είναι πολύ μεγάλη. Είναι λοιπόν απαραίτητο να υπάρχουν μοντέλα επεξεργασίας τα οποία λαμβάνουν υπόψη αυτή την κλίμακα, και προσφέρουν ένα τρόπο εκμετάλλευσης αυτής, χωρίς να απαιτούν την συνεισφορά του εκάστοτε προγραμματιστή σε κάθε επίπεδο αποθήκευσης, διαχείρισης και επεξεργασίας.

Ο χώρος της επεξεργασίας μεγάλων δεδομένων εξελίσσεται καθημερινά, όμως υπάρχουν κάποια μοντέλα τα οποία καθόρισαν την φιλοσοφία ανάπτυξης των μοντέλων επεξεργασίας στο σύνολό τους.

Β΄.1 MapReduce

Το MapReduce είναι ένα προγραμματιστικό μοντέλο και μια υλοποίηση που βασίζεται σε αυτό, με σκοπό την επεξεργασία και την δημιουργία μεγάλων σετ δεδομένων. Οι χρήστες ορίζουν μια συνάρτηση map η οποία επεξεργάζεται ένα ζεύγος κλειδιού/τιμής για να δημιουργήσει ένα σετ ενδιάμεσων ζευγών κλειδιού/τιμής, και μια συνάρτηση reduce η οποία συνενώνει όλες τις ενδιάμεσες τιμές που είναι συσχετισμένες με το ίδιο ενδιάμεσο κλειδί. Πολλές εργασίες στην πραγματικότητα μπορούν να εκφραστούν με αυτό το μοντέλο.

Τα προγράμματα που είναι γραμμένα σε αυτό το λειτουργικό στυλ είναι αυτόματα παραλληλοποιήσιμα και μπορούν να εκτελεστούν σε ένα μεγάλο σύμπλεγμα από τυπικά συστήματα του εμπορίου. Το σύστημα εκτέλεσης διαχειρίζεται τον κατακερματισμό των εισαχθέντων δεδομένων,

την χρονοδρομολόγηση της εκτέλεσης του προγράμματος ανάμεσα σε ένα σετ συστημάτων, τις αποτυχίες υλικού και την επικοινωνία μεταξύ διεργασιών. Αυτό επιτρέπει σε προγραμματιστές χωρίς ιδιαίτερη εμπειρία σε παράλληλα και καταναμημένα συστήματα, να μπορούν εύκολα να χρησιμοποιήσουν τους πόρους ενός μεγάλου καταναμημένου συστήματος [17][44].

Β'.1.1 Παράδειγμα Υπολογισμού

Έστω ότι θέλουμε να μετρήσουμε της φορές όπου κάθε λέξη εμφανίζεται σε μία μεγάλη συλλογή εγγράφων. Ο ψευδο-κώδικας που ανταποκρίνεται στο μοντέλο MapReduce για την επίλυση του παραπάνω προβλήματος είναι αυτός:

```
1  map(String key, String value):
2      // key: document name
3      // value: document contents
4      for each word w in value:
5          EmitIntermediate(w, "1");
6
7  reduce(String key, Iterator values):
8      // key: a word
9      // values: a list of counts
10     int result = 0;
11     for each v in values:
12         result += ParseInt(v);
13     Emit(AsString(result));
```

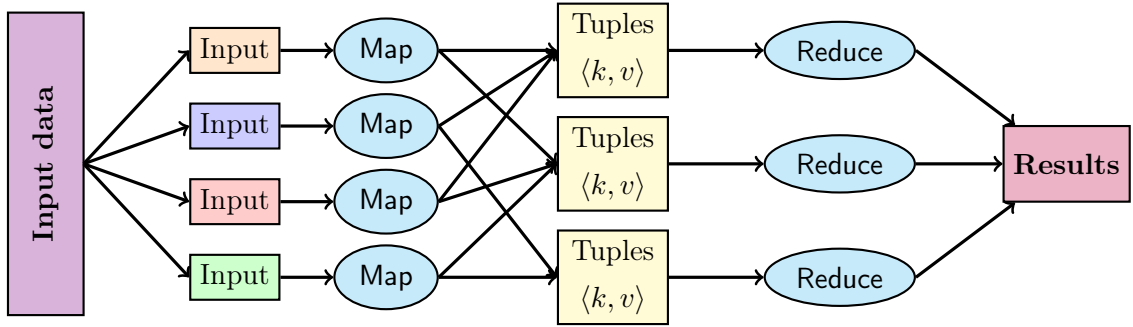
Η συνάρτηση map εκπέμπει την κάθε λέξη μαζί με έναν συσχετισμένο αριθμό εμφανίσεων. Η συνάρτηση reduce αθροίζει όλους τους αριθμούς εμφάνισης για την εκάστοτε λέξη [17].

Β'.1.2 Μοντέλο Εκτέλεσης

Μια εφαρμογή MapReduce πραγματώνει στην ουσία της μια σειρά βασικών χειρισμών σε μια αλληλουχία από μικρά έως μεγάλα σετ δεδομένων. Αυτά τα δεδομένα είναι με τέτοιο τρόπο οργανωμένα έτσι ώστε το μοντέλο εκτέλεσης του MapReduce να μπορεί να διανέμει διεργασίες, με σκοπό την παράλληλη εκτέλεσή τους. Τα δεδομένα ευρετηριάζονται χρησιμοποιώντας ένα ορισμένο κλειδί σε ζευγάρια <key,value>, στα οποία το κλειδί αντιπροσωπεύει κάποιο κριτήριο ομαδοποίησης συσχετισμένο με κάποια υπολογισμένη τιμή.

Οι κλήσεις της συνάρτησης Map διανέμονται σε πολλαπλά συστήματα δια της αυτόματης

κατακερμάτισης των δεδομένων εισόδου σε ένα σετ M διαιρέσεων. Οι διαιρέσεις εισόδου μπορούν να δεχτούν επεξεργασία παράλληλα από διαφορετικά συστήματα. Οι κλήσεις της συνάρτησης Reduce διανέμονται με την κατακερμάτιση του ενδιαμέσου χώρου κλειδιών σε R κομμάτια, χρησιμοποιώντας μια συνάρτηση κατακερματισμού όπως π.χ. $hash(key) \bmod R$. Ο αριθμός των διατμήσεων R και η συνάρτηση κατακερματισμού ορίζονται από τον χρήστη.



Σχήμα Β'.1: Απεικόνιση του μοντέλου εκτέλεσης MapReduce.

Β'.2 Vertex-Centric BSP

Από το προηγούμενο μοντέλο επεξεργασίας είναι εμφανές πως από μεγάλο όγκο δεδομένων μπορούν να εξαχθούν χρήσιμες πληροφορίες. Όσο περισσότερα δεδομένα υπάρχουν, τόσο πιο ακριβείς απαντήσεις μπορούν να εξαχθούν από ερωτήματα όπως το προαναφερθέν. Παρόλα αυτά, για συγκεκριμένου είδους δεδομένα, είναι επίσης δυνατή η εκμετάλλευση των σχέσεων μεταξύ των οντοτήτων εντός των δεδομένων. Σε αυτές τις περιπτώσεις, πιο σημαντικός από τα κομμάτια δεδομένων (και τα αθροίσματά τους) είναι ο τρόπος όπου αυτά σχετίζονται μεταξύ τους, όπως εξίσου σημαντική είναι και η δομή του δικτύου που δημιουργείται από αυτές τις συσχετίσεις [48].

Η δομή αυτού του δικτύου αναπαρίσταται με ένα γράφο. Ο γράφος είναι μια δομή που αναπαριστά αφηρημένες οντότητες (κορυφές ή κόμβους) και τις σχέσεις τους (ακμές). Οι γράφοι χρησιμοποιούνται στην επιστήμη των υπολογιστών από διάφορους τομείς για την αναπαράσταση δεδομένων και την έκφραση προβλημάτων. Δυστυχώς, ενώ πολύτιμες πληροφορίες μπορούν να ανακαλυφθούν από την ανάλυση διασυνδεδεμένων δεδομένων, η ανάλυση γράφων είναι συνήθως μια υπολογιστικά ακριβή διαδικασία που συνοδεύεται από διάφορες τεχνικές προκλήσεις [48].

Ένα μοντέλο υπολογισμού το οποίο χρησιμοποιείται για να ξεπεραστούν κάποιες από αυτές τις προκλήσεις είναι το Bulk Synchronous Parallel, το οποίο ορίζεται ως ο συνδυασμός τριών χαρακτηριστικών:

1. Ένας αριθμός από λειτουργικά μέρη, όπου το κάθε ένα εκτελεί λειτουργίες επεξεργασίας και/ή μνήμης.
2. Ένα δρομολογητή, ο οποίος παραδίδει μηνύματα από σημείο σε σημείο μεταξύ ζευγαριών λειτουργικών μερών.
3. Υποδομή για τον συγχρονισμό όλων ή ενός υποσυνόλου των λειτουργικών μερών σε συχνά διαστήματα από L μονάδες χρόνου, όπου L είναι η παράμετρος της περιοδικότητας. Ένας υπολογισμός αποτελείται από μια αλληλουχία supersteps (υπερβημάτων). Σε κάθε υπερβήμα, κάθε λειτουργικό μέρος αναλαμβάνει μια εργασία που αποτελείται από κάποιο συνδυασμό επεξεργασίας σε τοπικό επίπεδο, μετάδοσης μηνυμάτων και λήψη μηνυμάτων από άλλα λειτουργικά μέρη. Μετά από κάθε περίοδο L μονάδων χρόνου, ένας συνολικός έλεγχος πραγματοποιείται για να διαπιστωθεί εάν το υπερβήμα έχει ολοκληρωθεί από όλα τα λειτουργικά μέρη. Αν έχει ολοκληρωθεί, τότε το σύστημα προχωρά στο επόμενο υπερβήμα. Σε κάθε άλλη περίπτωση, η επόμενη περίοδος L μονάδων παρέχεται ως προς το μη ολοκληρωμένο υπερβήμα [66].

B'.2.1 Pregel

Το σύστημα Pregel είναι μία υλοποίηση της Google απέναντι στο πρόβλημα της κατανεμημένης επεξεργασίας μεγάλης κλίμακας γράφων. Σε υψηλό επίπεδο, η οργάνωση των προγραμμάτων του Pregel είναι εμπνευσμένη από το μοντέλο Bulk Synchronous Parallel. Όπως και στο BSP, έτσι και στο Pregel οι υπολογισμοί αποτελούνται από μια αλληλουχία υπερβημάτων. Κατά τη διάρκεια ενός υπερβήματος, μία συνάρτηση ορισμένη από τον χρήστη καλείται για τον κάθε κόμβο, εννοιολογικά παράλληλα. Η συνάρτηση ορίζει μια συμπεριφορά σε ένα μοναδικό κόμβο V και ένα μοναδικό υπερβήμα S . Μπορεί να διαβάσει μηνύματα που έχουν σταλεί στο V στο υπερβήμα $S - 1$, να στείλει μηνύματα σε άλλους κόμβους που θα ληφθούν στο υπερβήμα $S + 1$, και να τροποποιήσει την κατάσταση του V και των εξερχομένων από αυτόν ακμών. Τα μηνύματα τυπικά αποστέλλονται προς τους εξερχόμενους κόμβους, αλλά κάποιο μήνυμα μπορεί να σταλεί σε οποιονδήποτε κόμβο, αρκεί το αναγνωριστικό του να είναι γνωστό.

Τα δεδομένα εισόδου σε ένα υπολογισμό Pregel είναι ένας κατευθυνόμενος γράφος στον οποίο

κάθε κόμβος ταυτοποιείται από ένα αλφαριθμητικό αναγνωριστικό κόμβου. Κάθε κόμβος είναι συσχετισμένος με μία τροποποιήσιμη τιμή, η οποία παρέχεται από τον χρήστη. Οι κατευθυνόμενες ακμές είναι συσχετισμένες με τους πηγαίους τους κόμβους, και κάθε ακμή αποτελείται από μία τροποποιήσιμη τιμή η οποία παρέχεται από τον χρήστη, καθώς και το αναγνωριστικό του κόμβου προς τον οποίο κατευθύνεται.

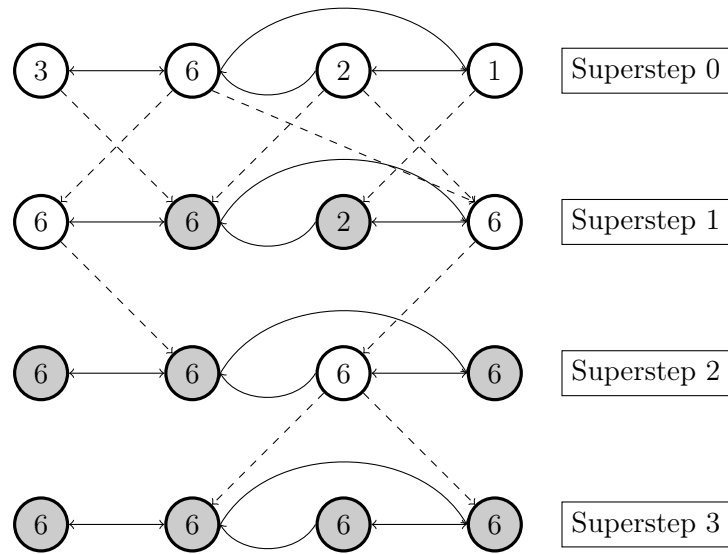
Ένας τυπικός Pregel υπολογισμός αποτελείται από δεδομένα εισόδου, όταν αρχικοποιείται ο γράφος, ακολουθούμενα από μία ακολουθία υπερβημάτων διαχωρισμένα από γενικά σημεία συγχρονισμού, μέχρι ο αλγόριθμος να τερματίσει, όπου και αποδίδεται το αποτέλεσμα.

Εντός ενός υπερβήματος, οι κόμβοι υπολογίζουν παραλλήλως, με τον καθένα να εκτελεί την ίδια, ορισμένη από τον χρήστη, συνάρτηση όπου εκφράζει την λογική ενός δοσμένου αλγορίθμου. Ένας κόμβος μπορεί να τροποποιήσει την κατάσταση του, ή την κατάσταση των εξερχομένων ακμών του, να λάβει μηνύματα απεσταλμένα σε αυτόν από το προηγούμενο υπερβήμα, να στείλει μηνύματα σε άλλου κόμβους (τα οποία θα ληφθούν από αυτούς στο επόμενο υπερβήμα), ή ακόμη και να μεταλλάξει την τοπολογία του γράφου. Σε αυτό το μοντέλο οι ακμές είναι δευτερεύουσες, καθώς δεν υπάρχει κάποιος συσχετισμένος υπολογισμός με αυτές.

Ο τερματισμός ενός αλγορίθμου βασίζεται στην έκδοση ψήφου τερματισμού από τον κάθε κόμβο. Στο υπερβήμα 0, κάθε κόμβος βρίσκεται σε ενεργή κατάσταση. Όλοι οι ενεργοί κόμβοι συμμετέχουν στον υπολογισμό οποιουδήποτε δοσμένου υπερβήματος. Ένας κόμβος αδρανοποιεί τον εαυτό του, εκδίδοντας μια ψήφο τερματισμού. Αυτό σημαίνει ότι ο κόμβος δεν έχει περαιτέρω εργασία να εκτελέσει, εκτός εάν ενεργοποιηθεί εξωτερικά, με συνέπεια αυτός ο κόμβος να μην εκτελεστεί στα επόμενα υπερβήματα, εκτός εάν λάβει κάποιο μήνυμα. Αν ενεργοποιηθεί ξανά από κάποιο μήνυμα, ο κόμβος πρέπει ρητά να αδρανοποιήσει τον εαυτό του ξανά. Ο αλγόριθμος στο σύνολό του τερματίζεται όταν όλοι οι κόμβοι είναι ταυτόχρονα αδρανείς και δεν υπάρχουν μηνύματα υπό μεταφορά.

Το αποτέλεσμα ενός προγράμματος Pregel είναι ένα σύνολο από τιμές που έχουν παραχθεί ρητά από τους κόμβους. Είναι συχνά ένας κατευθυνόμενος γράφος, ισομορφικός ως προς τα δεδομένα εισόδου, όμως αυτό δεν αποτελεί απαραίτητη ιδιότητα του συστήματος, καθώς κόμβοι και ακμές μπορούν να προστεθούν ή να αφαιρεθούν κατά τη διάρκεια του υπολογισμού. Για παράδειγμα, ένας αλγόριθμος συσσωμάτωσης μπορεί να δημιουργήσει ένα μικρό σύνολο από αποσυνδεδεμένους κόμβους, επιλεγμένους από ένα μεγάλο γράφο. Ένας αλγόριθμος εξόρυξης δεδομένων από γράφους μπορεί απλά να παράξει συνολικά στατιστικά που έχουν εξορυχθεί από τον γράφο.

Ένα παράδειγμα το οποίο επιδεικνύει αυτές τις έννοιες θα μπορούσε να είναι το εξής: έστω ότι υπάρχει ένας ισχυρά συνεκτικός γράφος όπου κάθε κόμβος περιέχει μία τιμή, και την διαδίδει την μεγαλύτερη τιμή σε κάθε κόμβο. Σε κάθε υπερβήμα, κάθε κόμβος που έχει εντοπίσει μία τιμή μεγαλύτερη από αυτή των μηνυμάτων του, την διαδίδει σε όλους του γειτονικούς κόμβους. Όταν κανείς επιπλέον κόμβος δεν αλλάζει σε ένα υπερβήμα, ο αλγόριθμος τερματίζεται [46].



Σχήμα Β'.2: Παράδειγμα μέγιστης τιμής. Οι διακεκομμένες γραμμές αντιπροσωπεύουν τα μηνύματα. Οι σκιασμένοι κόμβοι έχουν εκδώσει ψήφο αδρανοποίησης.

Κεφάλαιο Γ΄

Κώδικας ως Μεγάλα Δεδομένα

Ως πηγαίος κώδικας ορίζεται η οποιαδήποτε πλήρως εκτελέσιμη περιγραφή ενός συστήματος λογισμικού. Συνεπώς, θεωρείται ότι συμπεριλαμβάνεται από αυτόν τον ορισμό η γλώσσα μηχανής, γλώσσες υψηλότερου επιπέδου και εκτελέσιμες γραφικές αναπαραστάσεις συστημάτων [28].

Ως ανοιχτός κώδικας ορίζεται η διανομή ενός λογισμικού που πληρεί τα παρακάτω κριτήρια:

- Η διανομή του είναι ελεύθερη. Η άδεια χρήσης δεν απαγορεύει την πώληση ή την παραχώρηση του λογισμικού από οποιαδήποτε οντότητα.
- Η διανομή του λογισμικού πρέπει να συνοδεύεται από τον πηγαίο κώδικα του και να επιτρέπει την τροποποίηση και αναδιανομή του.
- Τρίτες διανομές που προκύπτουν από μία διανομή λογισμικού πρέπει να διανέμονται και αυτές από την ίδια άδεια χρήσεως.
- Εγγυάται την αυθεντικότητα του πηγαίου κώδικα που έχει παραχθεί από τον δημιουργό της διανομής λογισμικού.
- Δεν κάνει διακρίσεις μεταξύ ατόμων ή ομάδων.
- Δεν κάνει διακρίσεις μεταξύ πεδίων ενδιαφέροντος.
- Πλαισιώνει την αυτόματη μεταφορά της άδειας χρήσης.
- Η άδεια χρήσης δεν είναι συγκεκριμένη για κάποιο προϊόν.

- Η άδεια χρήσης δεν θέτει περιορισμούς σε συνοδευτικό λογισμικό.
- Η άδεια χρήσης πρέπει να είναι τεχνολογικά ουδέτερη [53].

Γ'.1 Η Διάδοση του Ανοιχτού Κώδικα

Για το μεγαλύτερο κομμάτι του 20ου αιώνα, η υλοποίηση και η θεωρία πίσω από κάποια τεχνολογική εξέλιξη παρέμεναν κεκλεισμένων των θυρών της εταιρίας ή οργανισμού που ήταν υπεύθυνη για αυτήν. Για την ακρίβεια, η ανάγκη χρηματοδότησης και ελέγχου τέτοιων τεχνολογικών εξελίξεων θεωρούνταν ένας από τους κύριους λόγους ύπαρξης αυτών των εταιριών [11].

Αν και από τις αρχές της δεκαετίας του 2000 διαφαινόταν η εμπορική επιτυχία της ανάπτυξης και χρήσης ανοικτού λογισμικού [10][42][12], πλέον είναι αδιαμφισβήτητο ότι το ανοιχτό λογισμικό έχει καθιερωθεί ως βιώσιμο μέσο εδραίωσης εμπορικών λειτουργιών [29][21].

Σημαντικό ρόλο στην εξέλιξη του ανοικτού λογισμικού, στο δεύτερο μισό της δεκαετίας, έπαιξε η δημιουργία του εργαλείου git. Το git είναι ένα version control system το οποίο έχει κατανομημένη αρχιτεκτονική και χρησιμοποιείται αμιγώς από την κοινότητα του ανοικτού λογισμικού. Εξίσου σημαντικό ρόλο έπαιξε και η ύπαρξη του GitHub, μιας πλατφόρμας που φιλοξενεί αποθετήρια που δημιουργούνται από το git [43].

Γ'.2 Ο Σκοπός της Ανάλυσης Κώδικα

Στα πλαίσια αυτής της πτυχιακής εργασίας, ο ανοιχτός πηγαίος κώδικας είναι κομβικής σημασίας, καθώς αποτελεί δυνητικά μια πηγή μεγάλων δεδομένων. Το 2019 το GitHub είχε 40 εκατομμύρια προγραμματιστές και 44 εκατομμύρια αποθετήρια, τα οποία είχαν δημιουργηθεί μεταξύ του 2018 και του 2019 (ιδιωτικά και δημόσια) [26]. Είναι εμφανές λοιπόν ότι υπάρχει πληθώρα πηγών από τις οποίες μπορούν να αντληθούν δεδομένα.

Λόγω της πληθώρας και του μεγάλου μεγέθους κάποιων λογισμικών ανοικτού κώδικα, όπως ο Linux Kernel με περίπου 25,5 εκατομμύρια γραμμές κώδικα το 2018 [41] ή ο GCC με περίπου 14,5 εκατομμύρια γραμμές κώδικα το 2014 [40], έχει ενδιαφέρον να μελετηθεί η διαδικασία εξόρυξης πληροφοριών, εξετάζοντας τον πηγαίο κώδικα όχι κυρίως ως προς την σημασιολογία του, αλλά ως δεδομένα τα οποία μπορούν να αναπαρασταθούν ως γράφοι. Μέσω αυτών των αναπαραστάσεων μπορεί να μελετηθεί το αν προκύπτουν μοτίβα [4] ή ενδιαφέρουσες σχέσεις

μεταξύ των κόμβων, εφαρμόζοντας τεχνικές και μοντέλα υπολογισμού που αρμόζουν σε σύνολα μεγάλων δεδομένων.

Η αντίληψη του κώδικα ως δεδομένα δεν έχει νόημα μόνο υπό το πλαίσιο του ανοικτού κώδικα. Είναι συνετό να μελετηθούν τεχνικές ανάλυσης μεγάλων βάσεων κώδικα, καθώς αυτές θα μπορούσαν να εφαρμοστούν σε προβλήματα εύρεσης αδυναμιών ασφάλειας [64], στην στατιστική ανάλυση κώδικα [62], ή ακόμη και σε ευρετικά προβλήματα πρόβλεψης κώδικα [63].

Με την προβλεπόμενη επέκταση του τομέα των υπηρεσιών που βασίζονται σε τεχνολογίες ανοικτού κώδικα [65], είναι πιθανό στο μέλλον να θεωρηθεί επιτακτική ανάγκη η δυνατότητα μελέτης και εξαγωγής πληροφοριών από μεγάλες βάσεις κώδικα. Ο σκοπός της παρούσας πτυχιακής εργασίας είναι να αποδώσει μία τέτοια δυνατότητα σε επίπεδο proof-of-concept, και να δείξει πως μπορεί να χρησιμοποιηθεί σε πρακτική εφαρμογή.

Κεφάλαιο Δ΄

NoSQL και Βάσεις Δεδομένων Γράφων

Οι περιορισμοί των κλασικών βάσεων δεδομένων, πιο συγκεκριμένα του σχεσιακού μοντέλου, να καλύψουν τις απαιτήσεις των σύγχρονων εφαρμογών, έχει οδηγήσει στην ανάπτυξη νέων τεχνολογιών βάσεων δεδομένων υπό την ονομασία NoSQL [5][52]. Ένας τύπος NoSQL βάσης δεδομένων είναι και οι βάσεις δεδομένων γράφων, οι οποίες είναι προορισμένες να αποθηκεύουν δεδομένα που αναπαρίστανται ως γράφοι.

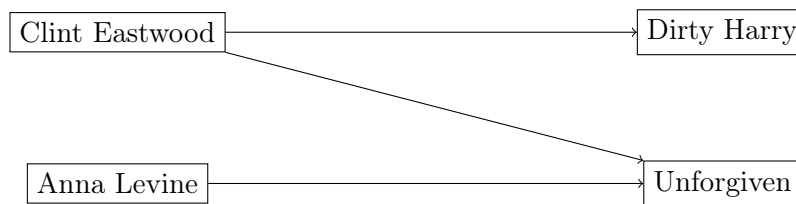
Κατά κύριο λόγο, έντονη ερευνητική δραστηριότητα γύρω από τις βάσεις δεδομένων γράφων υπήρξε στις αρχές της δεκαετίας του 1990, χωρίς να παραμείνει επιφανές ερευνητικό ζήτημα για πολύ καιρό. Πρόσφατα, αυτός ο τομέας έχει γίνει και πάλι δημοφιλής, καθώς για διάφορα σύνολα δεδομένων που συλλέγονται από σύγχρονες εφαρμογές, η αξία της πληροφορίας εντός αυτών των συνόλων βρίσκεται στις σχέσεις μεταξύ των οντοτήτων που παρουσιάζονται εντός των δεδομένων, τόσο ή περισσότερο από ότι στις οντότητες αυτές καθαυτές. Η συνεχόμενη εμφάνιση και αύξηση μεγάλων και σύνθετων συνόλων πληροφοριών που μπορούν να αναπαρασταθούν ως γράφοι κάνουν την ύπαρξη σύγχρονων βάσεων γράφων επιτακτική ανάγκη [5].

Δ΄.1 Μοντέλα Βάσεων Δεδομένων Γράφων

Ένα μοντέλο βάσης δεδομένων γράφων είναι ένα μοντέλο όπου στο οποίο οι δομές δεδομένων για το σχήμα είναι μοντελοποιημένες ως κατευθυνόμενοι γράφοι, ή γενικεύσεις αυτής της

δομής δεδομένων. Ο χειρισμός δεδομένων εκφράζεται με πράξεις προορισμένες ως προς τους γράφους και με κατασκευαστές τύπων [6].

Οι γράφοι μπορούν να χρησιμοποιηθούν για να μοντελοποιήσουν δεδομένα, όπου οι κόμβοι αναπαριστούν αντικείμενα ενός πεδίου ενδιαφέροντος, και οι ακμές αναπαριστούν τις σχέσεις μεταξύ αυτών των αντικειμένων. Για παράδειγμα, αν ένας γράφος χρησιμοποιείται για να μοντελοποιήσει δεδομένα σχετικά με κινηματογραφικές ταινίες, οι κόμβοι μπορούν να αναπαριστούν ηθοποιούς και ταινίες, και οι κατευθυνόμενες ακμές μπορούν να υποδεικνύουν ποιους ηθοποιούς μετείχαν σε ποιες ταινίες.



Σχήμα Δ'.3: Παράδειγμα ενός απλού γράφου.

Ενώ η παραπάνω αναπαράσταση είναι ενστικτώδης, είναι δύσκολο να εκφραστούν διαφορετικού είδους σχέσεις, καθώς είναι υπερβολικά απλή. Μία λύση σε αυτό το πρόβλημα δίνουν οι edge-labelled graphs.

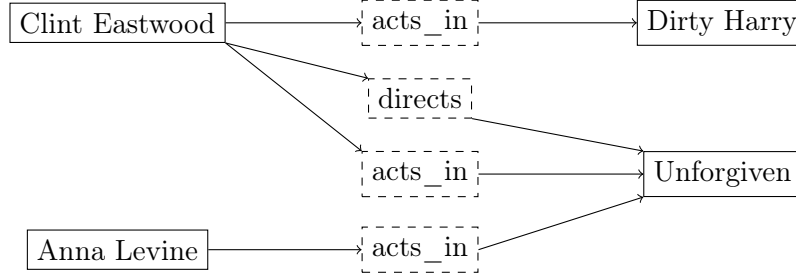
Δ'.1.1 Edge-labelled Graphs

Στους edge-labelled graphs ταμπέλες ανατίθενται σε ακμές, όπου υποδεικνύουν τους διαφορετικούς τύπους σχέσεων στο πεδίο που περιγράφουν. Ο τυπικός ορισμός ενός edge-labelled graph είναι ο παρακάτω:

- V είναι ένα περατό σύνολο από κόμβους.
- E είναι ένα περατό σύνολο από ακμές, τυπικά $E \subseteq V \times Lab \times V$, όπου Lab είναι ένα σύνολο από ταμπέλες [7].

Αν πάρουμε το παραπάνω παράδειγμα και το μετατρέψουμε σε edge-labelled graph, μπορούμε να προσθέσουμε την σχέση μιας οντότητας, όχι μόνο ως συμμετοχή, αλλά και να ορίσουμε το είδος της συμμετοχής αυτής. Στην προκειμένη περίπτωση ο Clint Eastwood πρωταγωνιστεί αλλά και σκηνοθετεί στην ταινία Unforgiven. Με αυτές τις ιδιότητες, το παραπάνω παράδειγμα

διαμορφώνεται ως εξής:



Σχήμα Δ'.4: Παράδειγμα ενός edge-labelled graph.

Δ'.1.2 Property Graphs

Στους edge-labelled graphs γίνεται χρήση ταμπελών για την υπόδειξη του τύπου μιας ακμής, με πολλαπλές ακμές να μπορούν να έχουν τον ίδιο τύπο. Με τον ίδιο τρόπο μπορούν να χρησιμοποιηθούν ταμπέλες για την τιτλοφόρηση κόμβων.

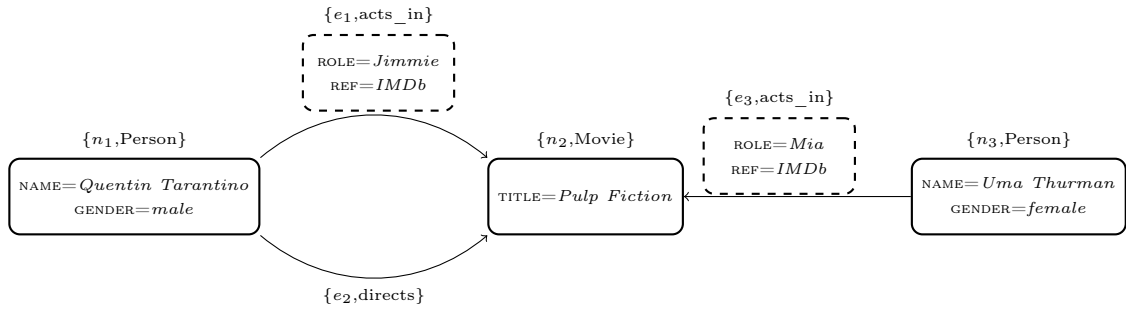
Στους property graphs και οι κόμβοι αλλά και οι ακμές μπορούν να έχουν ταμπέλες. Κάθε ακμή και κόμβος συσχετίζεται επιπλέον με ένα μοναδικό αναγνωριστικό, το οποίο μπορεί να χρησιμοποιηθεί ως τρόπος σύνδεσης περαιτέρω μεταδεδομένων, στη μορφή συνόλων ζευγαριών ιδιότητας-τιμής (τα οποία αποκαλούνται ως χαρακτηριστικά), με μία ακμή ή ένα κόμβο. Ενώ θα ήταν δυνατόν τα χαρακτηριστικά να μοντελοποιούνται ως επιπλέον ακμές, με αυτόν τον τρόπο δίνεται η δυνατότητα υποσημειώσεων εντός του γράφου, χωρίς να επηρεάζεται η συνολική του δομή.

Ο τυπικός ορισμός ενός property graph G είναι μία πλειάδα $(V, E, \rho, \lambda, \sigma)$ όπου:

- V είναι ένα περατό σύνολο από κόμβους.
- E είναι ένα περατό σύνολο από ακμές, τέτοιο ώστε V και E να μην έχουν κοινά στοιχεία.
- $\rho : E \rightarrow (V \times V)$ είναι ολική συνάρτηση. Ακόμη, $\rho(e) = (v_1, v_2)$ υποδεικνύει ότι e είναι κατευθυνόμενη ακμή από κάποιο κόμβο v_1 σε κάποιο κόμβο v_2 εντός του G .
- $\lambda : (V \cup E) \rightarrow Lab$ είναι ολική συνάρτηση με το Lab να είναι ένα σύνολο από ταμπέλες. Ακόμη, αν $v \in V$ (αντίστοιχα, $e \in E$) και $\rho(v) = l$ (αντίστοιχα, $\rho(e) = l$), τότε l είναι ταμπέλα κόμβου v (αντίστοιχα, ακμής e) στον G .

- $\sigma : (V \cup E) \times Prop \rightarrow Val$ είναι μερική συνάρτηση με $Prop$ ένα περατό σύνολο από ιδιότητες και Val ένα σύνολο τιμών. Ακόμη, αν $v \in V$ (αντίστοιχα, $e \in E$), $p \in Prop$ και $\sigma(v, p) = s$ (αντίστοιχα, $\sigma(e, p) = s$), και έπειτα s είναι η τιμή την ιδιότητας p για κόμβο v (αντίστοιχα, ακμή e) στον property graph G [7].

Ένα αντίστοιχο παράδειγμα που μπορεί να αναπαρασταθεί ως property graph είναι το εξής:



Σχήμα Δ΄.5: Παράδειγμα ενός property graph.

Δ΄.2 Cypher και Neo4j

Η Neo4j είναι μία βάση δεδομένων γράφων, όπου μαζί με την γλώσσα ερωτημάτων της (Cypher), χρησιμοποιεί το μοντέλο των property graphs. Όπως έχει αναφερθεί και πιο πριν, το μοντέλο αυτό αποτελείται από κόμβους οι οποίοι αναπαριστούν οντότητες (ανθρώπους, τραπεζικούς λογαριασμούς καταστήματα, κ.α.), και σχέσεις (γνωστές και ως ακμές) οι οποίες αναπαριστούν τις συνδέσεις μεταξύ των οντοτήτων. Στο μοντέλο γράφου, οι σχέσεις είναι όσο σημαντικές είναι και οι οντότητες. Ακόμη, οποιοσδήποτε αριθμός από ιδιότητες, στη μορφή ζευγαριών κλειδιού-τιμής, μπορεί να συσχετίζεται με κόμβους και σχέσεις. Αυτό επιτρέπει την μοντελοποίηση και την ερώτηση σύνθετων δεδομένων.

Τα ερωτήματα τα οποία γίνονται στην βάση δεδομένων Neo4j γίνονται με την χρήση της γλώσσας Cypher. Η Cypher είναι μία δηλωτική γλώσσα ερωτημάτων για property graphs, η οποία παρέχει την δυνατότητα ερωτημάτων, τροποποίησης δεδομένων, αλλά και την δυνατότητα καθορισμού ορισμών σχημάτων [23].

Δ'.2.1 Γραμμικά Ερωτήματα

Ένα ερώτημα Cypher παίρνει ως είσοδο έναν property graph και αποδίδει έναν πίνακα. Αυτοί οι πίνακες μπορούν να θεωρηθούν ως πάροχοι δεσμεύσεων για παραμέτρους που μετέχουν σε κάποιο μοτίβο εντός κάποιου γράφου, με κάποια επιπλέον επεξεργασία.

Η Cypher δομεί τα ερωτήματα γραμμικά. Αυτό επιτρέπει στους χρήστες να σκέφτονται ότι η επεξεργασία ερωτημάτων ξεκινάει από την αρχή του κειμένου του ερωτήματος και προχωράει γραμμικά προς το τέλος. Κάθε όρος εντός ενός ερωτήματος είναι μία συνάρτηση η οποία δέχεται ένα πίνακα και αποδίδει ένα πίνακα ο οποίος μπορεί να επεκτείνει τον αριθμό των πεδίων και να προσθέσει νέες πλειάδες. Το συνολικό ερώτημα είναι η σύνθεση αυτών των συναρτήσεων. Αυτή η γραμμική σειρά των όρων γίνεται κατανοητή εξ' ολοκλήρου δηλωτικά. Έτσι, αντί να γίνεται η δήλωση της προβολής των αποτελεσμάτων στην αρχή τους ερωτήματος, όπως στην SQL με την χρήση του όρου SELECT, στην Cypher η δήλωση της προβολής γίνεται στο τέλος του ερωτήματος με τον όρο RETURN.

Η γραμμική ροή των ερωτημάτων στην Cypher επεκτείνεται και στην σύνθεση ερωτημάτων. Με την χρήση του WITH, το ερώτημα συνεχίζει με τον προβληθέν πίνακα από το κομμάτι του ερωτήματος πριν το WITH, ως τον πίνακα οδηγό για το κομμάτι του ερωτήματος μετά το WITH. Ο όρος WITH επιτρέπει τις ίδιες προβολές δεδομένων με τον όρο RETURN, συμπεριλαμβανομένων και των αθροισμάτων. Επιπρόσθετα, η Cypher υποστηρίζει εμφωλευμένα υπό-ερωτήματα με την χρήση του όρου UNION [23].

Δ'.2.2 Εύρεση Μοτίβων

Η κεντρική ιδέα πίσω από τα ερωτήματα Cypher είναι η εύρεση μοτίβων. Τα μοτίβα στη Cypher εκφράζονται με οπτική μορφή ως ASCII art, όπως $(a) - [r] - > (b)$. Ο όρος MATCH χρησιμοποιεί ένα τέτοιου είδους μοτίβο και εισάγει νέες εγγραφές, με δεσμεύσεις των ταιριασμένων μοτίβων από τον ερωτηθέντα γράφο [23].

Δ'.2.3 Τροποποίηση Δεδομένων

Η Cypher υποστηρίζει όρους για την τροποποίηση ενός γράφου. Οι όροι τροποποίησης χρησιμοποιούν την ίδια οπτική μορφή με την εύρεση μοτίβων, αποδίδοντας ένα συνεκτικό σημασιολογικό μοντέλο. Οι βασικοί όροι τροποποίησης περιλαμβάνουν τον όρο CREATE για την δημιουργία νέων κόμβων και σχέσεων, τον όρο DELETE για την αφαίρεση οντοτήτων, και τον

όρο SET για την τροποποίηση ιδιοτήτων. Επιπλέον, η Cypher υποστηρίζει τον όρο MERGE, ο οποίος προσπαθεί να βρει το δοθέν μοτίβο, αλλά εάν αυτό δεν υπάρχει το δημιουργεί [23].

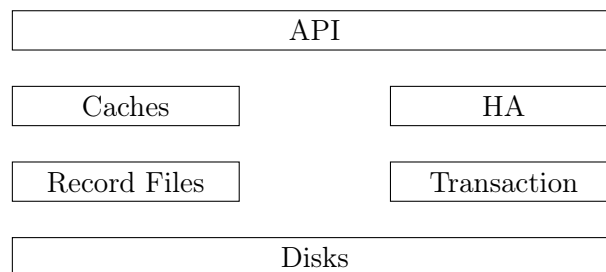
Δ΄.2.4 Αρχιτεκτονική Συστήματος

Τα δεδομένα γράφων στη Neo4j αποθηκεύονται στον σκληρό δίσκο του συστήματος στη μορφή αρχείων καταγραφής. Προκειμένου να βελτιωθεί η ταχύτητα ανάκτησης, η Neo4j χρησιμοποιεί δύο επίπεδα μνήμης cache: την cache του συστήματος αρχείων και μια cache αντικειμένων. Στον δίσκο, οι σχέσεις περιέχουν το μεγαλύτερο μέρος των πληροφοριών, ενώ κάθε κόμβος συγκρατεί μόνο την πρώτη αναφορά του στις σχέσεις. Αντιθέτως, στην cache αντικειμένων ο κόμβος συγκρατεί όλες τις σχέσεις και οι σχέσεις συγκρατούν όλες τις ιδιότητές τους.

Ένα ακόμη μέρος τους συστήματος είναι τα κομμάτια που υλοποιούν τις συναλλαγές, τα οποία συμπεριλαμβάνουν το αρχείο καταγραφής συναλλαγών και την διαχείριση συναλλαγών και εγγυώνται ότι οι ιδιότητες ACID των συναλλαγών ισχύουν.

Για την υλοποίηση δυνατοτήτων clustering, η Neo4j έχει κομμάτια υψηλής διαθεσιμότητας ή High Availability.

Τελικώς, όλη η αρχιτεκτονική πλαισιώνεται από το API της βάσης [34] [51].



Σχήμα Δ΄.6: Μια βασική επισκόπηση της αρχιτεκτονικής συστήματος της Neo4j.

Κεφάλαιο Ε΄

Το Σύστημα Ανάλυσης Κώδικα Scrubj

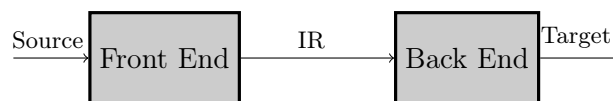
ΟΙ γλώσσες προγραμματισμού είναι ο ακρογωνιαίος λίθος της ανάπτυξης λογισμικού. Θεωρητικά, κάθε αποτέλεσμα που προκύπτει από την εκτέλεση ενός προγράμματος, μπορεί επαγωγικά να εξηγηθεί από τον κώδικα του [32]. Προκειμένου ένας επεξεργαστής να μπορεί να ερμηνεύσει τις οδηγίες ενός προγράμματος, οι δομές ελέγχου, το συντακτικό και όλες οι υπόλοιπες πτυχές της εκάστοτε γλώσσας [31] πρέπει να μετατραπούν σε γλώσσα μηχανής (γνωστή και ως δυαδικός κώδικας) [15]. Κάτι τέτοιο επιτυγχάνεται με τη χρήση ενός προγράμματος που ονομάζεται *compiler*.

Προκειμένου να γίνει κατανοητό το σύστημα ανάλυσης κώδικα το οποίο προτείνεται στο πλαίσιο αυτής της πτυχιακής εργασίας, είναι θεμιτό να γίνει μια συνοπτική περιγραφή της λειτουργίας κάποιων πτυχών ενός *compiler*, καθώς το προαναφερθέν σύστημα είναι ιδιαίτερα ενσωματωμένο στο *front end* του GCC [56], ο οποίος είναι ένας πασίγνωστος *compiler* για την προγραμματιστική γλώσσα C.

Έπειτα, αφού αναλυθούν τα επιμέρους σχετικά διαδικαστικά κομμάτια της μεταγλώττισης και της δομής των *compiler*, παρουσιάζεται ο κώδικας και η δομή του προτεινόμενου συστήματος ‘Scrubj’. Επίσης, γίνεται αναλυτική επεξήγηση των σχεδιαστικών επιλογών και του πηγαίου κώδικα του συστήματος.

Ε'.1 Η Δομή Ενός Compiler

Ένας compiler πρέπει να μπορεί να κατανοήσει ένα πηγαίο πρόγραμμα ως είσοδο και να αντιστοιχίσει την λειτουργικότητά του στο επιθυμητό σύστημα. Η ιδιαίτερη φύση αυτών των δύο διαδικασιών οδηγεί στην διαίρεση των εργασιών που αναλαμβάνει ένας compiler, και καταλήγει σε ένα σχεδιασμό που διαχωρίζει την διαδικασία της μεταγλώττισης σε δύο μεγάλα κομμάτια: το front end και το back end [15].



Σχήμα Ε'.7: Μια απλοποιημένη ροή λειτουργίας ενός compiler.

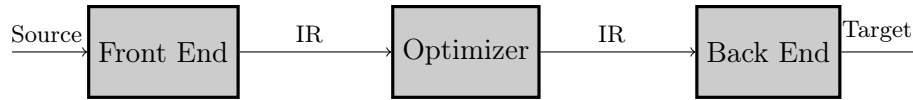
Το front end εστιάζει στην κατανόηση της πηγαίας γλώσσας του προγράμματος. Το back end εστιάζει στην αντιστοίχιση της λειτουργικότητας του προγράμματος στο επιθυμητό σύστημα. Αυτός ο διαχωρισμός έχει διάφορες σημαντικές επιπτώσεις στην σχεδίαση και την υλοποίηση των compiler.

Το front end πρέπει να κωδικοποιήσει το αποτέλεσμα της επεξεργασίας του από το πηγαίο πρόγραμμα σε κάποια δομή, η οποία μπορεί αργότερα να χρησιμοποιηθεί από το back end. Αυτή η ενδιάμεση αναπαράσταση (IR) γίνεται η καθολική αναπαράσταση του κώδικα υπό μεταγλώττιση. Σε κάθε φάση της μεταγλώττισης ο compiler έχει μία τέτοια αναπαράσταση. Μπορεί κατά τη διάρκεια της μεταγλώττισης να γίνεται χρήση πολλών διαφορετικών (IR), αλλά σε κάθε φάση μία αναπαράσταση είναι η καθολική ενδιάμεση αναπαράσταση [15].

Ο compiler μπορεί να κάνει πολλαπλά περάσματα επί της ενδιάμεσης μορφής του κώδικα πριν την παραγωγή του τελικού προγράμματος. Ο απώτερος σκοπός αυτής της διαδικασίας είναι η παραγωγή καλύτερου κώδικα, καθώς ο compiler μπορεί να μελετήσει τον κώδικα σε μία φάση μεταγλώττισης και να συγκρατήσει σχετικές λεπτομέρειες, τις οποίες μπορεί να χρησιμοποιήσει σε κάποιο επόμενο στάδιο, με σκοπό την βελτίωση της ποιότητας της τελικής μεταγλώττισης.

Η σύσταση μιας ενδιάμεσης αναπαράστασης καθιστά δυνατή την προσθήκη πολλαπλών φάσεων στην διαδικασία της μεταγλώττισης. Ο δημιουργός ενός compiler μπορεί να εισάγει ένα τρίτο κομμάτι στην διαδικασία της μεταγλώττισης ενδιάμεσα στο front end και στο back end. Αυτό το ενδιάμεσο κομμάτι, γνωστό και ως optimizer, δέχεται ως είσοδο μία ενδιάμεση αναπαράσταση και παράγει μία αντίστοιχη σημασιολογικά αναπαράσταση ως έξοδο. Χρησιμοποιώντας την ενδιάμεση αναπαράσταση ως διεπαφή, η εισαγωγή αυτού του ενδιάμεσου

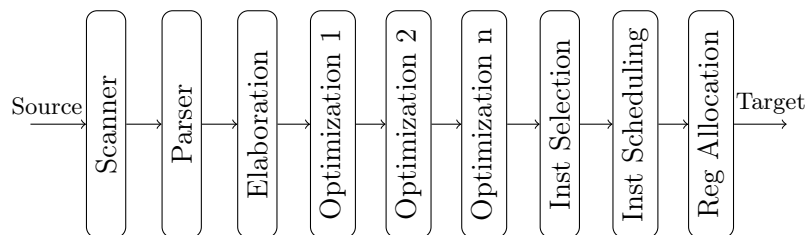
γενικού σταδίου μεταγλώττισης μπορεί να γίνει προκαλώντας ελάχιστη διαταραχή στην ροή εκτέλεσης του compiler[15]. Η δομή που προκύπτει ονομάζεται τριφασική, σε αντίθεση με την προηγούμενη διφασική δομή που παρουσιάστηκε παραπάνω.



Σχήμα Ε'.8: Ροή λειτουργίας ενός τριφασικού compiler.

Ο optimizer είναι ένας μετατροπέας ενδιάμεσων αναπαράστασεων, ο οποίος έχει ως σκοπό την βελτίωση αυτών. Ο optimizer μπορεί να περάσει μία ή περισσότερες φορές πάνω από την IR, να αναλύσει την IR και να την ξαναγράψει. Η IR μετά από αυτή τη διαδικασία μπορεί ενδεχομένως να παράξει ένα γρηγορότερο ή μικρότερο σε μέγεθος τελικό πρόγραμμα, περνώντας από το back end του compiler.

Τυπικά, η τριφασική δομή αναπαριστά τον κλασικό compiler με λειτουργίες βελτιστοποίησης. Πρακτικά, κάθε φάση είναι διαιρεμένη εσωτερικά σε μία σειρά από περάσματα. Το front end αποτελείται από δύο ή τρία περάσματα όπου αναλαμβάνουν τις λεπτομέρειες της αναγνώρισης δόκιμων πηγαίων προγραμμάτων και την παραγωγή της αρχικής ενδιάμεσης αναπαράστασης του προγράμματος. Το μεσαίο κομμάτι περιλαμβάνει περάσματα που πραγματοποιούν διάφορες βελτιστοποιήσεις. Ο αριθμός και ο σκοπός αυτών των περασμάτων διαφέρει από compiler σε compiler. Το back end αποτελείται από μια σειρά περασμάτων, όπου κάθε ένα από αυτά φέρνει την ενδιάμεση αναπαράσταση πιο κοντά στο instruction set του επιθυμητού συστήματος [15].



Σχήμα Ε'.9: Η δομή ενός τυπικού compiler.

Ε'.1.1 Front End

Προτού ο compiler προσπαθήσει να μεταφράσει μία έκφραση σε εκτελέσιμο κώδικα μηχανής, πρέπει πρώτα να καταλάβει το συντακτικό και τη σημασιολογία της. Υπεύθυνο για αυτή

τη διαδικασία είναι το front end. Εάν βρεθεί ότι ο εισαχτέος κώδικας είναι ορθός, τότε το front end δημιουργεί μια αναπαράσταση του κώδικα εντός της ενδιάμεσης αναπαράστασης του compiler. Εάν βρεθεί ότι ο εισαχτέος κώδικας δεν είναι ορθός, τότε ο χρήστης λαμβάνει ένα διαγνωστικό μήνυμα σφάλματος, το οποίο έχει ως σκοπό να βοηθήσει τον χρήστη να διαγνώσει τα προβλήματα του εισαχτέου κώδικα [15].

Για να ελεγχθεί το συντακτικό του κώδικα εισόδου, ο compiler πρέπει να συγκρίνει την δομή του κώδικα έναντι ενός ορισμού της εκάστοτε γλώσσας προγραμματισμού. Κάτι τέτοιο απαιτεί την ύπαρξη ενός επίσημου ορισμού, ενός αποδοτικού μηχανισμού ελέγχου της πιστότητας του κώδικα έναντι αυτού του ορισμού, και ένα σχέδιο για την συνέχεια της διαδικασίας σε περίπτωση εσφαλμένης εισόδου.

Με μαθηματικούς όρους, μία πηγαία γλώσσα είναι ένα σύνολο, συνήθως άπειρο, από αλφαριθμητικά ορισμένα από κάποια περατά σύνολα κανόνων, τα οποία χαρακτηρίζονται και ως γραμματική. Δύο ξεχωριστά περάσματα στο front end, το πέραςμα του scanner και του parser, καθορίζουν αν ο εισαχτέος κώδικας είναι μέλος του συνόλου των ορθών προγραμμάτων όπως ορίζεται από την γραμματική.

Ε'.1.2 Scanner

Οι γραμματικές των προγραμματιστικών γλωσσών συνήθως αναφέρονται σε λέξεις βασισμένες στο μέρος του λόγου τους, κάτι που αποκαλείται μερικές φορές και ως συντακτική κατηγορία. Βασίζοντας τους κανόνες γραμματικής σε μέρη λόγου επιτρέπει σε ένα μοναδικό κανόνα να περιγράφει πολλές προτάσεις.

Το πρώτο βήμα ως προς την κατανόηση του συντακτικού μιας πρότασης είναι η αναγνώριση των διακριτών λέξεων στο εισαχτέο πρόγραμμα και η κατηγοριοποίηση κάθε λέξης ως κάποιο μέρος του λόγου. Εντός ενός compiler αυτή η διαδικασία υπόκειται στο πέραςμα του scanner. Ο scanner λαμβάνει μία ροή από χαρακτήρες και την μετατρέπει σε μία ροή από κατηγοριοποιημένες λέξεις στη μορφή (p, s) , όπου p είναι το μέρος του λόγου της λέξης και s είναι η ορθογραφία της.

Ε'.1.3 Parser

Στο επόμενο βήμα, ο compiler αντιστοιχεί την ροή των κατηγοριοποιημένων λέξεων έναντι των κανόνων που ορίζουν το συντακτικό της εισαχτέας γλώσσας. Το πέραςμα του compiler

που διεξάγει αυτή τη διαδικασία ονομάζεται parser. Καθώς το scrubj βασίζεται ιδιαίτερα στον parser που είναι ενσωματωμένος στον GCC είναι θεμιτό να γίνει περαιτέρω ανάλυση αυτού του περάσματος.

Προκειμένου να περιγραφεί το συντακτικό μιας προγραμματιστικής γλώσσας χρειάζεται κάποιου είδους σημειογραφία. Η σημειογραφία που χρησιμοποιείται συνήθως είναι αυτή των γραμματικών χωρίς συμφραζόμενα ή CFG. Το συντακτικό ως μοντέλο μπορεί να οριστεί και ως γραμματική G . Η G είναι ένα σύνολο από κανόνες που περιγράφουν τον τρόπο διαμόρφωσης δόκιμων προτάσεων.

Ο τυπικός ορισμός μιας γραμματικής G χωρίς συμφραζόμενα αποτελείται από ένα σύνολο (T, NT, S, P) όπου:

- T είναι ένα σύνολο τερματικών, τα οποία είναι τα βασικά σύμβολα από τα οποία σχηματίζονται συμβολοσειρές. Ακόμη, είναι τα πρώτα συστατικά εξόδου που αποδίδονται από τον scanner.
- NT είναι ένα σύνολο από μη τερματικά, τα οποία είναι συντακτικές μεταβλητές που υποδηλώνουν σύνολα από συμβολοσειρές.
- S είναι ένα μη τερματικό, το οποίο χαρακτηρίζεται ως το αρχικό σύμβολο και το σύνολο των συμβολοσειρών που υποδηλώνει, είναι η γλώσσα που παράγεται από τη γραμματική.
- P είναι ένα σύνολο από παραγωγές. Κάθε παραγωγή έχει την μορφή $NT \rightarrow (T \cup NT)^+$, δηλαδή ένα μη τερματικό το οποίο ονομάζεται και ως η αριστερή πλευρά της παραγωγής, το σύμβολο $\rightarrow$ ή το $::=$, και ένα σώμα ή δεξιά πλευρά που αποτελείται από μηδέν ή περισσότερα τερματικά και μη-τερματικά.

Οι παραγωγές που προκύπτουν από την ανάλυση μιας πρότασης εισαχτέας γλώσσας τυπικά αναπαρίστανται από συντακτικά δέντρα. Ένα συντακτικό δέντρο είναι μια γραφική αναπαράσταση ενός σχηματισμού παραγωγών που φιλτράρει βαθμιαία τη σειρά με την οποία οι παραγωγές εφαρμόζονται για να αντικαταστήσουν τα μη-τερματικά. Κάθε εσωτερικό κόμβος ενός συντακτικού δέντρου αναπαριστά τη χρήση μιας παραγωγής [2].

Έστω πρόταση $(a + b) \times c$ η οποία συμβολίζεται ως E . Έστω γραμματική G :

1	$E \rightarrow (E)$
2	$\quad \mid E \text{ Op } n$
3	$\quad \mid n$
4	$\text{Op} \rightarrow +$
5	$\quad \mid -$
6	$\quad \mid \times$
7	$\quad \mid \div$

Ακόμη, ως Op συμβολίζεται ο τελεστής και ως n οι συντακτικές κατηγορίες. Αν η εφαρμογή των κανόνων γίνει με την σειρά (2,6,1,2,4,3), τότε:

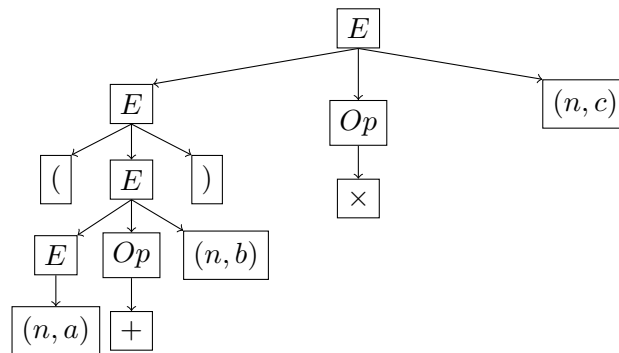
Κανόνας	Προτασιακή Μορφή
	E
2	$E \text{ Op } n$
6	$E \times n$
1	$(E) \times n$
2	$(E \text{ Op } n) \times n$
4	$(E + n) \times n$
3	$(n + n) \times n$

Όπως είναι εμφανές, αν μία πρόταση όπως η $a + b \times c$ μετατρέπεται σε $(n, a) + (n, b) \times (n, c)$ μετά το πέρασμα του scanner, τότε το αποτέλεσμα του parser πρέπει να είναι είτε ένα παράγωγο της εισακτέας πρότασης, είτε ένα μήνυμα σφάλματος σε περίπτωση μη έγκυρου προγράμματος. Για μία μη αμφισβητήσιμη γλώσσα, ένα συντακτικό δέντρο είναι αντίστοιχο ενός παραγώγου, όποτε μπορεί να θεωρηθεί και ως η έξοδος του parser [15].

Μία από τις δυσκολίες του parsing είναι η ανακάλυψη των γραμματικών συνδέσεων μεταξύ των φύλλων ενός συντακτικού δέντρου και της ρίζας [2]. Η προσέγγιση που ακολουθεί ο GCC ονομάζεται Top-Down Recursive-Descent.

Αν και η ανάλυση των διαφορετικών μεθοδολογιών parsing ξεφεύγουν από τα πλαίσια αυτής της πτυχιακής εργασίας, είναι θεμιτό να γίνει μια σύντομη αναφορά στο είδος του parser που χρησιμοποιεί ο GCC.

Ένας top-down parser ξεκινά την ανάλυση του από την ρίζα του συντακτικού δέντρου και



Σχήμα Ε'.10: Το αντίστοιχο συντακτικό δέντρο.

συστηματικά επεκτείνει το δέντρο προς τα κάτω, μέχρι τα φύλλα να αντιστοιχούν στις κατηγοριοποιημένες λέξεις που έχουν επιστραφεί από τον scanner. Σε κάθε σημείο, η διαδικασία εξετάζει το μερικώς κατασκευασμένο δέντρο. Έπειτα, επιλέγει ένα μη τερματικό σύμβολο στο χαμηλότερο επίπεδο του δέντρου και το επεκτείνει προσθέτοντας παιδιά που αντιστοιχούν δεξιότερο σχηματισμό κάποιας παραγωγής για αυτό το μη τερματικό [15]. Η διαδικασία αυτή συνεχίζεται μέχρι:

- το παρόν επίπεδο του συντακτικού δέντρου να περιέχει μόνο τερματικά σύμβολα, και η ροή εισόδου να έχει εξουθενωθεί.
- να υπάρχει μια ξεκάθαρη αναντιστοιχία μεταξύ του παρόντος επίπεδο του μερικώς κατασκευασμένου συντακτικού δέντρου και της ροής εισόδου.

Ένας recursive-descent parser, είναι ένας parser που είναι δομημένος ως ένα σύνολο από αμοιγώς αναδρομικές διαδικασίες, μία για κάθε μη τερματικό σύμβολο εντός της γραμματικής. Η διαδικασία που αντιστοιχεί σε ένα μη τερματικό A αναγνωρίζει όλες τις περιπτώσεις του A εντός της ροής εισόδου. Για να αναγνωριστεί ένα μη τερματικό B σε κάποιο δεξιότερο σχηματισμό της πλευράς A , ο parser καλεί μια διαδικασία που αντιστοιχεί στο μη τερματικό B . Έτσι, η ίδια η γραμματική δρα ως οδηγός για την υλοποίηση του parser [2][15].

Ε'.1.4 Intermediate Representations

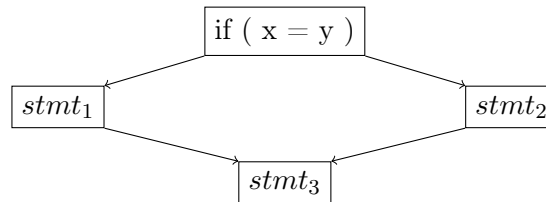
Προκειμένου να μεταφερθούν πληροφορίες από το ένα πέρασμα στο άλλο (π.χ. από το στάδιο της συντακτικής ανάλυσης σε κάποιο στάδιο βελτιστοποίησης), ένας compiler χρειάζεται μία αναπαράσταση όλης της γνώσης που έχει προκύψει από το πρόγραμμα που μεταγλωττίζεται.

Έτσι, σχεδόν όλοι οι compilers χρησιμοποιούν κάποια μορφή ενδιάμεσης αναπαράστασης για να καταστήσουν ένα μοντέλο του κώδικα που αναλύεται, μεταφράζεται και βελτιστοποιείται [15].

Ένας τύπος ενδιάμεσης αναπαράστασης είναι τα Abstract Syntax Trees τα οποία ουσιαστικά προκύπτουν από τα προαναφερθέντα συντακτικά δέντρα, απλώς χωρίς εξωτερικούς κόμβους. Παρόλο που τα δέντρα παρέχουν μια φυσική αναπαράσταση της γραμματικής δομής του πηγαίου κώδικα, όπως αναλύεται από τον parser, η άκαμπτη δομή τους τα καθιστά λιγότερο χρήσιμα για την αναπαράσταση άλλων ιδιοτήτων. Για την μοντελοποίηση αυτών, συνήθως οι compilers χρησιμοποιούν πιο γενικούς γράφους ως ενδιάμεσες αναπαραστάσεις [15].

Προτού γίνει αναφορά σε διάφορους τύπους γράφων που χρησιμοποιούνται εντός των μεταγλωττιστών, είναι συνετό να οριστεί η έννοια του basic block. Ένα basic block είναι μία ακολουθία από εντολές που πάντα εκτελούνται μαζί, εκτός εάν κάποια εντολή υψώσει κάποια εξαίρεση. Ο έλεγχος πάντα εισέρχεται εντός του basic block στην πρώτη εντολή και εξέρχεται στην τελευταία.

Μία συχνή ενδιάμεση αναπαράσταση είναι οι control-flow graphs. Ένας control-flow graph μοντελοποιεί την ροή ελέγχου μεταξύ των basic block ενός προγράμματος. Ένα CFG είναι ένας κατευθυνόμενος γράφος $G = (N, E)$. Κάθε κόμβος $n \in N$ αντιστοιχεί σε ένα basic block. Κάθε ακμή $e = (n_i, n_j) \in E$ αντιστοιχεί σε μία πιθανή μεταφορά ελέγχου από κάποιο block n_i σε κάποιο block n_j .



Σχήμα Ε'.11: Παράδειγμα ενός CFG.

Έστω κόμβος εισόδου n_0 και μοναδικός κόμβος εξόδου n_f . Εντός ενός CFG για μια διαδικασία, το n_0 αντιστοιχεί στο σημείο εισόδου μίας διαδικασίας. Αν μια διαδικασία έχει πολλαπλές εισόδους, ο compiler μπορεί να εισάγει ένα μοναδικό n_0 και να προσθέσει ακμές από τον n_0 στο κάθε πραγματικό σημείο εισόδου. Ομοίως, ο n_f αντιστοιχεί στην έξοδο της διαδικασίας. Πολλαπλές εξοδοί είναι πιο συχνές από πολλαπλές εισόδους, αλλά ο compiler μπορεί εύκολα να προσθέσει ένα μοναδικό n_f και να συνδέσει κάθε πραγματική έξοδο σε αυτόν [15].

Καθώς οι CFG μπορούν να γίνουν υπερβολικά μεγάλοι στην περίπτωση μεγάλων προγραμμάτων, διάφοροι μετασχηματισμοί λαμβάνουν μέρος προκειμένου να μειωθεί ο όγκος τους. Ένας από αυτούς τους μετασχηματισμούς, και η κύρια πηγή δεδομένων του εργαλείου Scrubj, είναι οι call graphs [38].

Ένας call graph έχει ένα κόμβο για κάθε συνάρτηση και μία ακμή για κάθε μέρος κλήσης αυτής. Έτσι, οι κλήσεις κώδικα q έστω από τρία διαφορετικά σημεία σε ένα πρόγραμμα p εντός του call graph έχουν τρεις ακμές τύπου (p, q) , μία για κάθε κλήση [15].

Ε'.2 Ιδιαιτερότητες του GCC

Προκειμένου να γίνουν κατανοητές κάποιες ιδιοσυγκρασίες του κώδικα του εργαλείου Scrubj, πρέπει να εξηγηθούν κάποιες διαφορές στις διαδικασίες του GCC από την τυπική θεωρία, όπως επίσης και κάποιες σχεδιαστικές επιλογές και ορολογίες που εμπεριέχονται σε αυτόν.

Ε'.2.1 GENERIC

Ο σκοπός της ενδιάμεσης αναπαράστασης GENERIC είναι να προσφέρει έναν ανεξαρτήτου γλώσσας τρόπο αναπαράστασης μίας ολόκληρης συνάρτησης σε δέντρα. Σε αυτό το στάδιο ο compiler δεν εφαρμόζει inlining [49] [57].

Ε'.2.2 GIMPLE

Το GIMPLE είναι ένα απλοποιημένο υποσύνολο του δέντρου GENERIC, με σκοπό την χρήση του στα στάδια βελτιστοποίησης του compiler. Το GIMPLE διατηρεί πολλά από την δομή των συντακτικών δέντρων όπως την λεκτική έκταση και τις δομές επιλογής. Οι εκφράσεις διασπώνται σε κώδικα τριών διευθύνσεων [2], χρησιμοποιώντας προσωρινές μεταβλητές για την συγκράτηση των ενδιάμεσων τιμών [49][58].

Ε'.2.3 Control Flow Graph

Η δομή ενός CFG δεν διαφέρει στην υλοποίησή της εντός του GCC. Είναι σημαντικό να σημειωθεί ότι η αναπαράσταση του διατηρείται κατά την διάρκεια της μεταγλώττισης, παρ' όλους τους χειρισμούς επί του γράφου μεταξύ των διαφόρων περασμάτων [55].

Ε'.3 Scrubj

Το σύστημα ανάλυσης κώδικα Scrubj αποτελείται από τέσσερα δομικά στοιχεία, τα οποία σε συνδυασμό πετυχαίνουν την καταγραφή και εισαγωγή των συναρτήσεων ενός προγράμματος γραμμένου σε γλώσσα C στη βάση δεδομένων γράφων Neo4j. Κάθε συνάρτηση αναπαρίσται ως κόμβος, ενώ κάθε κλήση συνάρτησης αναπαρίσται ως ακμή. Με το πέρας της διαδικασίας μεταγλώττισης ενός προγράμματος και της επεξεργασίας των δεδομένων από το Scrubj, ο χρήστης πρέπει τυπικά να έχει μία αναπαράσταση του call graph εισηγμένη στην βάση δεδομένων, στην οποία μπορεί να θέσει ερωτήματα με την γλώσσα ερωτημάτων Cypher.

Τα προαναφερθέντα δομικά στοιχεία είναι τα εξής: ο GCC, ο collector, ο receiver και η βάση δεδομένων Neo4j.

Σε υψηλό επίπεδο η λειτουργία του συστήματος είναι η παρακάτω:

- Η ροή των δεδομένων ξεκινάει από την διαδικασία της μεταγλώττισης. Ο compiler ξεκινά την διαδικασία της μεταγλώττισης βασισμένος στις οδηγίες του χρήστη, οι οποίες μπορεί να χρησιμοποιούν ειδικές ιδιότητες του GCC. Με την προσθήκη μιας επιπλέον ιδιότητας ο GCC διοχετεύει πληροφορίες στο πρόγραμμα collector, το οποίο εκτελείται αυτόματα από τον GCC κατά τη διάρκεια της μεταγλώττισης.
- Ο collector δέχεται τα δεδομένα από τον compiler, τα επεξεργάζεται, τα κατηγοριοποιεί και τα αποστέλλει στον receiver.
- Ο receiver δέχεται τα δεδομένα από τον collector και αναλόγως την κατηγορία αυτών (κόμβος ή ακμή), τα καταχωρεί στη βάση δεδομένων.
- Η βάση δεδομένων, μέσω της διαδικτυακής διεπαφής της, έχει την δυνατότητα να λαμβάνει και να απαντάει ερωτήματα σχετικά με τον γράφο ή γράφους που έχουν προκύψει.



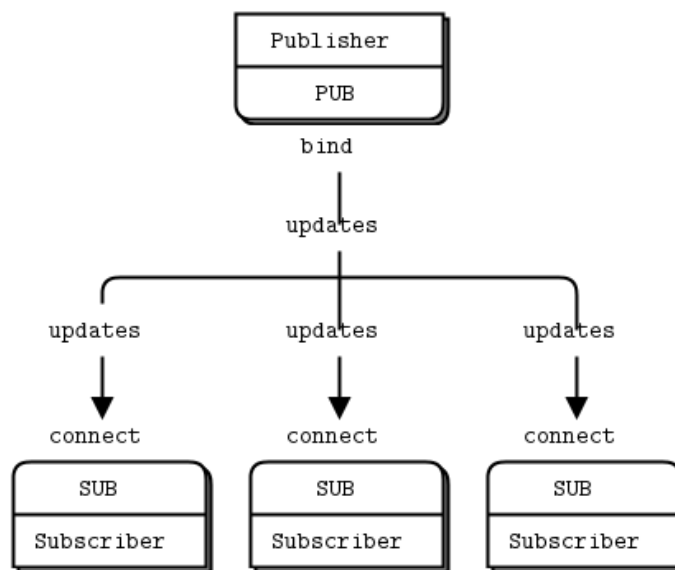
Σχήμα Ε'.12: Η δομή του συστήματος Scrubj.

Ε'.3.1 Μοντέλο Επικοινωνίας

Προκειμένου να εδραιωθεί η επικοινωνία μεταξύ κάποιων δομικών μερών, γίνεται χρήση της βιβλιοθήκης ZeroMQ. Η ZeroMQ είναι μια υψηλής επίδοσης, ασύγχρονη βιβλιοθήκη αποστολής μηνυμάτων, στοχευμένη ως προς τις καταναεμημένες ή τις ταυτόχρονες εφαρμογές. Αν και η ίδια παρέχει μια δομή ουράς μηνυμάτων, έχει την δυνατότητα να λειτουργεί χωρίς αυτοτελή διακομιστή μηνυμάτων [8].

Η ZeroMQ υποστηρίζει διάφορα κοινά μοτίβα αποστολής μηνυμάτων όπως το publisher/subscriber, request/reply, client/server και άλλα, πάνω από διάφορα πρωτοκόλλα όπως τα tcp, uds, multicast, WebSocket και άλλα [8].

Στο Scrubj γίνεται χρήση του μοτίβου publisher/subscriber πάνω από το πρωτόκολλο ipc το οποίο είναι ένα τύλιγμα των UNIX domain sockets [54].



Σχήμα Ε'.13: Μια τοπολογία publisher/subscriber υπό το ZeroMQ [30].

Φυσικά, όπως αναγράφεται στο σχήμα Ε'.12, δεν υπάρχει πολλαπλός αριθμός από subscribers παρά μόνο του receiver. Όμως, επειδή δεν μπορεί να καθοριστεί ακριβώς σε ποια στιγμή γίνεται η αποστολή δεδομένων από τον collector, καθώς λόγω μιας ιδιαιτερότητας της εκτέλεσής του από τον GCC, το socket καταστρέφεται και επαναδημιουργείται πολλές φορές κατά τη διάρκεια αυτής. Αυτό σημαίνει ότι ο receiver πρέπει να μπορεί να δέχεται την συχνή εδραίωση socket και την αποδοχή δεδομένων όποτε παρέχονται από τον collector.

Ε'.3.2 Collector

Ο collector επιτελεί τον σκοπό της αρχικής συλλογής και επεξεργασίας δεδομένων. Για να γίνει κατανοητός ο κώδικας και η εσωτερική λειτουργία του collector, πρέπει πρώτα να γίνει μια σύντομη αναφορά στο gcc-python-plugin.

Ως ένα βασικό δομικό συστατικό του συστήματος είναι ο GCC, ο οποίος προς το παρόν δεν υποστηρίζει την εξαγωγή επιλεγμένων δεδομένων από το front end του με κάποια ή κάποιες συγκεκριμένες επιλογές μεταγλώττισης. Συνεπώς οι δύο πιθανοί τρόποι εξαγωγής των θεμιτών δεδομένων είναι είτε η χρήση μιας τροποποιημένης έκδοσης του GCC, είτε η χρήση των GCC compiler plugins [59] που προσδίδουν πρόσβαση σε ένα API το οποίο μπορεί να ελέγξει και να τροποποιήσει την λειτουργία του compiler κατά τη διάρκεια της εκτέλεσής του.

Όπως είναι εμφανές, καμία από τις παραπάνω λύσεις δεν είναι ιδιαίτερα απλή, καθώς και οι δύο απαιτούν την γνώση των εσωτερικών δομικών στοιχείων του GCC. Την απλοποίηση στην πρόσβαση και παραλλαγή της εσωτερικής λειτουργίας του GCC δίνει το gcc-python-plugin. Το συγκεκριμένο plugin αποτελεί δημιουργία ενός προγραμματιστή της Red Hat εν ονόματι David Malcolm, και έχει ως σκοπό το τύλιγμα του αρχικού plugin API που προσφέρεται από τον GCC στη γλώσσα python [45].

Παρακάτω παρατίθεται ο κώδικας του collector, όπου έπειτα από αυτόν ακολουθεί η ανάλυσή του. Ο κώδικας είναι γραμμένος στην προγραμματιστική γλώσσα Python 3.

```
1 import gcc
2 import re
3 import zmq
4 import time
5 import pickle
6
7 # Creates a zmq context and the sockets needed
8 def connect():
9     context = zmq.Context()
10
11     # Socket to establish connection with client
12     publisher = context.socket(zmq.PUB)
13     publisher.bind("ipc:///tmp/feeds/0")
14     # Set SNDHWM, in case of slow subscribers
15     publisher.sndhwm = 1100000
16     time.sleep(0.2)
```

```

17
18     return publisher
19
20 # Replaces the temporary variables in the gimple statements
21 def unfold(tvars, temp_var):
22     restr1 = r'[A-Z]\.\w+'
23     restr2 = r'_[0-9]+'
24     rexp1 = re.compile(restr1)
25     rexp2 = re.compile(restr2)
26
27     try:
28         argument = tvars["'" + temp_var + "'"]
29     except KeyError:
30         return 'Error_arg'
31
32     while True:
33         m1 = rexp1.search(argument)
34         m2 = rexp2.search(argument)
35
36         if (m1 == None and m2 == None):
37             break
38
39         if m1:
40             temp_var = m1.group()
41         if m2:
42             temp_var = m2.group()
43
44         try:
45             if m1:
46                 argument = rexp1.sub(tvars["'" + temp_var + "'"],
47                                     argument)
48             if m2:
49                 argument = rexp2.sub(tvars["'" + temp_var + "'"],
50                                     argument)
51         except KeyError:
52             argument = 'Error_arg'
53
54     return argument
55
56 # Makes a dictionary with all the gimple statements UIDs
57 def intake(stmt, tvars):

```

```

56     restr1 = r'(^[A-Z]\.\w+)(\s=\s)([^\;]*)' # group(1) = UID, group(3) =
        stmt
57     restr2 = r'(^_[0-9]+)(\s=\s)([^\;]*)' # group(1) = _, group(3) = stmt
58     rexp1 = re.compile(restr1)
59     rexp2 = re.compile(restr2)
60
61     m1 = rexp1.search(str(stmt))
62     m2 = rexp2.search(str(stmt))
63     if m1:
64         tvars["'" + m1.group(1) + "'"] = m1.group(3)
65
66     if m2:
67         tvars["'" + m2.group(1) + "'"] = m2.group(3)
68
69     return tvars
70
71 def on_pass_execution(p, fn):
72     if p.name == '*free_lang_data':
73         global publisher
74         fname = ""
75         arguments = {}
76         tvars = {}
77         data = {}
78         callees_list = []
79         callers_list = []
80         parent_types_list = []
81         restr1 = r'[A-Z]\.[0-9]+'
82         restr2 = r'_[0-9]+'
83         rexp1 = re.compile(restr1)
84         rexp2 = re.compile(restr2)
85
86         for node in gcc.get_callgraph_nodes():
87
88             try:
89                 fn = node.decl.function
90                 cl = node.callees
91                 cr = node.callers
92
93                 parent = str(fn.decl.name) + '@' + str(fn.decl.location.
                    file) \
94                     + '_L' + str(fn.decl.location.line) + '_C' \

```

```

95         + str(fn.decl.location.column)
96     data['parent'] = parent
97
98     parent_type = str(fn.decl.type.type)
99     data['parent_type'] = parent_type
100
101     for edge in cl:
102         callees_list.append(str(edge.call_stmt.fn) + '@' \
103                             + str(edge.call_stmt.loc.file) + '_L' \
104                             + str(edge.call_stmt.loc.line) + '_C' \
105                             + str(edge.call_stmt.loc.column))
106     data['callees'] = callees_list
107
108     for edge in cr:
109         callers_list.append(str(edge.caller.decl.name) + '@' \
110                             \
111                             + str(edge.caller.decl.location.file) + '_L' \
112                             \
113                             + str(edge.caller.decl.location.line) + '_C' \
114                             \
115                             + str(edge.caller.decl.location.column))
116     data['callers'] = callers_list
117
118     for arg in fn.decl.type.argument_types:
119         parent_types_list.append(str(arg))
120
121     data['parent_argument_types'] = parent_types_list
122 except AttributeError:
123     continue
124
125 try:
126     for bb in fn.cfg.basic_blocks:
127         for stmt in bb.gimple:
128             tvars = intake(stmt, tvars)
129             if isinstance(stmt, gcc.GimpleCall):
130                 fnmatch1 = rexp1.search(str(stmt.fn))
131                 fnmatch2 = rexp2.search(str(stmt.fn))
132
133                 if fnmatch1:
134                     fname = str(unfold(tvars, fnmatch1.group
135                                         ())) \

```

```

132             + '@' + str(stmt.loc.file) \
133             + '_L' + str(stmt.loc.line) + '_C'
134             , \
135             + str(stmt.loc.column)
136         #print(fname)
137     elif fnmatch2:
138         fname = str(unfold(tvars, fnmatch2.group
139             ())) \
140             + '@' + str(stmt.loc.file) \
141             + '_L' + str(stmt.loc.line) + '_C'
142             , \
143             + str(stmt.loc.column)
144         #print(fname)
145     else:
146         fname = str(stmt.fn) + '@' + str(stmt.loc
147             .file) \
148             + '_L' + str(stmt.loc.line) + '_C'
149             , \
150             + str(stmt.loc.column)
151         #print(fname)
152     for i, arg in enumerate(stmt.args):
153         m1 = rexp1.search(str(stmt.args[i]))
154         m2 = rexp2.search(str(stmt.args[i]))
155         if m1:
156             arguments[i] = unfold(tvars, m1.group
157                 ())
158         elif m2:
159             arguments[i] = unfold(tvars, m2.group
160                 ())
161         else:
162             arguments[i] = str(stmt.args[i])
163         #print(arguments)
164         data[fname] = arguments
165         arguments = {}
166
167 except AttributeError:
168     continue
169
170 data_string = pickle.dumps(data)
171 publisher.send(data_string)

```

```

166         data.clear()
167         del callees_list[:]
168         del callers_list[:]
169         del parent_types_list[:]
170
171     if p.name == "ssa":
172         publisher.send(b"GCC_DISCONNECT")
173
174 publisher = connect()
175 gcc.register_callback(gcc.PLUGIN_PASS_EXECUTION, on_pass_execution)

```

Η λειτουργικότητα του κώδικα εξετάζεται από γραμμή σε γραμμή παρακάτω:

- Γραμμή 1-5: Η εισαγωγή των βιβλιοθηκών, όπου gcc είναι η βιβλιοθήκη του gcc-python-plugin, re είναι η βιβλιοθήκη κανονικών εκφράσεων, zmq είναι η βιβλιοθήκη του ZeroMQ, time είναι η βιβλιοθήκη που περιέχει συναρτήσεις σχετικές με τη μέτρηση χρόνου και pickle είναι μία βιβλιοθήκη σειριοποίησης.
- Γραμμή 8-18: Η συνάρτηση connect είναι υπεύθυνη για την εδραίωση της σύνδεσης με τον receiver, όπως αναλύθηκε στην ενότητα του μοντέλου επικοινωνίας. Μια σημαντική λεπτομέρεια είναι ότι υπάρχει ένα διάστημα παύσης 0.2 δευτερολέπτων μέσω της συνάρτησης sleep στη γραμμή 16, προκειμένου να συγχρονιστεί ο publisher με τον subscriber.
- Γραμμή 55-69: Η συνάρτηση intake είναι υπεύθυνη για την καταγραφή των unique ids των εκάστοτε εκφράσεων gimple, που βρίσκονται εντός των basic block που δημιουργεί ο compiler κατά τη διάρκεια της μεταγλώττισης. Με την χρήση κανονικών εκφράσεων γίνεται η αναγνώριση της μορφής των gimple statements. Αν η έκφραση αντιστοιχεί στην κανονική έκφραση, τότε αυτή εισέρχεται σε ένα dictionary με κλειδί του το UID και τιμή την ίδια την έκφραση.
- Γραμμή 21-52: Η συνάρτηση unfold έχει ως σκοπό να ξεδιπλώσει ένα gimple statement, δηλαδή να αντικαταστήσει τα UIDs εντός της αρχικής έκφρασης με αυτά που υπάρχουν στο dictionary που δημιουργήθηκε από την συνάρτηση intake. Η unfold σταματάει την εκτέλεση της όταν το τελικό gimple statement δεν έχει πλέον κανένα UID.
- Γράμμη 71-172: Η συνάρτηση on_pass_execution αποτελεί το κύριο σώμα του collector. Αυτή η συνάρτηση καλείται από τον GCC κάθε φορά που ο ίδιος βρίσκεται στο

στάδιο βελτιστοποίησης `*free_lang_data`, το οποίο είναι ένα στάδιο αμέσως μετά τη δημιουργία του call graph, προτού όμως γίνουν δραστικές αλλαγές στα στοιχεία που τον πλαισιώνουν. Η συνάρτηση συλλέγει στοιχεία από τον κάθε κόμβο του call graph και πιο συγκεκριμένα στις γραμμές 71-120 συλλέγει την συνάρτηση στην οποία βρίσκεται μέσα μια κλήση συνάρτησης (αναφέρεται και ως parent), τον τύπο του parent, τον τύπο των ορισμάτων του parent, το όνομα των συναρτήσεων που καλεί η εν λόγω συνάρτηση και το όνομα των συναρτήσεων που καλούν αυτή τη συνάρτηση. Από τη γραμμή 122 μέχρι τη γραμμή 162 ουσιαστικά γίνεται η καταγραφή όλων των κλήσεων συναρτήσεων που συμβαίνουν στο εκάστοτε basic block. Στις γραμμές 164-169 γίνεται η αποθήκευση, σειριοποίηση και αποστολή όλων των δεδομένων που έχουν εξαχθεί, όπως επίσης και το άδειασμα προσωρινών λιστών.

- Γραμμή 171-172: Αν το στάδιο βελτιστοποίησης έχει φτάσει σε ένα σημείο όπου πλέον δεν συγκρατεί χρήσιμα δεδομένα για το σκοπό του συστήματος, τότε ο collector ενημερώνει τον receiver ότι πρόκειται να αποσυνδεθεί.
- Γραμμή 174: Η έναρξη της σύνδεσης με τον receiver.
- Γραμμή 175: Η καταχώρηση της `on_pass_execution` ως συνάρτηση η οποία εκτελείται σε κάθε πέρασμα βελτιστοποίησης του GCC.

Ε'.3.3 Receiver

Ο σκοπός του receiver είναι η αποδοχή των δεδομένων που αποστέλλονται από τον collector, η κατηγοριοποίηση αυτών και καταχώρισή τους στη βάση δεδομένων γράφων Neo4j.

Ουσιαστικά η κατασκευή του γράφου γίνεται από τον receiver. Η ακριβής διαδικασία θα γίνει κατανοητή μετά την παράθεση του κώδικα.

```

1  import pickle
2  import zmq
3  import pprint
4  import re
5  from neo4j import GraphDatabase
6
7  # Neo4j Functions
8
9  def dbhandle(uri, user, password):
10     driver = GraphDatabase.driver(uri, auth=(user, password))

```

```

11     return driver
12
13 def dbclose(driver):
14     driver.close()
15
16 def create_project(driver, project_name):
17     with driver.session() as session:
18         tx = session.begin_transaction()
19         create_project_node(tx, project_name)
20         tx.commit()
21
22 def create_project_node(tx, project_name):
23     tx.run("CREATE (p:project {name:{project_name}})",
24           project_name=project_name)
25
26 def create_node(driver, loc, fname, ftype, argument_types, project_name):
27     with driver.session() as session:
28         tx = session.begin_transaction()
29         create_function_node(tx, loc, fname, ftype, argument_types,
30                             project_name)
31         tx.commit()
32
33 def create_function_node(tx, loc, fname, ftype, argument_types,
34                          project_name):
35     tx.run("MERGE (n:function:parent {loc:{loc}, name:{fname}}) \
36           ON CREATE SET n.type = {ftype}, \
37           n.argument_types = {argument_types} \
38           ON MATCH SET n.type = {ftype}, \
39           n.argument_types = {argument_types} \
40           WITH n \
41           MATCH (n:function:parent), (p:project) \
42           WHERE n.name={fname} AND p.name={project_name} \
43           MERGE (n)-[r:IN_PROJECT]->(p)", loc=loc,
44           fname=fname, ftype=ftype, argument_types=argument_types,
45           project_name=project_name)
46
47 def create_callee_node(driver, loc, fname, parent_name, parent_loc,
48                        arg_list, project_name):
49     with driver.session() as session:
50         tx = session.begin_transaction()
51         create_callee_function(tx, loc, fname, parent_name, parent_loc,

```

```

        arg_list, project_name)
49     tx.commit()
50
51     def create_callee_function(tx, loc, fname, parent_name, parent_loc,
        arg_list, project_name):
52         tx.run("MATCH (f:function:parent) \
53             WHERE f.name={parent_name} AND f.loc={parent_loc} \
54             MERGE (n:function:callee {loc:{loc}, name:{fname}}) \
55             WITH n \
56             MATCH (n:function:callee), (p:project) \
57             WHERE n.name={fname} AND p.name={project_name} \
58             MERGE (n)-[r:IN_PROJECT]->(p) \
59             WITH n \
60             MATCH (f:function:parent),(n:function:callee) \
61             WHERE f.name={parent_name} AND n.name={fname} \
62             MERGE (f)-[r:CALLS {arguments:{arg_list}}]->(n)",
63                 parent_name=parent_name, parent_loc=parent_loc, loc=loc,
64                 fname=fname, arg_list=arg_list, project_name=project_name)
65
66     # Zeromq PUB-SUB functions
67     def connect():
68         context = zmq.Context()
69
70         # Connect the subscriber socket
71         subscriber = context.socket(zmq.SUB)
72         subscriber.connect('ipc:///tmp/feeds/0')
73         subscriber.setsockopt(zmq.SUBSCRIBE, b'')
74
75         return subscriber
76
77     # Where it all goes down
78     def main():
79         subscriber = connect()
80         count = 0
81         gcc_disconnect = False
82         arg_list = []
83         master = {}
84         restr = r'(.+)(\\@)(.+)'
85         rexp = re.compile(restr)
86
87         # TODO: Add here project name input and add it in the db nodes

```

```

88     project_name = input("Enter the project name: ")
89
90     while True:
91         try:
92             msg = subscriber.recv()
93             if msg == b"GCC_DISCONNECT":
94                 if gcc_disconnect == False: # When ms pass without
95                     receive
96                     subscriber.RCVMTEO = 300000 # the conn will close
97                     gcc_disconnect = True # find alternative when
98                         possible
99                 else:
100                     continue
101             else:
102                 data = pickle.loads(msg)
103                 master[count] = data
104                 pprint.pprint(master[count])
105                 #data.clear()
106                 count += 1
107
108         except zmq.error.Again:
109             print("gcc disconnected")
110             break
111         except KeyboardInterrupt:
112             print("Interrupt received, stopping...")
113             break
114
115     # Add your uri, user and password
116     db = dbhandle("bolt://<ip>:7687", "user", "password")
117
118     # Create project node
119     create_project(db, project_name)
120
121     for key in master:
122         temp = master[key]
123         m1 = rexp.search(str(temp['parent']))
124         #print(str(m.group(3))) this is the loc
125         #print(str(m.group(1))) this is the fname
126         #print(str(temp['parent_type']))
127
128         if m1:

```

```

127         create_node(db, str(m1.group(3)), str(m1.group(1)),
128                     str(temp['parent_type']), str(temp['parent_argument_types']
129                     ),
130                     project_name)
131
132     for callee in temp['callees']:
133         m2 = rexp.search(str(callee))
134
135         try:
136             for arg in temp[callee].values():
137                 arg_list.append(arg)
138         except KeyError:
139             arg_list.append("no_args")
140
141         if m2:
142             create_callee_node(db, str(m2.group(3)), str(m2.group(1)),
143                                str(m1.group(1)), str(m1.group(3)),
144                                arg_list, project_name)
145             del arg_list[:]
146
147     dbclose(db)
148
149 if __name__ == "__main__":
150     main()

```

Η λειτουργικότητα του κώδικα εξετάζεται από γραμμή σε γραμμή παρακάτω:

- Γραμμή 1-5: Η εισαγωγή των βιβλιοθηκών δεν διαφέρει ιδιαίτερα σε σχέση με τον collector, με την μόνη διαφορά να βρίσκεται στην εισαγωγή της βιβλιοθήκης neo4j, η οποία εδραιώνει την επικοινωνία με την βάση δεδομένων Neo4j.
- Γραμμή 9-11: Η συνάρτηση dbhandle πραγματοποιεί την σύνδεση με την βάση δεδομένων μέσω της δημιουργίας του αντικειμένου driver. Στην προκειμένη περίπτωση για την επικοινωνία αυτή χρησιμοποιείται το πρωτόκολλο bolt [35].
- Γραμμή 13-14: Η συνάρτηση dbclose απλά τερματίζει την σύνδεση με την βάση δεδομένων.
- Γραμμή 16-24: Η συνάρτηση create_project (μαζί με την create_project_node) δημιουργεί ένα κόμβο στη βάση δεδομένων με το όνομα του project. Η ύπαρξη αυτού

του κόμβου χρησιμεύει στο να δείχνουν οι υπόλοιποι κόμβοι ότι ανήκουν σε κάποιο συγκεκριμένο project.

- Γραμμή 26-43: Η συνάρτηση `create_node` (μαζί με την `create_function_node`) δημιουργεί τους κόμβους `parent` που αναπαριστούν τις συναρτήσεις στις οποίες γίνεται ο ορισμός μιας συνάρτησης εντός του call graph. Κάθε κόμβος `parent` έχει τις ιδιότητες `name`, `type`, `argument_types` και `loc` (εκ του location), όπου το `format` τοποθεσίας εντός του κώδικα είναι `'file_L(number)_C(number)'`, με το γράμμα L να συμβολίζει την γραμμή και το γράμμα C την στήλη στην οποία βρίσκεται η συνάρτηση.
- Γραμμή 45-64: Η συνάρτηση `create_callee_node` (μαζί με την `create_callee_function`) δεν διαφέρει ιδιαίτερα από την προηγούμενη συνάρτηση, απλά απευθύνεται στις συναρτήσεις οι οποίες καλούν κάποια άλλη συνάρτηση. Οι κόμβοι τύπου `callee` έχουν μόνο τις ιδιότητες `name` και `loc` (με το ίδιο `format`). Ένα πράγμα στο οποίο διαφέρει αυτή η συνάρτηση από την προηγούμενη είναι το ότι αυτή εδραιώνει τις αλυσίδες μεταξύ των κόμβων, οι οποίες με τη σειρά τους αναπαριστούν τις κλήσεις μεταξύ των συναρτήσεων. Οι αλυσίδες αυτές φέρουν τον τύπο `CALLS` και έχουν μία μόνο ιδιότητα, αυτή των ορισμάτων (ως `arguments`).
- Γραμμή 67-75: Η συνάρτηση `connect` πλαισιώνει την σύνδεση μεταξύ του collector. Ακολουθεί το μοντέλο `publisher/subscriber` βάσει του προαναφερθέν πλαισίου επικοινωνίας.
- Γραμμή 78-146: Μεταξύ αυτών των γραμμών βρίσκεται η κύρια συνάρτηση του προγράμματος. Το πρώτο ουσιαστικό κομμάτι της συνάρτησης βρίσκεται μεταξύ των γραμμών 90 και 111. Σε αυτό το κομμάτι ο `receiver` αποδέχεται τα δεδομένα όπως του τα αποστέλλει ο `collector`. Αν ο `receiver` λάβει μήνυμα αποσύνδεσης τότε, εάν δεν λάβει δεδομένα μέσα στα επόμενα 5 λεπτά, διακόπτει την σύνδεση.

Το δεύτερο ουσιαστικό κομμάτι της συνάρτησης βρίσκεται μεταξύ των γραμμών 119 και 144. Σε αυτό το κομμάτι ο `receiver` εξάγει τα δεδομένα από ένα `dictionary` και τα κατηγοριοποιεί βάσει μια κανονικής έκφρασης. Ανάλογα με τον τύπο του κόμβου, γίνεται χρήση μίας από τις παραπάνω συναρτήσεις προκειμένου να εισαχθεί ο κόμβος στη βάση δεδομένων.

Κεφάλαιο Γ΄

Δοκιμή του Συστήματος

ΤΟ αρχικό πρόβλημα όπως έχει οριστεί είναι η εύρεση ενός τρόπου να εισαχθούν οι κλήσεις συναρτήσεων που περιέχονται σε κώδικα σε μια μη σχεσιακή βάση δεδομένων, προκειμένου να μπορούν να γίνουν ερωτήματα με σκοπό την ανακάλυψη μοτίβων ή την καλύτερη κατανόηση αυτού.

Οι αρχικοί στόχοι του συστήματος στο αρχικό στάδιο υλοποίησης ήταν οι εξής:

- Το εργαλείο που θα δημιουργηθεί να αφορά κάποια γλώσσα που έχει ευρεία αποδοχή.
- Να μην χρειάζεται προσαρμογή για κάθε διαφορετικό κομμάτι κώδικα.
- Να είναι εύκολο για έναν τρίτο χρήστη να το χρησιμοποιήσει χωρίς να γνωρίζει πως δουλεύει ο κώδικας.
- Να μην αυξάνει εκθετικά την ώρα που χρειάζεται η διαδικασία της μεταγλώττισης.
- Να μπορεί να ενσωματωθεί σε κάποια διαδικασία αυτόματης συλλογής κώδικα.

Ο βαθμός εκπλήρωσης αυτών των στόχων γίνεται ξεκάθαρος μέσα από τα σχετικά παραδείγματα χρήσης που επακολουθούν. Έπειτα, ακολουθούν διάφορα παραδείγματα ερωτημάτων προς την βάση δεδομένων, με σκοπό την διεξαγωγή αρχέτυπων analytics, προκειμένου να αξιολογηθεί η πρακτική χρησιμότητα του συστήματος. Τελικώς, γίνεται η ανάλυση σχετικών σχεδιαστικών επιτυχιών και αποτυχιών του συστήματος στα πλαίσια που έχουν τεθεί στην παρούσα πτυχιακή εργασία.

Γ'.1 Παράδειγμα Χρήσης

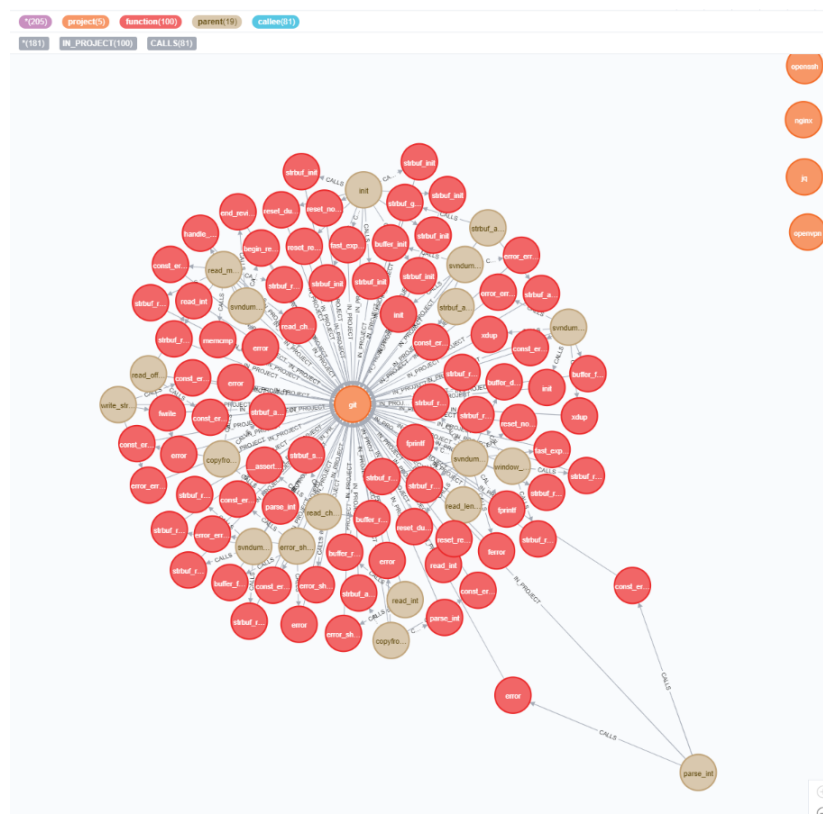
Έστω ότι είναι θεμιτό να χτιστεί ένα εργαλείο γραμμένο στην γλώσσα C. Ένα εργαλείο γραμμένο σε αυτή τη γλώσσα είναι το git . Με μερικές τροποποιήσεις στο Makefile, μπορεί να ενσωματωθεί το plugin που εξάγει τις ζητούμενες πληροφορίες.

```

Set CFLAGS, LDFLAGS and other *FLAGS variables. These might be
# tweaked by config.* below as well as the command-line, both of
# which'll override these defaults.
CFLAGS = -Wall -O0 -g -fno-builtin -fplugin=/home/pxcel/gcc-python-plugin/python.so \
        -fplugin-arg-python-script=/home/pxcel/scrubj/collector.py
LDFLAGS =
CC_LD_DYNPATH = -Wl,-rpath,
BASIC_CFLAGS = -I.
BASIC_LDFLAGS =

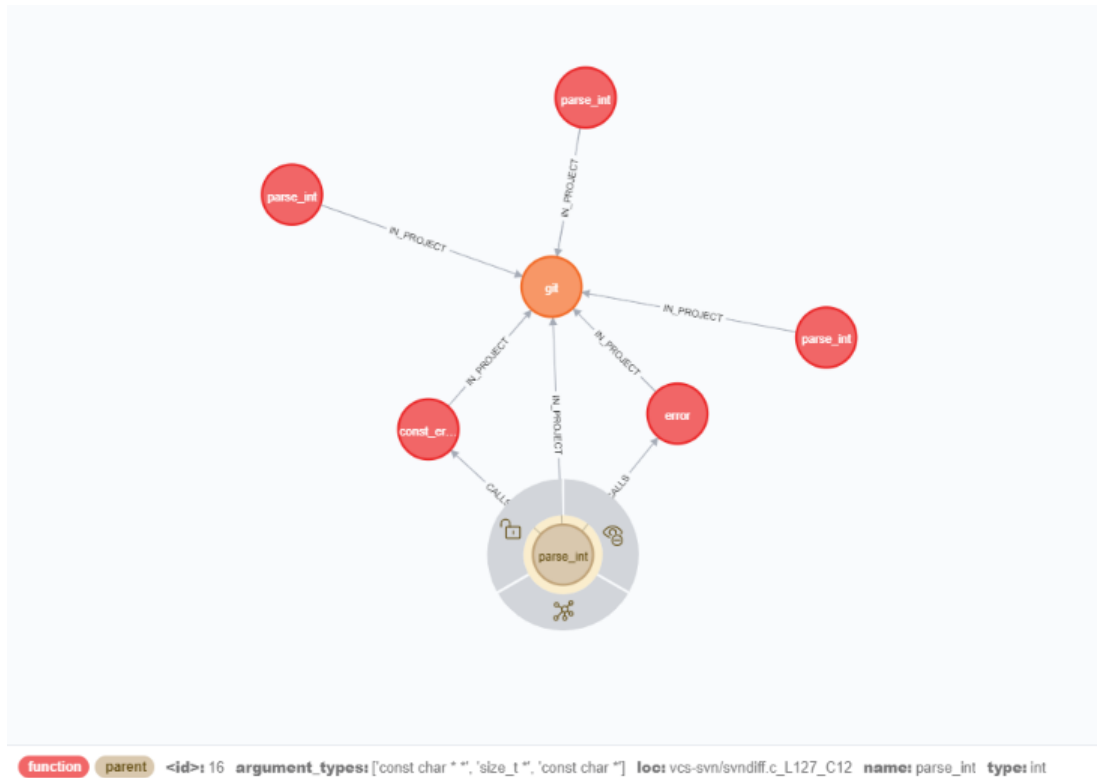
```

Έπειτα, αρκεί να εκτελεστεί ο receiver πριν τη διαδικασία του compilation και το Makefile να ολοκληρωθεί επιτυχώς. Αν δεν υπάρξει σφάλμα στην διαδικασία τότε το αποτέλεσμα στο ερώτημα MATCH (n:project) RETURN n LIMIT 25 στη Neo4j πρέπει να είναι παρόμοιο.



Σε ένα πιο μικροσκοπικό επίπεδο, αν γίνει η εξέταση ενός κόμβου, θα γίνει εμφανές ότι οι

ιδιότητες που περιγράφηκαν στο προηγούμενο κεφάλαιο είναι υπαρκτές.



Γ'.1.1 Ερωτήματα με την Cypher

Κάποια βασικά ερωτήματα που μπορούν να τεθούν στη βάση είναι τα εξής:

1. Να βρεθεί κάθε κόμβος που έχει κάποιο τύπο.
2. Να βρεθεί κάθε κόμβος που έχει κάποιο τύπο ορίσματος.
3. Να βρεθεί κάθε κόμβος που έχει πάνω ή κάτω από ένα αριθμό ορισμάτων.
4. Να βρεθεί κάθε κόμβος που κάνει παραπάνω από κάποιες κλήσεις συναρτήσεων.
5. Να βρεθεί πόσες φορές καλείται μια συνάρτηση x εντός ενός project.
6. Να βρεθεί ποιες είναι οι συναρτήσεις που καλούνται περισσότερες φορές εντός ενός project.
7. Να βρεθεί ποιος είναι ο τύπος και ο αριθμός των ορισμάτων μιας συνάρτησης.

8. Να βρεθεί ποιες συναρτήσεις καλούνται μεταξύ γραμμής α και γραμμής β.

Οι απαντήσεις για τα περισσότερα από αυτά τα ερωτήματα είναι αρκετά απλές.

1. Καθώς ο τύπος βρίσκεται μόνο σε κόμβους parent, το ερώτημα διαμορφώνεται ως:

```
1      match(n:parent)
2      where n.type = 'int'
3      return n
```



2. Το ερώτημα για να βρεθεί κάθε κόμβος που έχει κάποιο τύπο ορίσματος είναι το εξής:

```
1      match(n:parent)
2      where n.argument_types <> "[]" and substring(n.
          argument_types, 1, size(n.argument_types) - 2)
3      contains "'void *'"
4      return n
```



3. Για να βρεθεί κάθε κόμβος που έχει πάνω από ένα αριθμό ορισμάτων πρέπει να γίνει μετατροπή του πίναχα αργυμεντ_טיפες σε ένα string που περιέχει τα περιεχόμενα του. Δυστυχώς η Neo4j δεν υποστηρίζει πλήρεις κανονικές εκφράσεις για μπορεί να εκπληρωθεί τέτοιο ερώτημα. Το πιο κοντινό ερώτημα είναι να εντοπιστεί κάθε κόμβος που έχει πάνω από ένα όρισμα.

```

1      match(n:parent)
2      where n.argument_types <> "[]" and substring(n.
           argument_types, 1, size(n.argument_types) - 2)
3      contains " ,"
4      return n

```

```
$ match(n:parent) where n.argument_types <> "[]" and substring(n.argument_types, 1, size(n.argument_types) - 2_
```



The screenshot shows a query results interface with a sidebar on the left containing icons for Graph, Table (selected), Text, and Code. The main area displays two JSON objects under the heading 'n'. The first object is for 'copyfrom_target' and the second is for 'copyfrom_source'. Both objects have fields for 'name', 'loc', 'type', and 'argument_types'. The 'argument_types' field contains a list of strings representing C types: 'struct window *', 'const char *', 'size_t', and 'const char *'. At the bottom, a status message reads: 'Started streaming 16123 records after 1 ms and completed after 388 ms, displaying first 1000 rows.'

```
{
  "name": "copyfrom_target",
  "loc": "vcs-svn/svndiff.c_L181_C12",
  "type": "int",
  "argument_types": "['struct window *',
'const char *', 'size_t', 'const char
*']"
}
```

```
{
  "name": "copyfrom_source",
  "loc": "vcs-svn/svndiff.c_L168_C12",
  "type": "int",
  "argument_types": "['struct window *',
'const char *', 'size_t', 'const char
*']"
}
```

Started streaming 16123 records after 1 ms and completed after 388 ms, displaying first 1000 rows.

4. Για κάθε κόμβο που κάνει παραπάνω από κάποιες κλήσεις συναρτήσεων αρκεί:

```
1      match (n:parent)-[r]->(b:callee)
2      with count(r) as calls
3      match (n:parent)-[r]->(b:callee)
4      where calls > 2
5      return n
```

\$ match (n:parent)-[r]->(b:callee) with count(r) as calls match (n:parent)-[r]->(b:callee) where calls > 2 return n

Graph

Table

Text

Code

```
{
  "name": "copyfrom_target",
  "loc": "vcs-svn/svndiff.c_L181_C12",
  "type": "int",
  "argument_types": "['struct window *',
'const char * *', 'size_t', 'const char
*']"
```

```
{
  "name": "copyfrom_target",
  "loc": "vcs-svn/svndiff.c_L181_C12",
  "type": "int",
  "argument_types": "['struct window *',
'const char * *', 'size_t', 'const char
```

Started streaming 177432 records after 339 ms and completed after 11560 ms, displaying first 1000 rows.

5. Για το πόσες φορές καλείται μια συνάρτηση x εντός ενός project, έστω συνάρτηση strcmp, τότε:

```
1      match (n:function {name:"strcmp"})-[r]->(z:project {name:"
      git"})
2      return n.name, count(r)
```

\$ match (n:function {name:"strcmp"})-[r]->(z:project {name:"git"}) return n.name, count(r)

Table

Text

Code

n.name	count(r)
strcmp	1949

Started streaming 1 records after 201 ms and completed after 201 ms.

6. Για το ποιες είναι οι συναρτήσεις που καλούνται περισσότερες φορές εντός ενός project, αρκεί:

```

1      match (n:function)-[r]->(z:project {name:"git"})
2      return n.name, count(r) order by count(r) desc

```

\$ match (n:function)-[r]->(z:project {name:"git"}) return n.name, count(r) order by count(r) desc		📄	🔍
Table			
Text			
Code			
	n.name	count(r)	
	" "	3010	
	"die"	2101	
	"strcmp"	1949	
	"free"	1698	
	"const_error"	1272	
	"strbuf_release"	1130	
	"error"	1052	
	"strbuf_addf"	745	
	"strbuf_addstr"	734	
	"old_to_hex"	710	
	"strlen"	690	
	"printf"	618	
	"fprintf"	569	
	"strbuf_setlen"	535	
	"argv_array_push"	477	
	"BUG_fix"	427	
Started streaming 7981 records after 516 ms and completed after 583 ms, displaying first 1000 rows.			

Το αποτέλεσμα μπορεί να περιοριστεί με τον όρο LIMIT στον αριθμό των πρώτων x αποτελεσμάτων.

\$ match (n:function)-[r]->(z:project {name:"git"}) return n.name, count(r) order by count(r) desc LIMIT...		📄	🔍
Table			
Text			
Code			
	n.name	count(r)	
	" "	3010	
	"die"	2101	
	"strcmp"	1949	
	"free"	1698	
	"const_error"	1272	
	"strbuf_release"	1130	
	"error"	1052	
	"strbuf_addf"	745	
	"strbuf_addstr"	734	
	"old_to_hex"	710	
Started streaming 10 records after 259 ms and completed after 260 ms.			

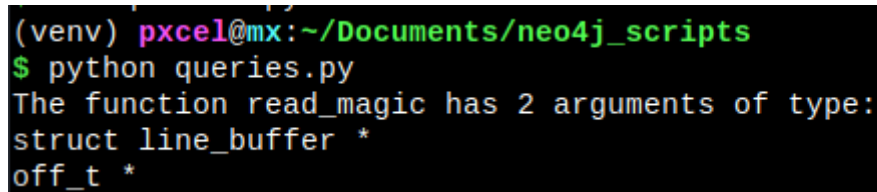
7. Για να απαντηθεί ποιος είναι ο τύπος και ο αριθμός των ορισμάτων μιας συνάρτησης δεν είναι αρκετή μόνο η χρήση της γλώσσας ερωτημάτων Cypher. Ο παρακάτω κώδικας σε Python εσωκλείει το επιθυμητό ερώτημα στην γλώσσα Cypher (έστω για την συνάρ-

τηση `read_magic` του `git`) και επεξεργάζεται την έξοδο του ερωτήματος, έτσι ώστε να προκύψει το επιθυμητό αποτέλεσμα.

```

1      import re
2      import ast
3      from neo4j import GraphDatabase
4
5      uri = "bolt://<host>:7687"
6      driver = GraphDatabase.driver(uri, auth=("user", "password"))
7
8      def print_argument_types(tx, fname, pname):
9          for record in tx.run("MATCH (n:parent)-[r]->(z:project)
10
11                                  "WHERE n.name = {fname} and z.name = {
12                                  pname}"
13
14                                  "return n.argument_types", fname=fname, pname=
15                                  pname):
16              arr = ast.literal_eval(str(record["n.argument_types"]
17              ))
18              print("The function {} has {} arguments of type:".
19                    format(fname, len(arr)))
20              for item in arr:
21                  print(item)
22
23      with driver.session() as session:
24          session.read_transaction(print_argument_types, "
25          read_magic", "git")

```



```

(venv) pxcel@mx:~/Documents/neo4j_scripts
$ python queries.py
The function read_magic has 2 arguments of type:
struct line_buffer *
off_t *

```

8. Επιπλέον κώδικας σε Python πρέπει να γραφτεί για να βρεθούν οι συναρτήσεις που καλούνται μεταξύ μίας γραμμής α και μίας γραμμής β .

```

1      def within_lines(tx, start, finish, filename, pname):
2          restr = r'(L)(\d+)'
3          rexp = re.compile(restr)

```

```

4
5         for record in tx.run("MATCH (n:callee)-[r]->(z:project)
6             "WHERE n.loc contains ({filename}) and z.name = {
7                 pname} "
8             "return n.name, n.loc", filename=filename, pname=
9                 pname):
10                match = rexp.search(record["n.loc"])
11                if match:
12                    if int(match.group(2)) >= start and int(match.
13                        group(2)) <= finish:
14                        print(record["n.name"])
15
16 with driver.session() as session:
17     session.read_transaction(within_lines, 1, 100, "difftool
18         .c", "git")
19 driver.close()

```

```

(venv) pxcel@mx:~/Documents/neo4j_scripts
$ python queries.py
strcmp
xstrdup
strcmp
git_config_bool
git_default_config
run_command_v_opt
error
const_error
strtol
error
const_error
strtol
error
const_error
get_oid_hex
error
const_error
error
const_error
get_oid_hex
error
const_error
error
const_error
error
const_error
error
const_error
error
const_error
strbuf_setlen
strbuf_addch
strbuf_addstr

```

Γ'.2 Τελικός Απολογισμός

Μετά από μακρά περίοδο δοκιμών, αποδείχθηκε ότι η υλοποίηση του εργαλείου μπορεί να υπηρετήσει τον σκοπό ενός proof-of-concept, χωρίς να υπάρχει η δυνατότητα να χρησιμοποιηθεί σε μεγάλη κλίμακα. Ο κύριος περιορισμός αποδείχθηκε πως είναι ο μεγάλος επιπρόσθετος αριθμός I/O στη διαδικασία της μεταγλώττισης, αλλά και η χρήση κάποιων μεθόδων χτισίματος κάποιων project μεγάλης κλίμακας όπως τα recursive makefiles.

Αν και δεν είναι ξεκάθαρο πως θα μπορούσε να μετρηθεί η διαφορά της επίδοσης του GCC ανά πέρασμα βελτιστοποίησης με το σύστημα ενεργό ή ανενεργό, είναι εύκολο να εξηγηθεί η αύξηση του συνολικού χρόνου μεταγλώττισης.

Όπως εξηγήθηκε στο προηγούμενο κεφάλαιο, ο collector τροποποιεί την συμπεριφορά του compiler, αναγκάζοντάς τον να εκτελεί μια συνάρτηση γραμμένη στη γλώσσα Python σε κάθε πέρασμα βελτιστοποίησης. Είναι γνωστό ότι η προγραμματιστική γλώσσα Python δεν έχει υψηλή απόδοση σε διάφορα σενάρια, λόγω του stack-based bytecode interpreter που χρησιμοποιεί [9]. Άρα, ακόμη και αν ένα πέρασμα δεν αξιοποιήσει την επιπλέον λειτουργικότητα της επιπλέον συνάρτησης, μόνο η κλήση του interpreter είναι αρκετή για να προσθέσει επιπλέον χρόνο ανά πέρασμα βελτιστοποίησης.

Εμπειρικά διαπιστώθηκε ότι ακόμη και για ένα πρόγραμμα μερικών δεκάδων γραμμών, ο GCC μπορεί να 'σπάσει' την διαδικασία βελτιστοποίησης σε κομμάτια, δηλαδή να εφαρμόσει τα περάσματα βελτιστοποίησης για ένα υποσύνολο του αρχικού προγράμματος, και έπειτα να τα εφαρμόσει ξανά για ένα άλλο υποσύνολο, προσθέτοντας ακόμη περισσότερο χρόνο στη διάρκεια της μεταγλώττισης, εάν το σύστημα είναι ενεργό.

Ακόμη, το σύστημα μπορεί να λειτουργήσει μόνο εάν ο GCC εκτελεί την διαδικασία της μεταγλώττισης με ένα thread. Διαπιστώθηκε εμπειρικά ότι εάν ο GCC εκτελεί το compilation με πολλά threads, αυτά τα threads μπορεί να χρησιμοποιούν τα ίδια UUIDs ως ονόματα προσωρινών μεταβλητών αποθήκευσης, όμως να αποθηκεύουν διαφορετικά gimple statements ανά UUID, καθιστώντας τα αποτελέσματα του συστήματος ανακριβή. Φυσικά, κάτι τέτοιο έχει ιδιαίτερα αρνητική επίπτωση στην ταχύτητα μεταγλώττισης λογισμικού που υποστηρίζει τέτοιου είδους παραλληλία.

Ενδεικτικά, παρακάτω παρουσιάζεται η επίπτωση στην ταχύτητα μεταγλώττισης του προγράμματος Nginx. Ο αρχικός χρόνος μεταγλώττισης όπως μετρήθηκε με το εργαλείο GNU Time είναι:

```
real    0m26.339s
user    0m23.692s
sys     0m2.855s
```

Ενώ με την χρήση του Scrubj ο χρόνος μεταγλώττισης είναι σχεδόν δυόμιση φορές μεγαλύτερος, δεν είναι αρκετό να μετρηθεί μόνο ο χρόνος της μεταγλώττισης αλλά και ο χρόνος που χρειάζεται ο receiver για να εισάγει τα τελικά δεδομένα στην βάση δεδομένων. Ο χρόνος μεταγλώττισης με το Scrubj είναι:

```
real    1m2.770s
user    0m30.473s
sys     0m6.605s
```

Ο χρόνος ολοκλήρωσης της εκτέλεσης του receiver είναι:

```
real    5m43.951s
user    0m7.628s
sys     0m2.939s
```

Στην αρχή του κεφαλαίου έγινε αναφορά στους αρχικούς στόχους σχεδίασης του συστήματος. Από τα παραπάνω είναι φανερό πως το εν λόγω σύστημα έχει κάποιους σοβαρούς περιορισμούς. Παρόλα αυτά όμως το σύστημα κατάφερε να πετύχει τους τρεις από τους πέντε σχεδιαστικούς του στόχους.

Το σύστημα επεξεργάζεται και εξάγει πληροφορίες από κώδικα που είναι γραμμένος στην προγραμματιστική γλώσσα C, μια κατ' εξοχήν εδραιωμένη και καθιερωμένη γλώσσα προγραμματισμού. Επιπρόσθετα, το σύστημα είναι αγνωστικιστικό ως προς τον κώδικα που επεξεργάζεται, δηλαδή μπορεί να επεξεργαστεί οποιοδήποτε ορθό πρόγραμμα γραμμένο στην C, καθώς βασίζεται στο front end και τις ενδιάμεσες αναπαραστάσεις του GCC για να εξάγει τα ζητούμενα δεδομένα. Ένα ακόμη θετικό σημείο του συστήματος είναι ότι είναι εύκολο στη χρήση, υποθέτοντας ότι ο μέσος χρήστης του συστήματος έχει ένα βασικό επίπεδο κατάρτισης. Το μόνο που χρειάζεται για να ενσωματωθεί το σύστημα στη διαδικασία της μεταγλώττισης, δοσμένου ότι όλα τα υπόλοιπα δομικά στοιχεία του συστήματος είναι παρόντα, είναι η χρήση μερικών flag του GCC.

Οι δύο σχεδιαστικοί στόχοι που δεν επιτεύχθηκαν ήταν αυτοί της μη σημαντικής αύξησης του χρόνου μεταγλώττισης, όπως εξηγήθηκε παραπάνω, και της δυνατότητας ενσωμάτωσης αυτού του συστήματος σε κάποια διαδικασία αυτόματης συλλογής κώδικα. Όσο αναφορά τον

τελευταίο στόχο, είναι δύσκολο το παρόν σύστημα να ενσωματωθεί σε μια τέτοια διαδικασία, καθώς τα περισσότερα μη ασήμαντα προγραμματιστικά project συνήθως χρησιμοποιούν προγράμματα αυτόματης παραγωγής αρχείων χτισίματος, καθιστώντας δύσκολη την τοποθέτηση των απαραίτητων flags στο κατάλληλο σημείο για κάθε διαφορετικό project.

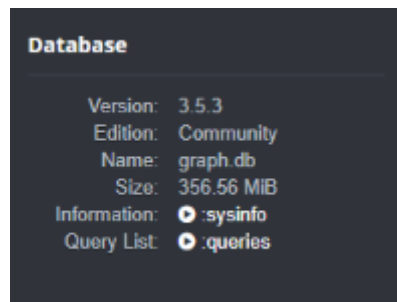
Τελικώς είναι σημαντικό να αναφερθεί ότι κάποια πραγματικά προγραμματιστικά project σε γλώσσα C κατάφεραν να χτιστούν με αυτό αυτό το σύστημα ενεργό. Τα εν λόγω project είναι τα εξής:

- Το OpenSSH, ένα πασίγνωστο πρόγραμμα απομακρυσμένης διαχείρισης [20].
- Ο Nginx, ένας ευρέως διαδεδομένος διαδικτυακός διακομιστής [36].
- Ο jq, ένας command-line JSON processor [19].
- Το OpenVPN, ένα εδραιωμένο εργαλείο υλοποίησης εικονικών ιδιωτικών δικτύων [37].
- Το git, το πιο διαδεδομένο κατανεμημένο σύστημα version control [60].

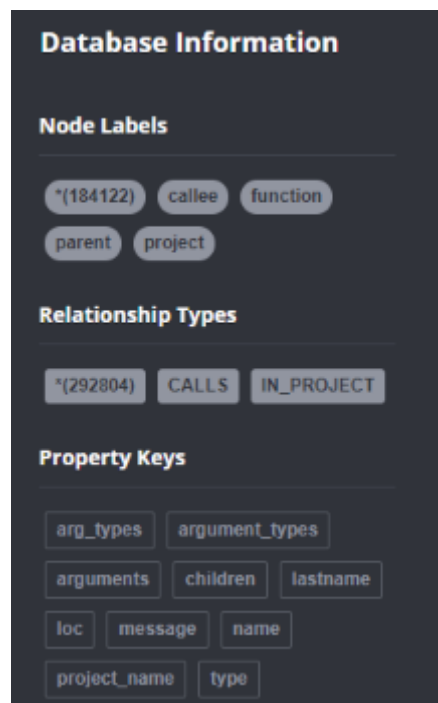
Με την χρήση του εργαλείου cloc [3] μπορεί να γίνει μέτρηση των γραμμών κώδικα σε C ανά πρόγραμμα.

- Το OpenSSH έχει 110138 γραμμές κώδικα.
- Ο Nginx έχει 142480 γραμμές κώδικα.
- Ο jq έχει 32353 γραμμές κώδικα.
- Το OpenVPN έχει 77247 γραμμές κώδικα.
- Το git έχει 232611 γραμμές κώδικα.

Μετά το πέρας της μεταγλώττισης των άνωθεν προγραμμάτων με συνολικό αριθμό γραμμών κώδικα ίσο με 594829 γραμμές, το μέγεθος της βάσης δεδομένων στη Neo4j ήταν περίπου 356 megabyte, όπως φαίνεται παρακάτω:



Αντίστοιχα, ο γράφος αποτελούνταν από 184122 κόμβους, οι οποίοι διασυνδέονταν με 292804 σχέσεις, όπως φαίνεται παρακάτω:



Το συνολικό μέγεθος των αποθετηρίων των παραπάνω project ήταν 58.6 megabyte. Δεδομένου του ότι πολλά αποθετήρια περιέχουν αρχεία τα οποία δεν συμμετέχουν στην διαδικασία της μεταγλώττισης, είναι ενθαρρυντικό γεγονός ότι η βάση δεδομένων κατέληξε να έχει μεγαλύτερο όγκο δεδομένων από τα αρχικά δεδομένα που χρησιμοποιήθηκαν για να την κατασκευάσουν. Ίσως αυτό να σημαίνει ότι στο μέλλον, ένα σύστημα το οποίο δεν έχει τα τρωτά σημεία του Scrubj και έχει την δυνατότητα να κλιμακώνεται ομαλά, θα μπορέσει να επιτύχει μεγάλης κλίμακας και μεγάλου όγκου αποθήκευση και επεξεργασία κώδικα ως δεδομένα.

Βιβλιογραφία

- [1] Atul Adya, Barbara Liskov, and Patrick O’Neil. «Generalized isolation level definitions». In: *Proceedings of 16th International Conference on Data Engineering (Cat. No. 00CB37073)*. IEEE. 2000, pp. 67–78.
- [2] Alfred V Aho, Jeffrey D Ullman, and Ravi Sethi. *Compilers: principles, techniques and tools*. Addison-Wesley, 1986.
- [3] AlDanial. *cloc counts blank lines, comment lines, and physical lines of source code in many programming languages*. 2020. URL: <https://github.com/AlDanial/cloc> (visited on 05/30/2020).
- [4] Miltiadis Allamanis et al. «Mining semantic loop idioms from Big Code». In: *Technical Report*. 2016.
- [5] Renzo Angles. «A comparison of current graph database models». In: *2012 IEEE 28th International Conference on Data Engineering Workshops*. IEEE. 2012, pp. 171–177.
- [6] Renzo Angles and Claudio Gutierrez. «Survey of graph database models». In: *ACM Computing Surveys (CSUR)* 40.1 (2008), pp. 1–39.
- [7] Renzo Angles et al. «Foundations of modern query languages for graph databases». In: *ACM Computing Surveys (CSUR)* 50.5 (2017), pp. 1–40.
- [8] The ZeroMQ authors. *ZeroMQ / Get started*. 2020. URL: <https://zeromq.org/get-started/> (visited on 05/21/2020).
- [9] Gergő Barany. «Python interpreter performance deconstructed». In: *Proceedings of the Workshop on Dynamic Languages and Applications*. 2014, pp. 1–9.
- [10] Jürgen Bitzer, Wolfram Schrettl, and Philipp JH Schröder. «Intrinsic motivation in open source software development». In: *Journal of Comparative Economics* 35.1 (2007), pp. 160–169.

- [11] Marcel Bogers and Joel West. «Managing distributed innovation: Strategic utilization of open and user innovation». In: *Creativity and innovation management* 21.1 (2012), pp. 61–75.
- [12] Andrea Bonaccorsi and Cristina Rossi. «Why open source software can succeed». In: *Research policy* 32.7 (2003), pp. 1243–1258.
- [13] Rick Cattell. «Scalable SQL and NoSQL data stores». In: *Acm Sigmod Record* 39.4 (2011), pp. 12–27.
- [14] Min Chen, Shiwen Mao, and Yunhao Liu. «Big data: A survey». In: *Mobile networks and applications* 19.2 (2014), pp. 171–209.
- [15] Keith Cooper and Linda Torczon. *Engineering a compiler*. Elsevier, 2011.
- [16] Christopher John Date. *SQL and relational theory: how to write accurate SQL code*. " O'Reilly Media, Inc.", 2011.
- [17] Jeffrey Dean and Sanjay Ghemawat. «MapReduce: Simplified data processing on large clusters». In: (2004).
- [18] David DeWitt and Jim Gray. «Parallel database systems: the future of high performance database systems». In: *Communications of the ACM* 35.6 (1992), pp. 85–98.
- [19] Stephen Dolan. *jq*. 2020. URL: <https://stedolan.github.io/jq/> (visited on 05/21/2020).
- [20] OpenBSD Foundation. *OpenSSH*. 2020. URL: <https://www.openssh.com/> (visited on 05/21/2020).
- [21] The Linux Foundation. *2017 Linux Kernel Development Report*. 2017. URL: <https://www.linuxfoundation.org/publications/2017/10/2017-state-of-linux-kernel-development/> (visited on 04/01/2020).
- [22] Armando Fox et al. «Cluster-based scalable network services». In: *Proceedings of the sixteenth ACM symposium on Operating systems principles*. 1997, pp. 78–91.
- [23] Nadime Francis et al. «Cypher: An evolving query language for property graphs». In: *Proceedings of the 2018 International Conference on Management of Data*. 2018, pp. 1433–1445.

- [24] Amir Gandomi and Murtaza Haider. «Beyond the hype: Big data concepts, methods, and analytics». In: *International journal of information management* 35.2 (2015), pp. 137–144.
- [25] John Gantz and David Reinsel. «Extracting value from chaos». In: *IDC iview* 1142.2011 (2011), pp. 1–12.
- [26] GitHub. *The State of the OCTOVERSE*. 2019. URL: <https://octoverse.github.com/> (visited on 04/01/2020).
- [27] Theo Haerder and Andreas Reuter. «Principles of transaction-oriented database recovery». In: *ACM computing surveys (CSUR)* 15.4 (1983), pp. 287–317.
- [28] Mark Harman. «Why source code analysis and manipulation will always be important». In: *2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*. IEEE. 2010, pp. 7–19.
- [29] Red Hat. *The State of Enterprise Open Source*. 2020. URL: <https://www.redhat.com/en/enterprise-open-source-report/2020> (visited on 04/01/2020).
- [30] Pieter Hintjens. *0MQ - The Guide*. 2020. URL: <http://zguide.zeromq.org/page:all> (visited on 05/21/2020).
- [31] Charles A Hoare. *Hints on programming language design*. Tech. rep. STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1973.
- [32] Charles Antony Richard Hoare. «An axiomatic basis for computer programming». In: *Communications of the ACM* 12.10 (1969), pp. 576–580.
- [33] John Howard et al. «Scale and performance in a distributed file system». In: *ACM SIGOPS Operating Systems Review* 21.5 (1987), pp. 1–2.
- [34] Hongcheng Huang and Ziyu Dong. «Research on architecture and query performance based on distributed graph database neo4j». In: *2013 3rd International Conference on Consumer Electronics, Communications and Networks*. IEEE. 2013, pp. 533–536.
- [35] Neo Technology Inc. *Bolt Protocol*. 2020. URL: <https://boltprotocol.org/> (visited on 05/21/2020).
- [36] Nginx Inc. *nginx*. 2020. URL: <https://nginx.org/en/> (visited on 05/21/2020).
- [37] OpenVPN Inc. *OpenVPN is an open source VPN daemon*. 2020. URL: <https://github.com/OpenVPN/openvpn> (visited on 05/21/2020).

- [38] Pankaj Jalote. *An integrated approach to software engineering*. Springer Science & Business Media, 2012.
- [39] Butler W Lampson. «. Atomic transactions». In: *Distributed Systems—Architecture and Implementation*. Springer, 1981, pp. 246–265.
- [40] Michael Larabel. *GCC Soars Past 14.5 Million Lines Of Code & I’m Real Excited For GCC 5*. 2015. URL: https://www.phoronix.com/scan.php?page=news_item&px=MTg30TQ (visited on 04/01/2020).
- [41] Michael Larabel. *The Linux Kernel Has Grown By 225k Lines of Code So Far This Year From 3.3k Developers*. 2018. URL: https://www.phoronix.com/scan.php?page=news_item&px=Linux-September-2018-Stats (visited on 04/01/2020).
- [42] Josh Lerner and Jean Tirole. «Some simple economics of open source». In: *The journal of industrial economics* 50.2 (2002), pp. 197–234.
- [43] Jon Loeliger and Matthew McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development*. " O’Reilly Media, Inc.", 2012.
- [44] David Loshin. *Big data analytics: from strategic planning to enterprise integration with tools, techniques, NoSQL, and graph*. Elsevier, 2013.
- [45] David Malcolm. *GCC plugin that embeds CPython inside the compiler*. 2020. URL: <https://github.com/davidmalcolm/gcc-python-plugin> (visited on 05/21/2020).
- [46] Grzegorz Malewicz et al. «Pregel: a system for large-scale graph processing». In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 2010, pp. 135–146.
- [47] James Manyika. «Big data: The next frontier for innovation, competition, and productivity». In: http://www.mckinsey.com/Insights/MGI/Research/Technology_and_Innovation/Big_data_The_next_frontier_for_innovation (2011).
- [48] Claudio Martella et al. *Practical graph analytics with apache giraph*. Vol. 1. Springer, 2015.
- [49] Jason Merrill. «Generic and gimple: A new tree representation for entire functions». In: *Proceedings of the 2003 GCC Developers’ Summit*. Citeseer. 2003, pp. 171–179.
- [50] ABM Moniruzzaman and Syed Akhter Hossain. «Nosql database: New era of databases for big data analytics-classification, characteristics and comparison». In: *arXiv preprint arXiv:1307.0191* (2013).

- [51] Neo4j. *The Neo4j Operations Manual v4.0*. 2020. URL: <https://neo4j.com/docs/operations-manual/4.0/> (visited on 04/22/2020).
- [52] nosql-database.org. *NOSQL DEFINITION*. 2009. URL: <https://nosql-database.org/> (visited on 04/02/2020).
- [53] Bruce Perens et al. «The open source definition». In: (1999), pp. 171–188.
- [54] Martin Lucina Pieter Hintjens Martin Sustrik. *zmq_ipc(7) - 0MQ Api*. 2020. URL: <http://api.zeromq.org/3-2:zmq-ipc> (visited on 05/21/2020).
- [55] GNU Project. *Control Flow (GNU Compiler Collection (GCC) Internals)*. 2020. URL: <https://gcc.gnu.org/onlinedocs/gccint/Control-Flow.html> (visited on 04/25/2020).
- [56] GNU Project. *GCC, the GNU Compiler Collection*. 2020. URL: <https://gcc.gnu.org/> (visited on 04/25/2020).
- [57] GNU Project. *GENERIC (GNU Compiler Collection (GCC) Internals)*. 2020. URL: <https://gcc.gnu.org/onlinedocs/gccint/GENERIC.html> (visited on 04/25/2020).
- [58] GNU Project. *GIMPLE (GNU Compiler Collection (GCC) Internals)*. 2020. URL: <https://gcc.gnu.org/onlinedocs/gccint/GIMPLE.html> (visited on 04/25/2020).
- [59] GNU Project. *plugins - GCC Wiki*. 2020. URL: <https://gcc.gnu.org/wiki/plugins> (visited on 04/25/2020).
- [60] The Git Project. *git*. 2020. URL: <https://git-scm.com/about/> (visited on 05/21/2020).
- [61] G Putzolu and IL Traiger. «Granularity of Locks and Degrees of Consistency in a Shared Database». In: *IFIP Working Conference on Modelling in Data Base Management Systems*. 1976, pp. 365–394.
- [62] Veselin Raychev, Martin Vechev, and Andreas Krause. «Predicting program properties from "big code"». In: *ACM SIGPLAN Notices* 50.1 (2015), pp. 111–124.
- [63] Veselin Raychev, Martin Vechev, and Eran Yahav. «Code completion with statistical language models». In: *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2014, pp. 419–428.
- [64] Rebecca Russell et al. «Automated vulnerability detection in source code using deep representation learning». In: *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2018, pp. 757–762.

- [65] statista. *Projected revenue of open source services from 2017 to 2022*. 2018. URL: <https://www.statista.com/statistics/270805/projected-revenue-of-open-source-software-since-2008/> (visited on 04/02/2020).
- [66] Leslie G Valiant. «A bridging model for parallel computation». In: *Communications of the ACM* 33.8 (1990), pp. 103–111.
- [67] Xindong Wu et al. «Data mining with big data». In: *IEEE transactions on knowledge and data engineering* 26.1 (2013), pp. 97–107.